CMPT 355 Class Project

Team 6

Daniel Morris

Kevin Noonan

Qi Guo

Wei Zhang

Nam Pham

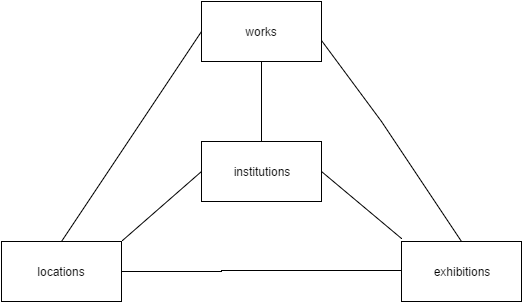
**Part A**.

Part A The new design for database

1.The goal of our design was to cover all the data from each individual databases, while keeping a clear database structure to keep queries and transactions in an easier form, as well as minimizing the changes each member need to make. Our newly designed database structure serves the need of combining 5 individual databases together to allow patron to monitor all five individual databases at the same time. Hence the patron can simply get data information from our newly created database instead of going to each of databases to get data. With respect to table relationships, we tried to maintain an intuitive relationship between two tables so that it doesn’t cause complex transactions. After several meetings, we agreed on keeping the following data information:

Works information, exhibition information, locations information, institution information and their relations information between each other.

Hence we then came up with the following database structure regard our design.



A0:Simplified e-r diagram for the over all new database

This new database shows a comprehensive information which covers each individual museum’s works(museums, items) locations, exhibitions and institutions. As illustrated in graph A0, the overall data flow is straightforward, the institutions hold all the works, locations and exhibitions. Works are held by the institutions also are placed in different locations, as well as showcased in some exhibitions. Of all the exhibitions, they are took placed in some locations, and both the locations and exhibitions are belong to some institutions.

1.1 works and corresponding relationships information

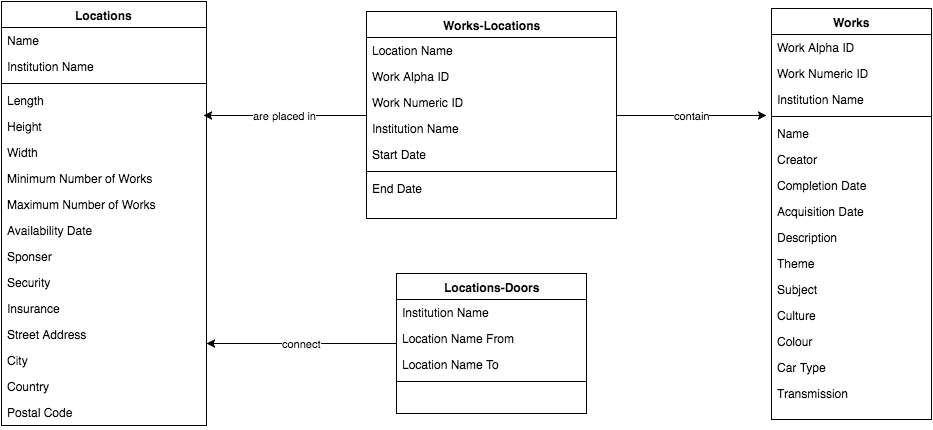
group project e-r parta works-worksvalue-worksmedia(v1).png

A1: diagram shows the works and corresponding relationships

When we try to design the new database structure, we realised that each member’s database has different data information for work’s categorizations. We decided to collect each member’s type and subtype together, then identifying the similarities and differences, trying to fit all the works into the new created set of categorization system.

And we create a information to store the value of works as well as a information to collect the medias of each work.

1.2 locations and corresponding relationships information



A2: Diagram show the location and corresponding relationships

Locations table stores information about the different locations. The connections between different locations are stored in location-door table. And the works also stored in different locations. The patron and the insurance company can also trace any changes of location for any from the relationship between the works and locations. That means this works-location relationship allow users to determine which location works have been to, are currently in, and will be in.

1.3 Exhibition and corresponding relationships

|  |  |
| --- | --- |
|  |  |
|  |  |

group project e-r parta exhibitions-exhibitionsworks-works (v1).png

A3. Diagram shows exhibitions and works relationship

This works-exhibition relationship allow the patron to retrieve the information of any works showcased in different exhibitions.

|  |  |
| --- | --- |
|  |  |
|  |  |

group project e-r parta exhibitons-exhibitionslocations-locations(v2).png

A4. Diagram shows the exhibitions and locations relationship

The exhibitions are took place in different locations. The patron can trace any changes of location for any exhibitions from the relationship between the exhibitions and locations. That means this exhibitions-locations relationship allow users to determine which location exhibitions have took place, are currently taking place, and will take place.

1.4 institution and corresponding relationships

group project e-r parta works-institutions-locations-exhibitions(v1).png

A5. Diagram shows the institution and corresponding relationships

The works are contained in different institution, and institutions also have different locations to store this works and to take place various exhibitions. Hence institutions have relationships with works, locations and exhibitions.

**Part B**.

The database is being made such that the history of the database is stored at all times. The current database design allows for the history of **work locations**, **~~work keepers~~**, works Owners, transactions **~~Temporary Exhibitions Locations~~, exhibitions**, **exhibition locations**, and **exhibition works.** Throughout the database design this historical data is made possible through the use of start dates and end dates. The start date is always necessary and the end date is optional based on whether or not the end date of the historical attribute is known. The following is a breakdown of how each historical structure will be implemented:

**Exhibitions:**  When exhibitions are created it is necessary that their start date is specified. This start date will be essential for storing the temporal location of the exhibition and the works showcased in the exhibition. The end date of an exhibition is not necessary because it is sometimes not planned for when an exhibition will end.

**~~Exhibit Locations:~~**  ~~Once an exhibition is made the location of the exhibition can be set using this table. The location of an exhibit will always be known. Since exhibitions can be either travelling or non-traveling this will need to be implemented into this table. If the exhibition is non-traveling the database will use the start date and end date of the exhibition for the start and end date of this exhibit location. If the exhibition is traveling, then the start date and end date of when the exhibition will be in the specified location will also need to be stated. When data is being imputed to this table the database will perform checks to ensure that conflicts do not occur in the database such as dates overlapping where they are not supposed to be. Also when a location for an exhibition is set or changed the database will make the necessary changes to all the works that have already been placed into the exhibition.~~

**Exhibit Locations:** Once an exhibition is made the location of the exhibition can be saved. The history of all exhibition’s locations will also be saved. This will be done by saving the start and end date of the time that a specific exhibition is in a specific location. When an exhibit’s location data is added to the database the database will also adapt the data for all the works in that exhibition to change location as well.

**Exhibit Works:**  Once an exhibition has been created in the museum then works can be scheduled to be placed into it. When works are entered into an exhibition the database will check to see if the exhibition has a specified location. If the location for the exhibition is specified, then the database will ~~update the works locations table~~ save the location of the work to that specific location. ~~to account for where and when the works will be in the location specified~~.

https://cdn.discordapp.com/attachments/253682960379674629/255611808256098304/group_project_e-r_partb_exhibitions-locations_exhibitions_works-exhibitionsv2.png

Shown above is a diagram relating exhibitions to their works and locations. Every exhibition in the database can hold works and will be placed in locations so these tables will be related. The diagram shows that the locations and works exist outside of the exhibitions table. This means that the works and locations will need to store their individual start and end dates to save their historical data.

**Work Locations:**  The location of a work will always be known. The ~~works locations table~~ database will contain the start date-time and end date-time for a work’s location. A work’s location is necessary to be tracked at a minute to minute basis due to insurance reasons. A work’s location can be either inserted manually or automatically. As previously stated the works location will be automatically updated when the work is in an exhibition with a specified location.

https://cdn.discordapp.com/attachments/253682960379674629/255611813989842945/group_project_e-r_partb_works-locations_locations_exhibitions-locationsv2.png

Shown above is a diagram relating locations to their works and exhibitions. Every location in the database can hold works and exhibitions so these tables will be related. The diagram shows that the exhibitions and works exist outside of the location table. This means that the works and exhibitions will need to store their individual start and end dates to save their historical data.

https://cdn.discordapp.com/attachments/253682960379674629/255611815705182208/group_project_e-r_partb_works-locations_works_works-exhibitionsv2.png

Shown above is a diagram relating works to their locations and exhibitions. Every work in the database can be held in locations and exhibitions so these tables will be related. The diagram shows that the exhibitions and locations exist outside of the works table. This means that the locations and exhibitions will need to store their individual start and end dates of when the works are in them to save their historical data.

**Work Owners:** The owner of a work will always be known. The database will know the start and end date of when each of the work’s owners owned the work. When one of the institutions in the database buys or sells a work this data will be automatically updated.

**Transactions:** Since it is important to record all of the databases transactions it is crucial that the date of a transaction is stored. Whenever a transaction has been inserted to the database a new entry to this table will be made. This attribute will store data on the transaction including the date the transaction was made.

https://cdn.discordapp.com/attachments/253682960379674629/255611812685283328/group_project_e-r_partb_works_transactions_works-ownershipsv2.png

Shown above is a diagram relating works to their transactions and ownership tables. The works table does not store any of its historical data because it will never be updated. The transactions table stores the date of transaction in order to recover past transactions and the ownership table records the start date and end date for each of a works owner.

**~~Work Keepers:~~** ~~The keeper of a work will always be known for he can be the lender or the borrower of the item. Since the keeper of a work can change it is necessary for insurance reasons that the history of all works is recorded. When a keeper of a work is defined it is necessary that the start date is specified but the end date is optional since it is often not known when a keeper will not be responsible for a work anymore.~~

**~~Temporary Exhibitions Locations:~~** ~~When travel exhibitions are created, it is necessary that~~

~~their start date is specified. The temporary location always need to keep track the data of the sponsor, security, the insurance value and the address when it is on the travelling because when a temporary can be used multiple times in many exhibitions, its other data can also be changed. So, we would want to keep track of the history of being used of a temporary location.~~

Through further analysis of the database it has become apparent that our preliminary

database design does not account for some other data that could also be ~~temporal~~ stored historically. This data includes the state of the work and the insurance value of the work. The state of the work

includes things such as “In good condition”, “Stolen”, “Damaged” etc. Since the state of the

work could change it is important that all the previous states of each work are recorded. It is

also apparent that the insurance value of a work could change as well. It would be nice to see

the previous value for a work so that functionality will be added to the database. The following is a breakdown of how each temporal structure will be implemented:

**Work State:** When a work is first entered into the museum it’s state will also be inputted presumably as “In good condition”. The start date of the work’s state will be set to the date of acquisition of the work. If the state of the work changes then the end date of that state will be saved and a new state for that work will be saved.

**Work Insurance Value:** When a work is first entered into the museum it’s insurance value will also be inputted. The start date of this value will be set to the date of acquisition of the work. If the insurance value of the work changes then the end date of that value will be saved and a new value for that work will be saved.

e-r Works WorksState WorksInsuranceValue.png

Shown above is the a diagram of how the state and insurance value of a work can be stored historically. The state and value of the works are separated into their own tables that include a start date and an end date. These dates will determine what time period the specific work is in for a specific state and insurance value.

~~Adding the functionality for these two extra temporal data types requires the creation~~

~~of two more tables that relate to works. The following is a diagram of how that relationship would look. The new tables are highlighted to easily show where they fit into the whole database design.~~

**Part C**.

This part of the report will be split up into ~~two~~ three sections. The first section will be describing our overall design. The ~~first~~ second section will be describing the purposes of the domains we have chosen for the new database structure. The ~~second~~ third section will be describing the tables and attributes and their individual purposes and design.

**Overall Design:**

group project e-r partc high level with ns without fk(v5).png

group project e-r partc with ns prefix without fk(v16).png

These are the overall high-level and low-level design that we come up with for our team database. We added ns prefix before each table name so that there won’t be any conflicts with our existing tables when we create tables in our own database.

**Institutions** table is the key table for the entire database structure. It is a newly added table for each team member that contains all information regarding different museums that do transactions with our museums.

Since we all have **Works** table in our individual database but with different type, subtype and custom type, we decide to add six new attributes, namely, theme, subject, culture, colour, car type and transmission to team works table, so that all data are stored in individual database.

**Works Media** table is a secondary table to store different materials associated to all works.

**Works State** table is a newly added table comparing to individual database. We decide to add this table so that we can keep track of work state, such as stolen, damaged, restoration, in good condition from time to time.

**Works Value** table is a new table for everyone, which stores changes in insurance value for each work.

**Exhibitions** table contains both in-museum exhibitions and travelling exhibitions. A new boolean attribute, is travelling, is added to the table to distinguish between those two types of exhibitions.

There are major updates in **Locations** comparing to individual database. We add availability date, sponsor, security, insurance, street address, city, country subdivision, country and postal code to this table in order to keep track of information of travelling exhibitions. We not only store each city that the traveling exhibition reaches, but also all the in between locations. For example, we use “city 1-city 2” to represent the exhibition is going between locations but has not arrived to the destination city 2 yet.

**Exhibitions** **Locations** table stores location information for each exhibition. It contains exhibition start date and end date, which are identical to the dates in Exhibitions table. If the current exhibition is of type travelling, traveling start date and end date would be the moment that it reaches different locations. We keep track of the one week packing up, transportation, and set-up time period when traveling exhibitions go between locations for security reasons.

**Locations** **Doors** table contains all doors connecting different locations in every team member’s museum. All doors are one way to avoid redundant data.

**Works** **Ownership** table keeps track of the changes in works ownership once the works are sold or bought. We decide to store works ownership information for potentially borrowed works and long term borrowed works so that we can tell all the borrowed works from owned works.

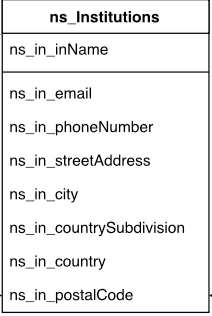
**Transactions** table is an important temporal table in our database. All information concerning buying, selling, borrowing in, borrowing out, lending in, lending out are stored in this table. We have two attributes giver name and receiver name in this table to distinguish seller, buyer, borrower or lender.

**DOMAIN:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Domain*** | ***Type*** | ***Attributes that uses this domain*** | ***Description*** |
| ns\_inName | varchar(50) | ns\_in\_inName  ns\_wk\_inName  ns\_lc\_inName  ns\_ex\_inName  ns\_tr\_inName  ns\_tr\_inNameReceiver  ns\_tr\_inNameGiver  ns\_wo\_inName  ns\_wo\_inNameOwner  ns\_wl\_inName  ns\_we\_inName  ns\_el\_inName  ns\_wv\_inName  ns\_ws\_inName  ns\_wm\_inName  ns\_ld\_inName | This domain is the type for the name of the institutions that the museum will associate to, including the museum itself. Because institution’s name exists in most of the tables and act as a part of the primary key and foreign key in those tables. And because the longest institution name from the team members is 50, so, this domain restriction will be 50. |
| ns\_email | varchar(100) | ns\_in\_email | This domain is the type for attribute that is the email of the institutions for contact. It will only be used once in the institution table. And because some possible email address can be 100 characters, so, this domain restriction will be 100. |
| N ns\_phoneNumber | varchar(20) | ns\_in\_phoneNumber | This domain is the type for attribute that is the phone number of the institutions for contact. It will only be used once in the institution table. It is a set of string rather than a regular number. And because the phone number form can have at most 20 characters, so, this domain restriction will be 20. |
| ns\_streetAddress | varchar(50) | ns\_in\_streetAddress  ns\_lc\_StreetAddress | The domain type for the street address of the institution which is a part of its contact information. Because the locations for travelling exhibitions also possess this trait, the location table will also use this domain. And because the longest street address from the team members is 50, so, this domain restriction will be 50. |
| ns\_city | varchar(60) | ns\_in\_city  ns\_lc\_city | The domain type for the city of the institution which is a part of its contact information. Because the locations for travelling exhibitions also possess this trait, the location table will also use this domain. And because the longest city name on the world is 58, so, this domain restriction will be 60. |
| ns\_countrySubdivision | varchar(20); | ns\_in\_countrySubdivision  ns\_lc\_countrySubdivision | The domain type for the state/province of the institution which is a part of its contact information. Because the locations for travelling exhibitions also possess this trait, the location table will also use this domain. And because the longest country name from the team members is 20, so, this domain restriction will be 20. |
| ns\_country | varchar(20) | ns\_in\_country  ns\_lc\_country | The domain type for the country of the institution which is a part of its contact information. Because the locations for travelling exhibitions also possess this trait, the location table will also use this domain. And because the longest country name from the team members is 20, so, this domain restriction will be 20. |
| ns\_postalcode | varchar(50) | ns\_in\_postalCode  ns\_lc\_postalCode | The domain type for the postal code of the institution which is a part of its contact information. Because the locations for travelling exhibitions also possess this trait, the location table will also use this domain. And because there will be longer postal code in the future, this domain restriction will be 50. |
| ns\_IDAlpha | varchar(10) | ns\_wk\_IDAlpha  ns\_tr\_IDAlpha  ns\_tr\_IDAlphaOther  ns\_wo\_IDAlpha  ns\_wl\_IDAlpha  ns\_we\_IDAlpha  ns\_wv\_IDAlpha  ns\_ws\_IDAlpha  ns\_wm\_IDAlpha | The domain for the id-alpha, which is one part of the identification key for the items. The attribute using this domain will all be on the part of the primary key and foreign key in theirs, so all tables that associate with the items will use this domain. And because the longest id-alpha from the team members is 10, so, this domain restriction will be 10. |
| ns\_IDNumeric | interger | ns\_wk\_IDNumeric  ns\_tr\_IDNumeric  ns\_tr\_ns\_IDNumericOther  ns\_wo\_IDNumeric  ns\_wl\_IDNumeric  ns\_we\_IDNumeric  ns\_wv\_IDNumeric  ns\_ws\_IDNumeric  ns\_wm\_IDNumeric | The domain for the id-number, which is one part of the identification key for the items. The attribute using this domain will all be on the part of the primary key and foreign key in theirs, so all tables that associate with the items will use this domain. And one of the member used long integer, therefore this domain type will be integer. |
| ns\_wkName | varchar(200) | ns\_wk\_wkName | This domain is the type for the work name. It will only be used in the work table. And because the longest work name from the team members is 200, so, this domain restriction will be 200. |
| ns\_creator | varchar(50) | ns\_wk\_creator | This domain is the type for the creator of the work’s name. It will only be used in the work table. And because the longest creator name from the team members is 50, so, this domain restriction will be 50. |
| ns\_coDate | date | ns\_wk\_coDate | This domain is the type for the completion date of the work. It will only be used in the work table. |
| ns\_acDate | date | ns\_wk\_acDate | This domain is the type for the date the museum acquire work. It will only be used in the work table. |
| ns\_wkDescription | text | ns\_wk\_wkDescription | This domain is the type for the work name. It will only be used in the work table. And its type will be a text. |
| ns\_theme | varchar(50) | ns\_wk\_theme | This domain is the type for the theme of the work. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. Because the longest theme from the team members is 50, so, this domain restriction will be 50. |
| ns\_subject | varchar(50) | ns\_wk\_subject | This domain is the type for the subject of the work. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. Because the longest subject from the team members is 50, so, this domain restriction will be 50. |
| ns\_culture | varchar(50) | ns\_wk\_culture | This domain is the type for the culture of the work. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. Because the longest culture from the team members is 50, so, this domain restriction will be 50. |
| ns\_colour | varchar(50) | ns\_wk\_colour | This domain is the type for the colour of the work. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. This domain restriction will be 50 to mark the same as others type/subtype. |
| ns\_carType | varchar(50) | ns\_wk\_carType | This domain is the type for the car type of the work because one member has a car museum. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. This domain restriction will be 50 to mark the same as others type/subtype. |
| ns\_transmission | varchar(50) | ns\_wk\_transmission | This domain is the type for the car transmission type of the work because one member has a car museum. It will only be used in the work table. This domain was used to be called type/subtype, but due to the combination of team members database, it need to be changed to something more meaningful. This domain restriction will be 50 to mark the same as others type/subtype. |
| ns\_lcName | varchar(58) | ns\_lc\_lcName  ns\_wl\_lcName  ns\_el\_lcName  ns\_ld\_lcName1  ns\_ld\_lcName2 | This domain is the type for the name of the locations that the museum has. Because the location name will act as a part of the primary key and in location table, the other tables that associate with the locations information will also use this domain for its attribute. And because the longest location name from the team members is 58, so, this domain restriction will be 58. |
| ns\_length | real | ns\_lc\_length | This domain is the type for the length of the museum’s location. It will only be used in the location table. |
| ns\_height | real | ns\_lc\_height | This domain is the type for the height of the museum’s location. It will only be used in the location table. |
| ns\_width | real | ns\_lc\_width | This domain is the type for the width of the museum’s location. It will only be used in the location table. |
| ns\_minNumWorks | smallint | ns\_lc\_minNumWorks | This domain is the type for the minimum work capacity of the museum’s location. It will only be used in the location table. |
| ns\_maxNumWorks | smallint | ns\_lc\_maxNumWorks | This domain is the type for the maximum work capacity of the museum’s location. It will only be used in the location table. |
| ns\_availabilityDate | date | ns\_lc\_availabilityDate | This domain is the type for the date that the museum will be available. It will only be used in the location table. |
| ns\_sponsor | varchar(50) | ns\_lc\_sponsor | This domain is the type for the sponsor of the traveling exhibition, which will use the temporary locations. It will only be used in the location table. |
| ns\_security | varchar(50) | ns\_lc\_security | This domain is the type for the security of the traveling exhibition, which will use the temporary locations. It will only be used in the location table. |
| ns\_insurance | integer | ns\_lc\_insurance  ns\_wv\_InsuranceValue | This domain is the type for the insurance of the traveling exhibition, which will use the temporary locations, so it will be used in the location table. However, it can also be used to show the value of the item, therefore it will also be used in work value table. |
| ns\_exName | varchar(100) | ns\_ex\_exName  ns\_we\_exName  ns\_el\_exName | This domain is the type for the name of the exhibitions that the museum held. Because the exhibition names will act as a part of the primary key in exhibition table, the other tables that associate with the exhibitions information will also use this domain for its attribute. And because the longest exhibition name from the team members can be 100, so, this domain restriction will be 100. |
| ns\_isTraveling | boolean | ns\_ex\_isTraveling | This domain is the special type for the attribute of the exhibition table to tell whether the exhibition is a traveling one or not. It will only be used in the exhibition table. |
| ns\_exStartDate | date | ns\_ex\_exStartDate  ns\_we\_exStartDate  ns\_el\_exStartDate | This domain is the type for the date that the exhibitions will be held. Because the exhibition starting date will act as a part of the primary key in exhibition table, the other tables that associate with the exhibitions information will also use this domain for its attribute. |
| ns\_exDescription | text | ns\_ex\_exDescription | This domain is the type for the exhibition description. It will only be used in the exhibition table. And its type will be a text. |
| ns\_exEndDate | date | ns\_ex\_exEndDate  ns\_we\_exEndDate  ns\_el\_exEndDate | This domain is the type for the date that the exhibitions will end. Though the exhibition end date won’t act as a part of the primary key in exhibition table, the other tables that associate with the exhibitions information will still use this domain for its attribute |
| ns\_travelingStartDate | date | ns\_el\_travelingStartDate | This domain is the type for the travelling exhibition start date. It will only be used in the Exhibitions\_ location table. |
| ns\_travelingEndDate | date | ns\_el\_travelingEndDate | This domain is the type for the travelling exhibition end date. It will only be used in the Exhibitions\_ location table. |
| ns\_trDate | date | ns\_tr\_trDate | This domain is the type for the dates of the transactions that the museum makes. It will only be used in the Transactions table. |
| ns\_trType | varchar(20); | ns\_tr\_trType | This domain is the type for the types of the transactions that the museum makes, including loan, borrow, purchase and sell. It will only be used in the Transactions table. |
| ns\_woStartDate | date | ns\_wo\_woStartDate | This domain is the type for the date that the new owner obtains the item. It will only be used in the Works\_Ownership table. |
| ns\_woEndDate | date | ns\_wo\_woEndDate | This domain is the type for the last date that the previous owner has the item. It will only be used in the Works\_Ownership table. |
| ns\_wlStartDateTime | timestamp | ns\_wl\_wlStartDateTime | This domain is the type for the date that the work move into the location. It will only be used in the Works\_Locations table. Moreover, it is a timestamp type to record exactly the date and time the works move into the locations. |
| ns\_wlEndDateTime | timestamp | ns\_wl\_wlEndDateTime | This domain is the type for the date that the work move out of the location. It will only be used in the Works\_Locations table. Moreover, it is a timestamp type to record exactly the date and time the works move out of the locations. |
| ns\_wvStartDate | date | ns\_wv\_startDate | This domain is the type for the date that the item get the new insurance value. It will only be used in the Works\_Value table. |
| ns\_wvEndDate | date | ns\_wv\_endDate | This domain is the type for the last day that the item has the old insurance value. It will only be used in the Works\_Value table. |
| ns\_state | varchar(30) | ns\_ws\_state | This domain is the type show the state of the work in a period, including: in good condition, stolen, damaged, and restoration. It will only be used in the Works\_State table. |
| ns\_wsStartDate | date | ns\_ws\_wsStartDate | This domain is the type for the date that the item start its current state. It will only be used in the Works\_State table. |
| ns\_wsEndDate | date | ns\_ws\_wsEndDate | This domain is the type for the date that the item end its current state. It will only be used in the Works\_State table. |
| ns\_material | varchar(100) | ns\_wm\_material | This domain is the type for the materials of the works. It will only be used in the Works\_Media table. |

**TABLES**

-- (1) **ns\_Institutions table**:



This table will contain all the information of the institutions that the museum associate with, including itself, the team museums and other external museums. The information of the museum will be its name and the contact information.

* ns\_in\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the institutions. And because the attribute is a primary key, it must be NOT NULL.

* ns\_in\_email

The attribute use the domain ns\_email for storing the email of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_phoneNumber

The attribute use the domain ns\_phoneNumber for storing the phone number of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_streetAddress

The attribute use the domain ns\_streetAddress for storing the street address of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_city

The attribute use the domain ns\_city for storing the city address of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_countrySubdivision

The attribute use the domain ns\_ countrySubdivision for storing the state/province of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_country

The attribute use the domain ns\_ country for storing the country of the institutions as theirs contact information. The attribute can be NULL.

* ns\_in\_postalCode

The attribute use the domain ns\_postalCode for storing the postal code of the institutions as theirs contact information. The attribute can be NULL.

* PRIMARY KEY (ns\_in\_InName )

This primary key will only need the name of the institutions, which will be the heart of the database when it exists in all the next tables.

-- (2) **ns\_Works table:**



This table contain all the information of the works that each member’s museum related to. They don’t need to be owned by the museum to go into the system.

* ns\_wk\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wk\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wk\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the team museums that are related to the works (not necessary ownership related). And because the attribute is a primary key, it must be NOT NULL.

* ns\_wk\_wkName

The attribute use the domain ns\_ wkName for storing the name of the works. The attribute can be NULL.

* ns\_wk\_creator

The attribute use the domain ns\_ creator for storing the name of the creator of the works. The attribute can be NULL.

* ns\_wk\_coDate

The attribute use the domain ns\_ coDate for storing the date the works were completed. The attribute can be NULL.

* ns\_wk\_acDate

The attribute use the domain ns\_ acDate for storing the date the works were acquire by the museum. The attribute can be NULL.

* ns\_wk\_wkDescription

The attribute use the domain ns\_ wkDescription for storing the description of the works. The attribute can be NULL.

* ns\_wk\_theme

The attribute use the domain ns\_ theme. The attribute was created as a way to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.

* ns\_wk\_subject

The attribute use the domain ns\_ subject. The attribute was created to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.

* ns\_wk\_culture

The attribute use the domain ns\_ culture. The attribute was created to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.

* ns\_wk\_colour

The attribute use the domain ns\_ colour. The attribute was created to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.

* ns\_wk\_carType
* The attribute use the domain ns\_ carType. The attribute was created to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.
* ns\_wk\_transmission

The attribute use the domain ns\_ tranmission. The attribute was created to combine team members type and subtypes. Therefore, the team members can add their own types/subtypes meaning in the team database. The attribute can be NULL.

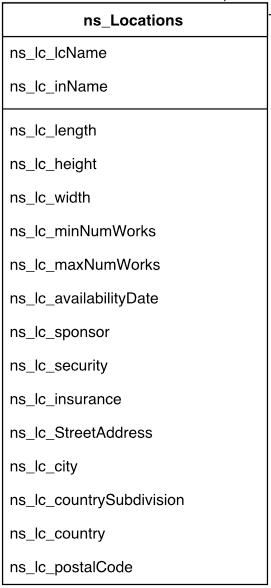
* PRIMARY KEY (ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The primary key including the identification of the works and the name of the museum that is related to them. This primary key will also play a big part in the database when it also requires this type of key from other tables that relates to works information.

* FOREIGN KEY ( ns\_wk\_inName)

The works table have connection with the institutions table to pre-check the institutions name.

-- (3) **ns\_Locations table:**



This table contain all the information of the locations and temporary locations that the team museums have.

* ns\_lc\_lcName [PK]

The attribute use the domain ns\_lcName for storing the name of the location in each museum. And because the attribute is a primary key, it must be NOT NULL.

* ns\_lc\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the team museums that own the locations. And because the attribute is a primary key, it must be NOT NULL.

* ns\_lc\_length

The attribute use the domain ns\_ lenght for storing the length of the locations. The attribute can be NULL.

* ns\_lc\_height

The attribute use the domain ns\_ height for storing the height of the locations. The attribute can be NULL.

* ns\_lc\_width

The attribute use the domain ns\_ width for storing the width of the locations. The attribute can be NULL.

* ns\_lc\_minNumWorks

The attribute use the domain ns\_ minNumWorks for storing the minimum capacity of the locations. The attribute can be NULL.

* ns\_lc\_maxNumWorks

The attribute use the domain ns\_ maxNumWorks for storing the maximum capacity of the locations. The attribute can be NULL.

* ns\_lc\_availabilityDate

The attribute use the domain ns\_availabilityDate for storing the date that the locations are available. The attribute can be NULL.

* ns\_lc\_sponsor

The attribute use the domain ns\_sponsor for storing the sponsor of the travelling exhibition, which will be held in a temporary location. The attribute can be NULL.

* ns\_lc\_security

The attribute use the domain ns\_security for storing the head of the security of the travelling exhibition, which will be held in a temporary location. The attribute can be NULL.

* ns\_lc\_insurance
* The attribute use the domain ns\_insurance for storing the insurance value of all the works in each location in the travelling exhibition, which will be held in a temporary location. The attribute can be NULL.
* ns\_lc\_StreetAddress

The attribute use the domain ns\_streetAddress for storing the address of the temporary location. The attribute can be NULL.

* ns\_lc\_city

The attribute use the domain ns\_city for storing the city of the temporary location. The attribute can be NULL.

* ns\_lc\_countrySubdivision

The attribute use the domain ns\_countrySubdivision for storing the state/province of the temporary location. The attribute can be NULL.

* ns\_lc\_country

The attribute use the domain ns\_country for storing the country of the temporary location. The attribute can be NULL.

* ns\_lc\_postalCode

The attribute use the domain ns\_ postalCode for storing the postal code of the temporary location. The attribute can be NULL.

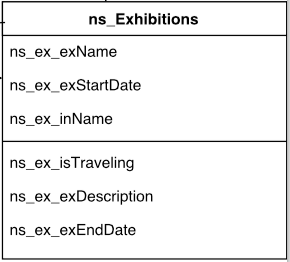
* PRIMARY KEY (ns\_lc\_lcName, ns\_lc\_inName)

The primary key including the location name and the name of the museum owning it. As the primary key of one of the main tables, this primary key will also set up the primary in the other tables that relate to the locations information.

* FOREIGN KEY (ns\_lc\_inName) REFERENCES ns\_institutions( ns\_in\_inName)

The locations table have connection with the institutions table to pre-check the institutions name.

-- (4) **ns\_Exhibitions TABLE:**



This table contain all the information of the exhibitions and travelling exhibitions that the team museums have

* ns\_ex\_exName [PK]

The attribute use the domain ns\_exName for storing the name of the exhibitions hold by each museum. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ex\_exStartDate [PK]

The attribute use the domain ns\_exStartDate for storing the date that the exhibitions start. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ex\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums held the exhibitions. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ex\_isTraveling

The attribute use the domain ns\_isTraveling for storing the information whether or not the exhibitions is a travelling one. The attribute must be NULL for the purpose of later trigger.

* ns\_ex\_exDescription

The attribute use the domain ns\_ exDescription for storing the description of the exhibitions. The attribute can be NULL.

* ns\_ex\_exEndDate

The attribute use the domain ns\_exEndtDate for storing the date that the exhibitions end. The attribute can be NULL and can be updated later.

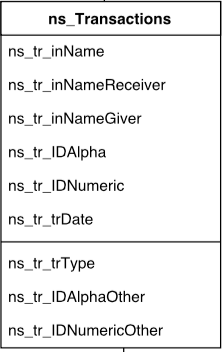
* PRIMARY KEY (ns\_ex\_exName, ns\_ex\_exStartDate, ns\_ex\_inName )

The primary key has the exhibition name and institution as the one in the location, however, it also includes the start date because there may be other exhibitions with the same name at different timeline. This primary key will also define other primary key and foreign key of the other tables that relate to exhibitions information.

* FOREIGN KEY ( ns\_ex\_inName ) REFERENCES ns\_institutions(ns\_in\_inName )

The exhibitions table have connection with the institutions table to pre-check the institutions name.

-- (5) **ns\_Transactions TABLE:**



This table contain all the transaction that the member museum has made and the time they perform it. The change in this table can also affect the ownership table.

* ns\_tr\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that is related to this transaction. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_inNameReceiver [PK]

The attribute use the domain ns\_inName for storing the name of the institutions that are the borrower or the buyer in this transaction. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_inNameGiver [PK]

The attribute use the domain ns\_inName for storing the name of the institutions that are the lender or the seller in this transaction. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item in the transaction. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item in the transaction. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_trDate [PK]

The attribute use the domain ns\_trDate for storing the date that the transaction is made. And because the attribute is a primary key, it must be NOT NULL.

* ns\_tr\_trType

The attribute use the domain ns\_ trType for storing the type of the transaction, such as: buy, sell, borrow in, borrow out, lend in, lend out. The attribute can be NULL, though it should not, for it is one part of the table identification.

* ns\_tr\_IDAlphaOther

The attribute use the domain ns\_IDNumeric for storing the part of the old identification key of the item in the transaction. This attribute will only be used when there is a transaction between the museum of the team members, and we need to store the item old identification for later recognition. The attribute can be NULL.

* ns\_tr\_ns\_IDNumericOther

The attribute use the domain ns\_IDAlpha for storing the part of the old identification key of the item in the transaction. This attribute will only be used when there is a transaction between the museum of the team members, and we need to store the item old identification for later recognition. The attribute can be NULL.

* PRIMARY KEY (ns\_tr\_inName, ns\_tr\_inNameReceiver, ns\_tr\_inNameGiver, ns\_tr\_IDAlpha, ns\_tr\_IDNumeric, ns\_tr\_trDate)

The primary key will be defined mostly by the works table and institution table for its connection with those 2 tables. We also use the transaction date as part of the transaction to avoid duplicate data.

* FOREIGN KEY (ns\_tr\_inName, ns\_tr\_IDAlpha , ns\_tr\_IDNumeric ) REFERENCES ns\_works(ns\_wk\_inName, ns\_wk\_IDAlpha, ns\_wk\_IDNumeric )

The table have connection with the works table to pre-check the items information.

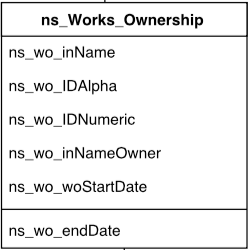
* FOREIGN KEY (ns\_tr\_inNameReceiver) REFERENCES ns\_institutions(ns\_in\_inName)

The table have connection with the institutions table to pre-check the institutions name for the receiver.

* FOREIGN KEY (ns\_tr\_inNameGiver) REFERENCES ns\_institutions(ns\_in\_inName)

The table have connection with the institutions table to pre-check the institutions name for the giver.

-- (6) **ns\_Works-Ownership TABLE:**



This table show the old and current ownership of each works in the system, which is done by the effect from the transaction table.

* ns\_wo\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that is related to the works (not necessary the owner). And because the attribute is a primary key, it must be NOT NULL.

* ns\_wo\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wo\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wo\_inNameOwner [PK]

The attribute use the domain ns\_inName for storing the name of the owners of the works. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wo\_woStartDate [PK]

The attribute use the domain ns\_woStartDate for storing the date that the ownership start. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wo\_woEndDate

The attribute use the domain ns\_woEndDate for storing the date that the ownership end. The attribute can be NULL.

* PRIMARY KEY (ns\_wo\_inName, ns\_wo\_IDAlpha, ns\_wo\_IDNumeric, ns\_wo\_woStartDate,ns\_wo\_inNameOwner )

The primary key will be defined mostly by the works table and institution table for its connection with those 2 tables. We also use the transaction date as part of the PK to avoid duplicate data.

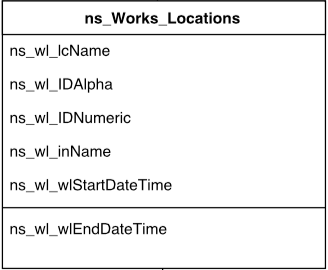
* FOREIGN KEY (ns\_wo\_IDAlpha, ns\_wo\_IDNumeric, ns\_wo\_inName) REFERENCES ns\_Works(ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The table have connection with the works table to pre-check the items information.

* FOREIGN KEY (ns\_wo\_inNameOwner ) REFERENCES ns\_Institutions(ns\_in\_inName)

The table have connection with the institutions table to pre-check the institutions name for the receiver.

-- (7) **ns\_Works-Locations TABLE:**



This table shows the history of works transferring between location inside a museum or between the member museums or even when it is on loan.

* ns\_wl\_lcName [PK]

The attribute use the domain ns\_lcName for storing the name of the location in each museum that are or used to store the works. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wl\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wl\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wl\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that is related to the works and contain the locations. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wl\_wlStartDateTime [PK]

The attribute use the domain ns\_wlStartDateTime for storing the date and time that the works go into the location. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wl\_wlEndDateTime

The attribute use the domain ns\_wlEndDateTime for storing the date and time that the works go into the location. The attribute can be NULL.

* PRIMARY KEY (ns\_wl\_lcName,ns\_wl\_IDAlpha, ns\_wl\_IDNumeric, ns\_wl\_inName, ns\_wl\_wlStartDateTime)

The primary key will be defined mostly by the works table and locations table for its connection with those 2 tables. We also use the start date time as part of the PK to avoid duplicate data.

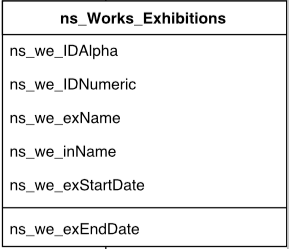
* FOREIGN KEY ( ns\_wl\_inName ,ns\_wl\_IDAlpha, ns\_wl\_IDNumeric) REFERENCES ns\_works(ns\_wk\_inName, ns\_wk\_IDAlpha, ns\_wk\_IDNumeric )

The table have connection with the works table to pre-check the items information.

* FOREIGN KEY (ns\_wl\_lcName, ns\_wl\_inName) REFERENCES ns\_locations(ns\_lc\_lcName, ns\_lc\_inName )

The table have connection with the locations table to pre-check the institutions and locations name.

-- (8) **ns\_Works-Exhibitions:**



This table shows the history of works that have been used in the exhibition, based on the period the exhibition is held and end.

* ns\_we\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_we\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_we\_exName [PK]

The attribute use the domain ns\_exName for storing the name of the exhibitions that used to or will use the items. And because the attribute is a primary key, it must be NOT NULL.

* ns\_we\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that is related to the works and will held the exhibitions. And because the attribute is a primary key, it must be NOT NULL.

* ns\_we\_exStartDate [PK]

The attribute use the domain ns\_exStartDate for storing the date that the exhibitions start. And because the attribute is a primary key, it must be NOT NULL.

* ns\_we\_exEndDate

The attribute use the domain ns\_exEndDate for storing the date that the exhibitions end. The attribute can be NULL and can be updated later.

* PRIMARY KEY (ns\_we\_IDAlpha, ns\_we\_IDNumeric, ns\_we\_exName, ns\_we\_inName, ns\_we\_exStartDate)

The primary key will be defined mostly by the works table and exhibitions table for its connection with those 2 tables. We also use the start date time as part of the PK to avoid duplicate data.

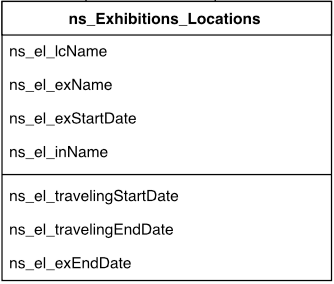
* FOREIGN KEY (ns\_we\_IDAlpha, ns\_we\_IDNumeric, ns\_we\_inName) REFERENCES ns\_works(ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The table have connection with the works table to pre-check the items information.

* FOREIGN KEY (ns\_we\_exName, ns\_we\_inName , ns\_we\_exStartDate ) REFERENCES ns\_exhibitions(ns\_ex\_exName, ns\_ex\_inName, ns\_ex\_exStartDate)

The table have connection with the exhibitions table to pre-check the exhibitions information.

-- (9) **ns\_Exhibitions-Locations TABLE:**



This table shows the history of where all the exhibitions have been held. The travelling exhibitions can be one of the special case when we must use multiple locations to represent one exhibition.

* ns\_el\_lcName [PK]

The attribute use the domain ns\_lcName for storing the name of the location that is used for the exhibitions. And because the attribute is a primary key, it must be NOT NULL.

* ns\_el\_exName [PK]

The attribute use the domain ns\_exName for storing the name of the exhibitions that are held in the locations. And because the attribute is a primary key, it must be NOT NULL.

* ns\_el\_exStartDate [PK]

The attribute use the domain ns\_exStartDate for storing the date that the exhibitions start. And because the attribute is a primary key, it must be NOT NULL.

* ns\_el\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that own the location and will held the exhibitions. And because the attribute is a primary key, it must be NOT NULL.

* ns\_el\_travelingStartDate

The attribute use the domain ns\_travelingStartDate for storing the date that the travelling exhibition end in each temporary location. The attribute can be NULL and can be updated later.

* ns\_el\_travelingEndDate

The attribute use the domain ns\_travelingEndDate for storing the date that the travelling exhibition end in each temporary location. The attribute can be NULL and can be updated later.

* ns\_el\_exEndDate

The attribute use the domain ns\_exEndDate for storing the date that the exhibitions end. The attribute can be NULL and can be updated later.

* PRIMARY KEY (ns\_el\_lcName, ns\_el\_exName, ns\_el\_exStartDate, ns\_el\_inName),

The primary key will be defined mostly by the locations table and exhibitions table for its connection with those 2 tables. We also use the start date time as part of the PK to avoid duplicate data.

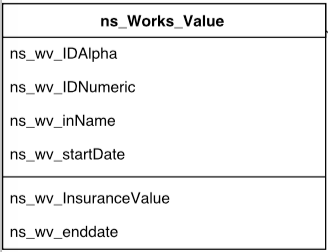
* FOREIGN KEY (ns\_el\_lcName, ns\_el\_inName) REFERENCES ns\_locations(ns\_lc\_lcName, ns\_lc\_inName)

The table have connection with the locations table to pre-check the institutions and locations name.

* FOREIGN KEY (ns\_el\_exName, ns\_el\_exStartDate, ns\_el\_inName) REFERENCES ns\_exhibitions(ns\_ex\_exName, ns\_ex\_exStartDate, ns\_ex\_inName)

The table have connection with the exhibitions table to pre-check the exhibitions information.

-- (10) **ns\_Works\_Value TABLE:**



This table shows the history of works value have been changed in the time it is belong to a team museum. When the item is no longer belong to the museum, we only need to update its end date and never care about its new value anymore.

* ns\_wv\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wv\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wv\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums that currently have the works when its value changed. So, when they don’t own the work anymore, they don’t need to update its new value. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wv\_InsuranceValue [PK]

The attribute use the domain ns\_insurance for storing the old and new value of the work when it is owned by a team museum, however, there won’t need any update when the its locations is changed, even between the team museum. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wv\_startDate

The attribute use the domain ns\_wvStartDate for storing the date that the new value is set. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wv\_enddate

The attribute use the domain ns\_exEndDate for storing the date that the old value is no more. The attribute can be NULL and can be updated later.

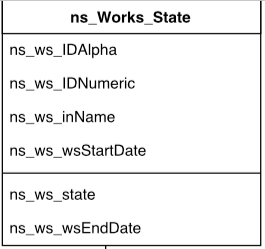
* PRIMARY KEY (ns\_wv\_IDAlpha, ns\_wv\_IDNumeric, ns\_wv\_inName, ns\_wv\_InsuranceValue)

The primary key will be defined mostly by the works table. We also use the start date time as part of the PK to avoid duplicate data.

* FOREIGN KEY (ns\_wv\_IDAlpha, ns\_wv\_IDNumeric, ns\_wv\_inName) REFERENCES ns\_works(ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The table have connection with the works table to pre-check the items information.

-- (11) **ns\_Works\_State TABLE:**



This table shows the history of works state have been changed in the time it is belong to a team museum. When the item is no longer belong to the museum, we only need to update its end date and never care about its new state anymore.

* ns\_ws\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ws\_IDNumeric[PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ws\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that own the works when it current state changed. And because the attribute is a primary key, it must be NOT NULL.

ns\_inName NOT NULL,

* ns\_ws\_wsStartDate [PK]
* The attribute use the domain ns\_wsStartDate for storing the date that the new state is set. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ws\_state

The attribute use the domain ns\_state for storing the old and new state of the work when it is owned by a team museum, however, there won’t need any update when the its locations is changed, even between the team museum. The attribute can be NULL and can be updated later

* ns\_ws\_wsEndDate

The attribute use the domain ns\_exEndDate for storing the date that the old state is over. The attribute can be NULL and can be updated later

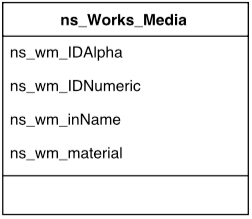
* PRIMARY KEY (ns\_ws\_IDAlpha, ns\_ws\_IDNumeric, ns\_ws\_inName, ns\_ws\_wsStartDate)

The primary key will be defined mostly by the works table. We also use the start date time as part of the PK to avoid duplicate data.

* FOREIGN KEY (ns\_ws\_IDAlpha, ns\_ws\_IDNumeric, ns\_ws\_inName) REFERENCES ns\_works(ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The table have connection with the works table to pre-check the items information.

-- (12) **ns\_Works\_Media TABLE:**



This table contain the material that made up the works.

* ns\_wm\_IDAlpha [PK]

The attribute use the domain ns\_IDAlpha for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wm\_IDNumeric [PK]

The attribute use the domain ns\_IDNumeric for storing the part of the identification key of the item. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wm\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums of the team that is related to the works. And because the attribute is a primary key, it must be NOT NULL.

* ns\_wm\_material [PK]
* The attribute use the domain ns\_material for storing the material that made the works. And because the attribute is a primary key, it must be NOT NULL.

ns\_material NOT NULL,

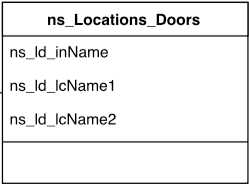
* PRIMARY KEY (ns\_wm\_IDAlpha, ns\_wm\_IDNumeric, ns\_wm\_inName, ns\_wm\_material)

The primary key will be defined mostly by the works table. We also use the material as part of the PK to avoid duplicate data.

* FOREIGN KEY (ns\_wm\_IDAlpha, ns\_wm\_IDNumeric , ns\_wm\_inName) REFERENCES ns\_works(ns\_wk\_IDAlpha, ns\_wk\_IDNumeric, ns\_wk\_inName)

The table have connection with the works table to pre-check the items information.

-- (13) **ns\_Locations\_Doors TABLE:**



This table contain the information of the door that connect the rooms in each team member locations.

* ns\_ld\_inName [PK]

The attribute use the domain ns\_inName for storing the name of the museums that own the locations. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ld\_lcName1 [PK]

The attribute use the domain ns\_lcName for storing the name of the location that have the have the door connect to the other rooms. And because the attribute is a primary key, it must be NOT NULL.

* ns\_ld\_lcName2 [PK]

The attribute use the domain ns\_lcName for storing the name of the location that have the have the door connect to the other rooms. And because the attribute is a primary key, it must be NOT NULL.

* PRIMARY KEY (ns\_ld\_inName, ns\_ld\_lcName1, ns\_ld\_lcName2)

The primary key will be defined mostly by the locations table. The 2 rooms connect by the door will also be a part of the PK

* FOREIGN KEY (ns\_ld\_lcName1, ns\_ld\_inName) REFERENCES ns\_locations(ns\_lc\_lcName, ns\_lc\_inName)

The table have connection with the locations table to pre-check the institutions and locations name.

* FOREIGN KEY (ns\_ld\_lcName2, ns\_ld\_inName) REFERENCES ns\_locations(ns\_lc\_lcName, ns\_lc\_inName)

The table have connection with the locations table to pre-check the institutions and locations name.

**Part D**.

**1. Kevin Noonan’s report**

~~Currently my database is comprised of ten tables. The new structure that I will be converting my database to has twelve tables. The new tables that are being added to the structure are the Media table, the WorksState table, the WorksInsuranceValue table and a new combined table called WorksKeepers. The WorksKeepers table takes attributes from the onloanto table and owners table from my previous database to make one organized table. Most of the tables have one or two attributes that need to be changed to accommodate the new structure I will now describe the changes to the tables and the testing requirements for each.~~

In my database there is a decent amount of changes to be made to accommodate for the new schema. Below I will list the tables that currently exist in my database alongside the tables that will exist in the new database that will be used in the team database.

**Institutions:**

|  |
| --- |
| **New table:** |
| **Institutions** |
| in\_inName |
| in\_email  in\_phoneNumber  in\_streetAddress  in\_city  in\_countrySubdivision  in\_country  in\_postalCode |

The institutions table is a new table that we have added to the schema. In order for me to add this table to the new schema I will have to create each value manually. The primary key will be the name of the institution that could possibly own a work and all of it’s information. Because this information didn’t exist before it will have to be inputted manually. Because of the manual insertion the testing for this table will be to create a query of the whole table and check the results.

**Works:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current table:**   |  | | --- | | **Works** | | wksletterkey  wksnumberkey | | wkscarname  wksdescription  wksbodytype  wkstranstype  wksmanufacturer  wksengine  wksdateofcompletion  wksinsurancevalue  wksownership  wksdonor  wkscountry  wksdateofacquisition | | **New table:**   |  | | --- | | **Works** | | wk\_idalpha  wk\_idnumeric  wk\_inname | | wk\_wkname  wk\_creator  wk\_completionDate  wk\_acquisitionDate  wk\_wkdescription  wk\_theme  wk\_subject  wk\_culture  wk\_colour  wk\_carType  wk\_transmission | |

My works table will go through a couple of changes in order to accommodate the new works table. I will insert my museums name from the institutes table into wk\_inname which is a new attribute to the works table. Additionally wk\_theme, wk\_subject, and wk\_colour I will set to null as these attributes do not reflect anything in my table. Wksinsurancevalue will not to be transferred as this data will go to the new works\_value table. Also wksownership and wksdonor will not be added as they will go to works\_ownership table. The remaining attributes will be added in a insert into statement. The testing for this table will be to create a query that will display all of the data that has been added. Additional testing will be using a query between the works table and the works\_value table using the primary keys to check the insurance value. Another test will be to use a query between the works table and the works\_ownership table to check the ownership of borrowed works and owned works.

~~One attribute that will be changed in the works table is the wksdonor. This attribute is not needed as it is already represented in the wksowners table. I will be removing this attribute from the table. Also I will be removing wksengine attribute which represents the media of the table and the wksinsurancevalue. I will be taking the data from the wksengine column and adding it to the new Media table and the wksinsurancevalue will go to the WorksInsuranceValue table. A new attribute that will be added to the works table is the wk-question attribute which adds a question column with the attribute wkscountry being the renamed wk-answer in the new structure. I will test these new changes but creating a query to check for just the columns wk-question and wk-answer. I can also create a query to test for wksengine to see if it still exists. That query should return with no value if it was removed correctly.~~

**Locations:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **Locations** | | locroomname | | locroomwidth  locroomlength  locroommin  locroommax  locavailable | | **New:**   |  | | --- | | **Locations** | | lc\_lcname  lc\_lcmuseumName FK-in | | lc\_length  lc\_width  lc\_height  lc\_minNumWorks  lc\_maxNumWorks  lc\_availableDate  lc\_sponsor  lc\_security  lc\_insurance  lc\_streetAdress  lc\_city  lc\_countrySubdivision  lc\_country  lc\_postalCode | |

The locations table can be easily adjusted to accommodate the new schema. Since we combined the locations table with the temporary locations table we have a new set of attributes to be stored. These will be manually written in as values to be inserted into the table. Additionally we have also added a height value to our room size. This will be manually added as a static value as all of the rooms will be the same height. Along with those changes we also add the institute name to the locations this will be taken from the institution table. The testing for the new values since they were manually added the first test should involve using a query for the whole table to make sure the values are correct. Another test query will be to use the locations table and the works location table to check the temporary locations and which works are associated with them.

~~The locations table has multiple changes added to the table. The new attribute will be added called lc-museumName that represents the museums name to that location. Two attributes that currently exist in my table are locroomwidth and locroomlength. These attributes will be deleted and a new domain dimension tab under the attribute lc-dimension will be created, that has the attributes length, width and height. Additionally two new attributes will be added to the locations table called lc-telStartDate and lc-telEndDate. Testing for the new lc-dimension will be to query the attribute to see if the proper values are stored in the length, width and height. The lc-telStartDate and lc-telEndDate attributes will be tested by calling the query with those two attributes and the two date values in the table TempExhibitionsLocations to see if they are the same value.~~

**WorksLocations:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **worksLocations** | | locletterkey FK-wk  locnumberkey FK-wk  locroomname  locstartdate | |  | | **New:**   |  | | --- | | **WorksLocations** | | wl\_lcName FK-lc  wl\_wkIDAlpha FK-wk  wl\_wkIDNumeric FK-wk  wl\_inName FK-lc, wk  wk\_wlStartDateTime | | wk\_wlEndDateTime | |

~~The workslocations table will have two additional attributes added to the table. One attribute will be wl-lcmuseumname which is the museums name that is associated with that location. The second attribute will be and end date time with a timestamp. As a start date time already exists adding an end date time is essential. These two attributes will be tested by using a query to call three columns wl-lcmuseumName, wk-workLocationEndDateTime and wl-lcName and making sure that the first two are associated with the wl-lcName.~~

My works locations table will need to have two changes in order to accommodate the new schema. First I will create a new timestamp attribute for wk-wlEndDateTime and use a insert into statement for the table. Second I need to use the institute names from both the works table and locations table in order to represent which work is in which location in the works locations table. In order to test this table I will create two separate queries. The first query will be between the works locations table and the works table with the institute names equal to each other. This result should show only the works that are owned by my museum. The second query will be between the works locations table and the locations table with the institute names equal to each other except my museum name. This should show all of the works that are owned by a private owner.

**~~TempExhibitionsLocations:~~**

~~The TempExhibitionsLocations table will have the following additional attributes to the table. The templocadd attribute in my current table holds a city as it’s address. This will be changed to tel-lcname attribute in the new table and the newly named tel-tempExhibitionLocationAddress will contain an exact address. Also a new attribute that will be added to the table is the tel-lcMuseumName which will contain the museum that is associated with the temporary exhibition. The testing for this table will be to create a query that checks the tel-lcname, tel-lcMuseumName and tel-tempExhibitionLocationAddress and cross reference that with the attributes in the Locations table lc-name and lc-museumName.~~

**Exhibitions:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **exhibitions** | | exbname | | exdesc  exstartdate  exbenddate | | **New:**   |  | | --- | | **exhibitions** | | ex\_exName  ex\_exStartDate  ex\_inName FK-in | | ex\_isTraveling  ex\_exDescription  ex\_exEndDate | |

The exhibitions table will be easily accommodated to the new schema. The ex\_inName attribute will added from the institutions table and the attribute ex\_isTraveling will be manually added with a static variable of false. It should be noted that this may be changed in the future with the addition of a traveling exhibition in which case there will be an update to the column to change it to true based off of the ex\_exName. Testing for this will be to first test the manually inputted variables by creating a query for the entire table. This will also show the institute names. Additionally I can create another query between the exhibitions table and the works exhibitions table to check for which work is associated with which exhibition and if it is traveling or not.

~~The exhibitions table will have two new additional attributes added. The attribute ex-museumName that associates the museum name with the exhibition and the boolean attribute ex-isTraveling that holds the value of if the exhibit is in the museum or not. These attributes will be tested by using a query to compare ex-isTraveling with ex-exhibitionName and ex-exhibitionStartDate. Additionally these values will be queried with the locations attribute lc-telStartDate. If the ex-isTraveling value is true then the ex-exhibitionStartDate will be five days after the lc-telStartDate. Otherwise the value should be shown as false and the ex-exhbitionName will be located in the museum.~~

**ExhibitionsLocations:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **exhibitionLocation** | | exbloc  exblname FK - ex | |  | | **New:**   |  | | --- | | **ExhibitionsLocations** | | el\_lcName FK-lc  el\_exName FK-ex  el\_exStartDate FK-ex  el\_inName FK-lc, ex | | el\_travelingStartDate  el\_travelingEndDate  el\_exEndDate | |

The exhibitionslocations table has five additional attributes that need to be added to work with the new database structure. Four date attributes need to be added. The attributes for the exhibition start date and end date will be from the exhibition table start date and end date. At first the traveling start date and end date will be set to null on insertion. Once I have added traveling exhibitions to the database I can create another insert where the exhibition is traveling and set the date values to the availability date of the traveling exhibition. The testing for this table will be to create a query between the exhibition locations table and the locations table so I can cross reference the data within the table with the locations information. I will check this query to make sure the data is correct.

~~Two of these date attributes are for the exhibitions start date and end date which are named el-exStartDate and el-exEndDate~~. ~~The other two date attributes are for the TempExhibitionsLocations table exhibitions which are named el-lctelStartDate and el-lctelEndDate. The other attribute that needs to be added is the museum name as el-lcexMuseumName that is associated with the exhibition location. The testing for these new attributes will be to create a query the el-exStartDate and el-exEndDate with the el-exName and el-lcexMuseumName to check that the dates match with the exhibitions and the associated museum. I will create a separate query to check the el-lctelStartDate and el-lctelEndDate with the el-exName to check that those dates match with their respective exhibitions.~~

**ExhibitionsWorks:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **exhibitionsWorks** | | exbwname FK-ex  exbletterkey FK-wk  exbnumberkey FK-wk | |  | | **New:**   |  | | --- | | **Works Exhibitions** | | we\_IDAlpha FK-wk  we\_IDNumeric FK-wk  we\_exName FK-ex  we\_inName FK-wk, ex  we\_exStartDate FK-ex | | we\_exendDate | |

The ~~exhibitionworks~~ Works Exhibitions table will be have three additional attributes added to the table. The new attributes are two date attributes we\_exStartDate and we\_exEndDate that track the date of the exhibition. The third attribute is the museum name as we\_inName that is associated with each exhibition that exists. The date attributes will be added from the exhibitions table startdate value and enddate value. The museum name will be added from the exhibitions table but will represent a foreign key to both the works and exhibitions table.  ~~These attributes will be tested by creating a query with the ew-exName and the eq-exMuseumName and make sure that each exhibition is with it’s associated museum. I will then add the ew-exStartDate and ew-exEndDate and make sure those dates match with the associated exhibitions.~~ The table will be tested by creating a query with the exhibitions table to compare the start date and end date between exhibitions. If there are only nine exact exhibitions we know that the date was implemented correctly.

**Works Ownership:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Owners** | | ownersname  ownersworksl FK – wk  ownersworksn FK – wk  ownersacq | |  | | *New:*   |  | | --- | | **Works\_Ownership** | | wo\_inName FK-wk  wo\_IDAlpha FK-wk wo\_IDNumeric FK-wk  wo\_inNameOwner FK-in  wo\_woStartDate | | wo\_EndDate | |

The owners table will not exist in the new database structure but the values from my database will be transferred to the new ~~WorksKeepers~~ Works-Ownership table. These attributes are ownersname, ownersworkssl, ownersworkssn, ownersacq. ~~These attributes will be renamed as wkk-workKeeperName, wkk-wkIDAlpha, wkk-wkIDNumeric and wkk-workKeeperStateDate respectively. The testing for these values will be answered in the WorksKeepers section.~~ The ownersacq date will become the start date for works ownership. Additionally the institution name will be added to this table from the institutions table. The end date will be at first set to null and will be updated if needed be. Testing for this table will be to create a query between the works ownership table and the institutions table so we can compare the date between the institutions name and their info that is stored in the institutions table.

**Doors:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Current:**   |  | | --- | | **Doors** | | drsnumber  drsopento FK -lc  drscloseto FK - lc | |  | | **New:**   |  | | --- | | **Locations\_Doors** | | ld\_inName FK-lc  ld\_lcName1 FK-lc  ld\_lcname2 FK-lc | |  | |

The doors table will be easy to accommodate the new schema. The drsnumber attribute is no longer needed so we will insert the date of drsopento and drscloseto to ld\_lcName1 and ld\_lcname2 respectively. Additionally the institution name will become an attribute and that data will be inserted from the locations table. To test this query we simply need to create a query of the table and check with our data that each location was inputted properly and represents its individual door. Additionally we can create a query with the locations table so that when the database is merged we can check to see which location creates a door in which museum.

~~The doors table will have an attribute removed and then replaced with a new attribute. The attribute that is being removed is the drsnumber attribute. The new attribute will be the dr-lcMuseumName which will hold the museums name for the associated door location. This will be tested by selecting the doors table and cross checking the location attributes are matched with their associated museum.~~

**Transactions:**

|  |
| --- |
| **Transactions** |
| tr\_inname FK-wk  tr\_innamereceiver FK-in  tr\_innamegiver FK-in  tr\_idalpha FK-wk  tr\_idnumeric FK-wk  tr\_trdate |
| tr\_trtype  tr\_idalphaother  tr\_idnumericother |

The transactions table is a new table that is create to track the ownership of the works and how they were processed. In order to create this table I will create five different insert into statements. The reason for this is because we need to insert different values for each type of tr\_trtype. The possible values of this attribute are ‘Buy’, ‘Sell’, ‘Borrow in’, Borrow Out’, ‘Lend in’ and ‘Lend Out’. I know that there hasn’t been any works ‘Borrow Out’ yet so I will not create an insert for that but it can be updated if needed. In each insert statement I have the institution name from the works table. For the insert’s that involve a works coming in or buying I will use the institution name from the works table, otherwise i will use the onloanname from the onloanto table or the ownersname from the owners table. Next I will use the wksdonor from the works table for ‘Buy’ or ‘Borrowed’. Anything else will use either the onloanname variable or the institution name from the works table. I will then insert the wk\_idalpha and wk\_idnumeric. The dates will either be the wk\_acquistiondate, onloanstart or ownersacq. The type will be on of the attributes and will be typed into the select statement. Finally the last two attributes will be the wk\_idalpha and wk\_idnumeric from the works table just in case they were changed. This table will be tested by creating a query with the works table and comparing the works data with the tr\_idalpha and tr\_idnumeric within the transactions table.

**OnLoanTo:**

|  |
| --- |
| **Onloanto** |
| onloanname  onloanaddress  onloannum  onloanemail  onloanstart  onloanend  onloanletterkey FK-wk  onloannumberkey FK-wk |

The onloanto table will not exist in the new database as we use most of the attributes in the new schema in the transactions table. The onloanname will be used to insert the data of the who owns the works into the transactions. The onloanstart and onloanend will be used to insert the data of when the transaction began and if it ended in the transactions table. The rest of the attributes will be available to be called if needed for queries or other select statements. The testing for this table will be to create a query for the transactions table

~~This table will not exist in the new database structure but the values from my database will be transferred to the new WorksKeepers table. These attributes are onloanname, onloanaddress, onloannum, onloanemail, onloanstart and onloanend. These attributes will be put into the following new attributes in the WorksKeepers table: wkk-workKeeperName, wkk-workKeeperAddress, wkk-phoneNumber, wkk-email, wkkKeeperStartDate and wkkKeeperEndDate respectively. The testing for these values will be explained in the WorksKeepers section~~

.

**Media:**

|  |
| --- |
| **Works Media** |
| wm\_IdAlpha FK-wk  wm\_IdNumeric FK-wk  wm\_inName FK-wk  wm\_material |
|  |

The media table is a new table that is added to the database structure. This table will hold four attributes. The first two attributes wm\_IdAlpha and wm\_IdNumeric will be inserted from the works table. Additionally the institution name will also be inserted from the works table to represent the museum the works are associated with. The wm\_material attribute will be inserted from the works table as we are removing media from the works table to create it’s own table. This table will be tested by creating a query between the media table and the works table and making sure that each wm\_material is with it’s associated work id.

~~This table will hold three attributes. Two of the attributes are foreign keys to the works attributes wk-IDAlpha and wk-IDNumeric. These will be named md-wkIDAlpha and md-wkIDNumeric. The third attribute added will be the wksEngine attribute that was removed from the works table to be placed in this new Media table as md-material. This will be test by simply selecting the new Media table and making sure that the md-wkIDAlpha and md-wkIDNumeric matches with it’s respective md-material.~~

**~~WorksKeepers:~~**

~~The workskeepers table will be created from combining two of my old tables; onLoanto and owners. This new table will have nine new attributes. All of the attributes already exist which is except for one which is wkk-status. This attribute will hold the value of whether the value is loaned, borrowed or potentially borrowed. After that attribute is created it is simply a matter of updating the table and renaming the attributes. The attributes that are being added have been described in the owners and onloanto section. The testing will be to create a query that selects all the attributes and then cross reference them with the old tables onloan to and owners to make sure that the values are correct. Additionally a query will be made with wkk-wkIDAlpha and wkk-wkIDNumber and wkk-status to check that each work has a status of loaned, borrowed or potentially borrowed.~~

**WorksState:**

|  |
| --- |
| **WorksState** |
| ws\_IDAlpha FK-wk ws\_IDNumeric FK-wk  ws\_inname FK-wk  ws\_wsStartDate |
| ws\_state  ws\_wsEndDate |

The works state table is a secondary table that holds a new attribute called state. I will create this table by inserting the wk\_idalpha and wk\_idnumeric from the works table. I will also insert the institution name from the works table. The start date will be inserted from the works table as well from ws\_acquistiondate attribute. The ws\_state attribute will be manually updated as the works state change but for now the static variable will be ‘Good’. The ws\_wsEnddate will be set to null until there is a work that needs to be updated. I can test this table by creating a query with the works table and associating the state with the wk\_idalpha and wk\_idnumeric in the works table.

~~The worksstate table is a new table that is created with two foreign keys to the works table. These two attributes ws-wkIDAlpha and ws-wkIDNumeric will come from the wk-IDAlpha and wk-IDNumeric attributes. Additionally there are three other attributes that will be added to this table. Two the attributes are date values named ws-workStateStartDate and ws-workStateEndDate. The other attribute is wk-state which holds the state of the work such as “damaged” or “in good condition”. These will be tested by setting the wk-state of each work to “in good condition”, selecting the table and making sure that each work id is set to in good condition as none of the works have had that value changed yet. Also we will make sure that the end date value is set to null.~~

**Works Value:**

|  |
| --- |
| **Works Value** |
| wv\_IDAlpha FK-wk wv\_IDNumeric FK-wk  wv\_inName FK-wk  wv\_StartDate |
| wv\_insuranceValue  wv\_EndDate |

The works value table is a new table that is created from an attribute previously in the works table. This table stores temporal data for the insurance value of the work. I will create this table by inserting the wk\_idalpha, wk\_idnumeric, wk\_inname from the works table. Additionally I will insert the wk\_acquisitiondate as the wv\_startdate. The wv\_insurancevalue will be inserted from the wksinsurancevalue in the old works table. At first the wv\_enddate will be set to null until the insurance value of a work needs to be updated. This table will be tested by creating a query with the works table to compare the wk\_idalpha and wk\_idnumeric from the works table with the wv\_insurancevalue to make sure that the associated works match their value.

~~The worksInsuranceValue is a new table that is created with two foreign keys to the works table. These two attributes wiv-wkIDAlpha and wiv-wkIDNumeric will come from the wk-IDAlpha and wk-IDNumeric attributes. Additionally there are three other attributes that will be added to this table. Two the attributes are date values named wiv-workInsuranceValueStartDate and ws-workInsuranceValueEndDate. The third attribute is the wiv-insuranceValue which will come from the works table. The testing for this will be similar to the WorksState table except no values will be added except for the end date attribute to null. I will then create a query that compares the wk-IDAlpha and wk-IDNumeric with the work attributes and compare the insurance value to each to make that each id has the same insurance value.~~

**2. Daniel Morris’s report**

There is quite a bit I need to change to my database before it will be ready to merge with the design stated in part C above. Below is a list of the changes and the testing plan for each of the defined tables above compared to my assignment 3 design.

**- Institutions**

|  |
| --- |
| **Institutions** |
| in-inName |
| In-email  in-phoneNumber  in-streetAddress  In-city  in-countrySubdivision  In-country  in-postalCode |

The institutions table is a new table that we have devised will work well with our new schema. Since this table is completely new I will have to insert the data for it manually. This table will contain my museum as one row and then the rest will be institutions that my museum has borrowed from or loaned out to. Since this table is being created from scratch with manually entered data there will be nothing to test other than querying the entire table.

- Works table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Works** | | wkIdAlpha  wkIdNum | | wkName  wkType  wkSubtype  wkArtist  wkCompDate  wkAcqDate  wkValue  wkDesc  wkBorrowStatus  wkIsAboriginal | | *New group table:*   |  | | --- | | **Works** | | wk-IDAlpha  wk-IDNumeric  wk-InName | | wk-worksName  ~~wk-type~~  ~~Wk-subtype~~  Wk-theme  Wk-subject  Wk-culture  Wk-colour  wk-carType  wk-transmission  wk-Creator  wk-completionDate  wk-acquisitionDate  wk-workDexcription  wk-ownership  ~~wk-question~~  ~~wk-answer~~ | |

There are a few changes to be made to my Works table. The value of a Work will no longer be stored in the Works table. I will need to insert the name of my museum into every row of the new works table. I will create the new table WorksInsuranceValue and create a new entry in that table for each of my works. Since the insurance value of all my works has remained constant I will just insert every work into the WorksInsuranceValue table with their respective insurance values and set the start date to the work’s date of acquisition and the end date to null. My wkIsAboriginal attribute of my works table will now be inserted into the wk-culture attribute. Wherever the wkIsAboriginal was set to true the wk-culture will be set to ‘Australian Aboriginal’.~~splitting into two attributes in the new table. wk-question will always be “Is the work aboriginal?” for every work in my database and wk-answer will be the same as wkIsAboriginal: either yes or no.~~

To test the insurance value attribute changes in the table I will create a query to select the wkIdAlpha, wkIdNum, wkAcqDate, and wkValue from the old Works table and wiv-wkIdAlpha, wiv-wkIdNumeric, wiv-workInsuranceValueStartDate, wiv-insuranceValue from the new WorkInsuranceValue table. I will check to make sure these values match up. To test the wk-question and wk-answer attributes I will select the wkIdAlpha, wkIdNum, and wkIsAboriginal from the old Works table and wk-IDAlpha, wk-IDNumeric, and wk-answer from the new Works table. I will check to make sure these values match up. I will also select every wk-question from the new Works table to check if it is set to “Is the work Aboriginal?”.

- Locations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Rooms** | | rmName | | rmWidth  rmLength  rmSuggMax  rmSuggMin  rmAvailabilityDate | | *New:*   |  | | --- | | **Locations** | | lc-name  lc-museumName | | ~~lc-telStartDate~~  ~~lc-telEndDate~~  ~~lc-dimension~~  lc-minNumWorks  lc-maxNumWorks  lc-availableDate  lc-length  lc-width  lc-height  lc-sponsor  lc-security  lc-insurance  lc-streetAdress  lc-city  lc-countrySubdivision  lc-country  lc-postalCode | |

For the locations table I will need to add a new attribute to store the museum in which the location is in. I will set it to the default the name of my museum. The new database will combine my existing rooms and travelingExhibitionsLocations tables into one Locations table. Due to this we needed to add all the traveling exhibition info to the locations table. If the exhibition is not traveling then this data will simply be set to null.

To test this new locations table I will first port all my data over from the two tables and then select the whole table and manually check for imperfections. Since there is less than 15 locations in my museum this is a reasonable approach to testing this table.

~~The lc-telStartDate and telEndDate will be set to null for each location that isn’t a traveling exhibition. For my traveling exhibition though I will need to add a location for each temporary location. Since I did not implement a startDate and endDate to my travelingExhibition locations I will need to make these dates up. In my Rooms table the rmWidth, and rmLength will be both combined into lc-dimention in the new Locations table.~~

~~To test the changes to the room dimensions I will query the old Rooms table selecting the length and width, and the new Locations table selecting the dimension to confirm the date has been mapped accordingly. To test the lc-telStartDate I will query all of the temporary locations in my traveling exhibition and confirm that their start dates and end dates add up to the total duration of the exhibition.~~

- Exhibitions table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitions** | | exName | | exDesc  exStart  exEnd | | *New:*   |  | | --- | | **exhibitions** | | ex-exhibitionName  ex-exhibitionStartDate  ex-museumName | | ex-isTraveling  ex-exhibitionDescription  ex-exhibitonEndDate | |

For the exhibitions table I will have to add two new attributes. The first is the attribute ex-MuseumName which will be the same for all of my exhibitions. The other is a Boolean value ex-isTraveling, this will be manually set to false for all of my static exhibitions and true for my traveling exhibition.

To test these changes I will select ex-exhibitionName, ex-museumName, ex-isTraveling and check to see that the correct values for these attributes have been set.

Temporal data Table:

- ~~TempExhibitionsLocations table~~

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *~~Current:~~*   |  | | --- | | **~~travelingExhibitLocations~~** | | ~~treName FK-ex~~ | | ~~treSponsor~~  ~~treAddress~~  ~~treSecurity~~  ~~treInsurance~~ | | *~~New:~~*   |  | | --- | | **~~TempExhibitionsLocations~~** | | ~~tellcname FK-lc~~  ~~tel-lcMuseumName FK-lc~~  ~~tel-tempexhibitionLocationStartDate~~ | | ~~tel-sponsor~~  ~~tel-security~~  ~~tel-insurance~~  ~~tel-tempExhibitionLocationEndDate~~  ~~tel-tempExhibitionLocationAddress~~ | |

~~For the TempExhibitionsLocations table I will also need to add another attribute for the museum name. I will also need to add an attribute for the start date and end date. I will acquire these dates from the previously defined location that corresponds to the temp location.~~

~~To test this table, I will first ensure that every entry in the table has the name of my museum in the tel-lcMuesumName attribute. I will then query the location table’s lc-telStartDate and lc-telEndDate and make sure that there is a corresponding tempExhibitionsLocation for all of these locations.~~

- WorksLocation table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **worksLocations** | | locIdAlpha FK-wk  locIdNum FK-wk  locName  locMoveDate | |  | | *New:*   |  | | --- | | **WorksLocations** | | wl-lcName FK-lc  wl-wkIDAlpha FK-wk  wl-wkIDNumeric FK-wk  wl-lcmuseumName FK-lc  wk-workLocationStartDateTime | | wk- workLocationEndDateTime | |

For the worksLocations table I will need to add a new attribute for the museum name. It was also decided in our group that all of our temporal data be closed-closed temporal data so I will need to convert my closed-open temporal data to closed-closed. This also means adding the wk- workLocationEndDateTime attribute.

To test this table, I will first ensure that every entry in the table has the name of my museum in the wl-lcMuesumName attribute. I will then query my worksLocations table and the new worksLocations table sorting by locIdAlpha, locIdNum, locStartDate ordered by locStartDate. This will allow me to see very quickly if the dates have been converted correctly.

- ExhibitionsLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitLocation** | | exLocName FK – ex  exLocloc FK - rm | |  | | *New:*   |  | | --- | | **ExhibitionsLocations** | | el-lcName FK-lc  el-exName FK-ex  el-exStartDate FK-ex  el-lcexMuseumName FK-lc, ex | | el-lctelStartDate  el-lctelEndDate  el-exEndDate | |

For my exhibitLocation table I will need to add a few new attributes. I will need to add an attribute for the museum name, an attribute for the start date of the exhibition, the end date of the exhibition, and the start and date of the temporary location if this location is a temporary one. The el-lcexMuseumName attribute will be the same for every exhibitionLocation. The el-exStartDate and el-exEndDate attributes will be acquired from the exhibitions table. The el-lctelStartDate el-lctelEndDate will be acquired from the locations table previously defined.

To test this table, I will first ensure that every entry in the table has the name of my museum in the el-lcexMuesumName attribute. I will then query the exhibitionsLocations table with the exhibitions table matching together the exhibition names and will check that their start dates and end dates also match. I will also query the exhibitionsLocations table with the locations table matching together the location name and will check that the tempStartDates and tempEndDates also match.

- ExhibitionsWorks table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitWorks** | | ewIdAlpha FK-wk  ewIdNum FK-wk  ewName FK-ex | |  | | *New:*   |  | | --- | | **ExhibitionsWorks** | | ew-exName FK-ex  ew-wkIDAlpha FK-wk  ew-wkIDNumeric FK-wk  ew-exStartDate FK-ex  ew-exMuseumName FK-ex | | ew-endDate | |

For my exhibitionsWorks table I will need to add a few new attributes. I will need to add an attribute for the museum name, an attribute for the start date of the exhibition,and the end date of the exhibition. The ew-lcexMuseumName attribute will be the same for every exhibitionLocation. The ew-exStartDate and ew-exEndDate attributes will be acquired from the exhibitions table.

To test this table, I will first ensure that every entry in the table has the name of my museum in the ew-exMuesumName attribute. I will then query the exhibitionsWorks table with the exhibitions table matching together the exhibition names and will check that their start dates and end dates also match.

~~- WorksKeeper table~~

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *~~Current:~~*   |  | | --- | | **~~workOwner~~** | | ~~wkoIdAlpha FK – wk~~  ~~wkoIdNum FK – wk~~  ~~wkoName~~  ~~wkoDate~~ | |  | | **~~workLoan~~** | | ~~wklIdAlpha FK – wk~~  ~~wkIdNum FK – wk~~  ~~wklName~~ | | ~~wklAddress~~  ~~wklPhoneNum~~  ~~wklEmail~~  ~~wklStart\_date~~  ~~wklEnd\_date~~ | | *~~New:~~*   |  | | --- | | **~~WorksKeeper~~** | | ~~wkk-workKeeperName~~  ~~wkk-wkIDNumberic FK-wk~~  ~~wkk-wkIDAlpha FK-wk~~  ~~wkk-workKeeperStartDate~~  ~~wkk-status~~ | | ~~wkk-workKeeperAddress~~  ~~wkk-workKeeperEndDate~~  ~~wkk-email~~  ~~wkkphonenumber~~ | |

~~The worksKeeper table is the combination of the workOwner and workLoan table. Because this 2 tables have the same attributes and can work in the same way, all we need to do is to add another attribute to know whether or not the items are lent or borrowed. In my workOwner table I had wkoDate as a closed-open temporal data structure. I will need to convert this to closed-closed to work with the new WorksKeeper table.~~

~~To test this new table I will query it against the old tables I had and make sure the attributes have been moved over sussessfully. The wkk-status attributed should be borrowed, potentially borrowed, or loaded out.~~

* Works Ownership Tables

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Owners** | | wkoIdAlpha FK – wk  wkoIdNum FK – wk  wkoName  wkoDate | |  | | *New:*   |  | | --- | | **Works\_Ownership** | | wo\_inName  wo\_IDAlpha  wo\_IDNumeric  wo\_inNameOwner  wo\_woStartDate | | wo\_woEndDate | |

For the works ownership table I will need to change the temporal data structure from closed-open to closed-closed. I will alsoo have to include institution names as part of the primary key for the ownership table. This will mean I will have to insert into the institutions table all the wkoName as institutions.

To test this table I will select both Owners and Works\_Ownership, where the ID values of the works are equal, then check that wkoDate = wo\_woStartDate. I will then also query the owners table with the institutions table to make sure that the wkoName has actually been inserted as an institution.

- Doors table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | doorNum  doorRm1 FK – rm  doorRm2 FK - rm | |  | | *New:*   |  | | --- | | **Doors** | | dr-lcMuseumName FK-lc  dr-lcName1 FK-lc  dr-lcname2 FK-lc | |  | |

The doors table doesn’t have many major change. The doorNum was decided to be not needed by our group so I will simply delete it.

To test this table I will simply select all the doors in the table and manually check that all the data is correct.

- Media table

|  |
| --- |
| **Media** |
| medIdAlpha FK-wk  medlIdNum FK-wk  medMedia |
|  |

For this table, it is identical to my current one and have the same primary keys and foreign keys. Therefore, I can transfer my data to this table easily.

**New tables:**

- WorksState table

|  |
| --- |
| **WorksState** |
| ws-wkIDNumeric FK-wk  ws-wkIDAlpha FK-wk  ws-workStateStartDate |
| ws-state  ws-workStateEndDate |

To input my data into this new table shouldn’t be very difficult. All of the primary key attributes can come from the works table. ws-state will be set to “In good condition” for all my works and ws-workStateEndDate will be set to null.

To test this table I can simply select the entire table and make sure that every work is “In good condition”.

- WorksInsuranceValue table

|  |
| --- |
| **WorksInsuranceValue** |
| wiv-wkIDNumeric FK-wk  wiv-wkIDAlpha FK-wk  wiv-workInsuranceValueStartDate |
| wiv-insuranceValue  wiv- workInsuranceValueEndDate |

To input my data into this new table shouldn’t be very difficult. wiv- workInsuranceValueEndDate will be set to null and all the other attributes will come directly from the works table.

To test this table, I will query this table and my old works table selecting the IaAlpha and IdNumeric and the insuranceValue, ordering by idAlpha and idNumeric from both tables. It will then be easy to see if there are any discrepancies.

**3. Qi Guo’s report**

I’ll update my Works, Locations, Exhibitions, ExhibitionsLocations, ExhibitionsWorks tables and add WorksInsuranceValue, WorksState, WorksKeepers tables, TempExhibitionsLocations tables before merger.

- Works table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Works** | | wkIDLetter  wkIDNumber | | wkName  wkType  wkSubtype  wkAuthor  wkCompletionDate  wkAcquisitionDate  wkInsuranceValue  wkDescription  wkOwnership  wkCharacter | | *New group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDinstitution | | ns-wk-worksName  ns-~~wk-type~~  ns-~~wk-subtype~~  ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-Wk-ownershipStatus  ns-Wk-theme  ns-Wk-subject  ns-Wk-culture  ns-Wk-colour  ns-wk-carType  ns-wk-transmission  ~~wk-question~~  ~~Wk-answer~~ | |

I plan to update my Works table from the following aspects. Remove wkInsuranceValue attribute from Works table and make a new temporal table named WorksInsuranceValue table to keep track of changes in insurance value from time to time. Add a new attribute to Works table named wk-question with a new domain called question, which is a varchar of length 50. “What is the character of this work?” will be inserted to wk-question. Since all group members have different custom attributes from assignment 2, we decide to add an attribute called answer composing of the data from custom attribute to Works table and an attribute with a name of question containing corresponding questions based on answers.

Tests planfor changes in Workstable includes querying all data in newly added attribute wk-answer, along with wk-IDAlpha, wk-IDNumeric from old Works table and new Works table to check whether those data match up and querying all data in wk-question to see if those data are all set to “What is the character of this work?”.

- WorksInsuranceValue table

*New group table:*

|  |
| --- |
| **WorksInsuranceValue** |
| wiv-wkIDNumeric FK-wk  wiv-wkIDAlpha FK-wk  wiv-workInsuranceValueStartDate |
| wiv-insuranceValue  wiv- workInsuranceValueEndDate |

A new table WorksInsuranceValuementioned above will be needed so that the changes in insurance value can be recorded. This table will contain the following attributes: wiv-wkIDAlpha, wiv-wkIDNumeric, wiv-workInsuranceValueStartDate, wiv-insuranceValue, wiv-workInsuranceValueEndDate.

Test plan for changes in WorksInsuranceValue table will be query all data in WorksInsuranceValue table to check if are temporal data are stored correctly.

- WorksState table

*New group table:*

|  |
| --- |
| **WorksState** |
| ws-wkIDNumeric FK-wk  ws-wkIDAlpha FK-wk  ws-workStateStartDate |
| ws-state  ws-workStateEndDate |

A new table WorksStateis needed to keep track of different work states, namely, damaged, stolen, in good condition. This table will contain the following attributes: ws-wkIDAlpha, ws-wkIDNumeric, ws-workStateStartDate, ws-workStateEndDate, ws-state.

Test plan for changes in WorksState table is querying all data in this table to make sure that there are no time gap between ws-workStateStartDate and ws-workStateEndDate.

- Locations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Locations** | | lcName  lcMuseum | | lcDimension  lcMinNumWork  lcMaxNumWork  lcIsAvailable | | *New group table:*   |  | | --- | | ns-**Locations** | | ns-lc-name  ~~lc-museumName~~  ns-inInstitutionName | | ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode  ~~lc-telStartDate~~  ~~lc-telEndDate~~  ~~lc-dimension~~ | |

The following changes are my plans for Locations table. Update domain to lc-dimension from numeric to a custom domain named dimension, which is a set of data namely, length, width and height. Add a new attribute lc-availableDate to take place of Boolean attribute named lcIsAvailable. A new domain called availableDate of type date will be applied to this attribute. Add a new attribute lc-telStartDate to keep track of temporary exhibition locations. A new domain called tempExhibitionLocationStartDate will be applied to this attribute. Add a new attribute lc-telEndDate to keep track of temporary exhibition locations. A new domain called tempExhibitionLocationEndDate will be applied to this attribute.

Tests plan for Locations table includes querying all data in lc-dimension to see if length, height and width are all recorded; querying all data in lc-telStartDate, lc-telEndDate to see if they are consistent with the start date and end date in TempExhibitionsLocationstable**.**

querying all data in lc-availableDate of Locations table and data in el-exEndDate from ExhibitionsLocations table to check if lc-available is the next day of el-exEndDate.

* WorksLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **WorksLocations** | | wlIDLetter FK-wk wlIDNumber FK-wk wlLocation  wlStartDAte | | wkEndDate  wllcMuseum | | *New:*   |  | | --- | | ns-**WorksLocations** | | ns-wl-lcName FK-lc  ns-wl-wkIDAlpha FK-wk  ns-wl-wkIDNumeric FK-wk  ~~wl-lcmuseumName FK-lc~~  ns-wklcinInstitutionName FK-lc,wk  ns-wk-workLocationStartDateTime | | ns-wk- workLocationEndDateTime | |

Since I have museum name in my own WorksLocations table, there is no big changes needed. All I need to do is to use data from wllcMuseum and store it to ns-wklcinInstitutionName as they contain identical information.

Testing plan for this table includes queries all information in my old WorksLocations table and see if data ns-WorksLocations.

-  ~~TempExhibitionsLocations table~~

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *~~New group table:~~*   |  | | --- | | **~~TempExhibitionsLocations~~** | | ~~tel-lcname FK-lc~~  ~~tel-lcMuseumName FK-lc~~  ~~tel-tempexhibitionLocationStartDate~~ | | ~~tel-sponsor~~  ~~tel-security~~  ~~tel-insurance~~  ~~tel-tempExhibitionLocationEndDate~~  ~~tel-tempExhibitionLocationAddress~~ | |

~~A new temporal table named TempExhibitionsLocations will be added to keep track of all the changes in locations for travelling exhibitions. This new table involves the following attributes: tel-lcName, tel-lcMuseumName, tel-tempExhibitionLocationStartDate, tel-sponsor, tel-security, tel-insurance, tel-tempExhibitionLocationEndDate, tel-tempExhibitionLoationAddress.~~

~~Tests plan for TempExhibitionsLocations table includes querying all data in this table to see if all 5 records are stored.~~

- ExhibitionsLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **ExhibitionsLocations** | |  | | elLocation FK-lc  elExhibition FK-ex  elexStartDate FK-ex  elexEndDate  ellcMuseum FK-lc | | *New group table:*   |  | | --- | | ns-**ExhibitionsLocations** | | ns-el-lcName FK-lc  ns-el-exName FK-ex  ns-el-exStartDate FK-ex  ns-el-inName FK-lc, ex | | ns-el-lctelStartDate  ns-el-lctelEndDate  ns-el-exEndDate | |

For my ExhibitionsLocations table, I will add two attributes, el-lctelStartDate, el-lctelEndDate, to keep track of all the changes in locations for travelling exhibitions.

Tests plan for ExhibitionsLocations table is querying all data in el-lcName, el-lctelStartDate, el-lctelEndDate from ExhibitionsLocations table and all data in tel-lcName, tel-lctempExhibitionLocationStartDate, tel-lctempExhibitionLocationEndDate in TempExhibitionsLocations table to check when el-lcName and tel-lcName are the same, which means that it is a travelling exhibition, data in el-lctelStartDate and tel-lctempExhibitionLocationStartDate, el-lctelEndDate and tel-lctempExhibitionLocationEndDate match up.

- Exhibitions table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Exhibitions** | | exName  exStartDate | | exDescription  exEndDate | | *New group table:*   |  | | --- | | ns-**Exhibitions** | | ns-ex-exhibitionName  ns-ex-exhibitionStartDate  ~~ex-museumName~~  ns-ex-inInstitutionName | | ns-ex-istraveling  ns-ex-exhibitionDescription  ns-ex-exhibitonEndDate | |

There are two changes needed to be done to Exhibitions table. Add a new attribute ex-museumName to keep track of museum name for normal exhibitions and travelling exhibitions. Add a new Boolean attribute ex-isTravelling to distinguish travelling exhibitions from normal exhibitions.

Tests plan for Exhibitions table consists of querying all data in ex-museumName from Exhibitions table and lc-museumName from Locations table to see if they are of the same amount; querying all data in ex-museumName, ex-exhibitionStartDate, ex-isTravelling from Exhibitions table and all data in lc-museumName, lc-telStartDate from Locations table to check if when ex-museumName is the same as lc-museumName and ex-exhibitionStartDate is five days after lc-telStartDate, ex-isTravelling is true; otherwise, ex-isTravelling is false.

- ExhibitionsWorks table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **ExhibitionsWorks** | |  | | ewExhibition FK-ex  ewIDLetter FK-wk  ewIDNumber FK-wk  ewStartDate FK-ex  ewEndDate | | *New group table:*   |  | | --- | | ns-**Works\_Exhibitions** | | ns-ew-exName FK-ex  ns-ew-wkIDAlpha FK-wk  ns-ew-wkIDNumeric FK-wk  ns-ew-exStartDate FK-ex  ~~ns-ew-exMuseumName FK-ex~~  ns-ew-inName FK-ex | | ns-ew-endDate | |

For ExhibitionsWorks table, I’ll add a new attribute named ew-exMuseumName for museum name of normal exhibitions and travelling exhibitions.

Tests planfor ExhibitionsWorkstable includes querying all data in ew-exName, ew-exStartDate, ew-exMuseumName from ExhibitionsWorks table and all data in ex-exhibitionName, ex-exhibitionStartDate, ex-museumName from Exhibitions table to check if when ew-exName and ex-exhibitionName, ew-exStartDate and ex-exhibitionStartDate are identical, ex-exhibitionName and ex-museumName are same.

- ~~WorksKeepers table~~

Institutions table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Owners** | |  | | ownwkIDLetter FK-wk  ownwkIDNumber FK-wk  ownName |        |  | | --- | | **LoanedWorks** | | lwwkIDLetter FK-wk  lwwkIDNumber FK-wk | | lwInstituteName  lwAddress  lwPhoneNumber  lwEmail  lwStartDateTime  lwEndDateTime | | *New group table:*   |  | | --- | | **ns-Works-Ownership** | | ns\_wo\_inName  ns\_wo\_IDAlpha  ns\_wo\_IDNumeric  ns\_wo\_inNameOwner  ns\_wo\_woStartDate | | ns\_wo\_woendDate | |

My Owners table and LoanedWorks table will be merged to a new table called ~~WorksKeepers~~ Institution table since those two old tables contain information regarding either owners or obligors for different works, which have similar functionality.

Tests plan for ~~WorksKeepers~~ Institution table is querying all data in ~~WorksKeepers~~ Institution table and all data in wk-IDAlpha, wk-IDNumeric from Works table to check if all works stored in Works table have associated record in ~~WorksKeepers~~ Institution table.

- Doors table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | drLocation1 FK-lc  drLocation2 FK-lc  drLcMuseum FK-lc | |  | | *New:*   |  | | --- | | ns-Locations-**Doors** | | ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-ld-lcinInstitutionName FK-lc  ns-ld-lcName1 FK-lc  ns-ld-lcName2 FK-lc | |  | |

Since my Doors table contains every data needed for team Locations Doors table, I can just use old data in Doors table.

Test plan for this table would be query Doors table and ns\_locations\_Doors table to see whether they are identical.

- Media table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Media** | | meIDLetter FK-lc  meIDNumber FK-lc  meMaterial | |  | | *New:*   |  | | --- | | ns-Works-Media | | ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-wm-wkIDAlpha FK-wk  ns-wm-wkIDNumeric FK-wk  ns-wm-wkIDInstitution FK-wk  ns-wm-material | |  | |

I have to add institution information of my own museum to media table so that new Works Media table got everything needed to find the corresponding Works.

Test plan for this table would be query all data in Works Media table and Media table to see if they are identical and if institution's name are all my museum name.

**4. Wei Zhang’s report**

Descriptions of what changes I should do for each of my table And how I will test that my changes worked.  
4.1. For the **Works** table, delete the attribute media, because we will create the media table to put media information in there.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **museum** | | muletter  munumber | | muname  mudescription  mutype  musubtype  Muauthor  mumedia  mudate\_complete  mudate\_accquire  muinsurance\_value  muownership  mucharacteristic | | *New group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDinstitution | | ns-wk-worksName  ns-~~wk-type~~  ns-~~wk-subtype~~  ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-Wk-ownershipStatus  ns-Wk-theme  ns-Wk-subject  ns-Wk-culture  ns-Wk-colour  ns-wk-carType  ns-wk-transmission  ~~wk-question~~  ~~Wk-answer~~ | |

~~Add the 2 new attributes called 'wk-question', 'wk-characteristic', Then in this way we can combine everyone's character of description. The 'wk-question' is about all member's characteristic definition. And the 'wk-characteristic' is the data value for the characteristic.   
And also create the domain for ‘wk-question’ as ‘question’ and make the domain type as varchar(100)   
In addition, make the attribute ’wk-answer’ as ‘answer’ and make the domain type as varchar(50)~~

I added a ‘institution name’ attribute to my new works table. The reason for this is to making a clearly identifier which identifies the belonging of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

Also I added new attribute ns-Wk-theme,ns-Wk-subject, ns-Wk-culture, ns-Wk-colour, ns-wk-carType, ns-wk-transmission according to the new designed classification systems to all works.

However here comes a problem when I trying to merge my old database to the new one.

One attributes from previous old table and the attribute from new designed table, both of them represent the same functionality, but their domains are different, which can not converted by psql automatically.

For example I only use year as completion date in my old database, but we decide to use year-date as completion date in the new database. Hence I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~~~To test if the changes to works table worked, just query this table by selecting the names of attributes which were deleted and the attributes were added..~~

To test the media attribute changes in the table query this table to select the wkIdAlpha, wkIdNum, wkAcqDate, and media from the old Works table and wiv-wkIdAlpha, wiv-wkIdNumeric, wiv-workInsuranceValueStartDate, wiv-insuranceValue from the new media table. I will check to make sure these values match up. To test the wk-question and wk-answer attributes I will select the wkIdAlpha, wkIdNum, and wkchatacteristic from the old Works table and wk-IDAlpha, wk-IDNumeric, and wk-answer from the new Works table. I want to make sure these values match. I will also select every wk-question from the new Works table to check if it is set to “Is the work abstract?”.

2. For the **Locations** table, add additional attribute called lc-telStartDate as the location start to use date,add additional attribute called lc-telStartDate as the location end to use date  
And also create the domain for ‘lc-museumname as ‘museumname’ and make the domain type as varchar(50)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Locations** | | lc-name | | lc-dimension  lc-min\_capacity  lc-max\_capacity | | *New:*   |  | | --- | | ns-**Locations** | | ns-lc-name  ~~lc-museumName~~  ns-inInstitutionName | | ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode  ~~lc-telStartDate~~  ~~lc-telEndDate~~  ~~lc-dimension~~ | |

Because in this way we can distinguish the same location name with different museum name. Add additional attribute called ‘lc-availabledate’ to indicate when and where the location will be available to use in future. Also create the domain for attribute ‘lc-availableDate’ as ‘availableDate’ as date.

Also I added new attribute ns-lc-length, ns-lc-height, ns-lc-width according to the new designed dimensions to the locations.

Also I added new attribute ns-lc-sponsor, ns-lc-security, ns-lc-insurance, ns-lc-streetAddress, ns-lc-city, ns-lc-country, ns-lc-postalCode. Because we want to merge the travelling- location information in the locations table.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~~~To test if the changes to locations table worked, just query this table by selecting the names of attributes which are added.~~

To test the changes to the lc-availabledate I will query the old location table selecting the length and width, and the new Locations table selecting the dimension to confirm the date has been mapped accordingly. To test the lc-telStartDate I will query all of the temporary locations in my traveling exhibition and confirm that their start dates and end dates add up to the total duration of the exhibition.  
  
3.For the **Exhibitions** table, add attribute ‘ex-isTravelling’ to indicate whether the exhibition is travelling exhibition or not.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitions** | | ex-exhibitionName  ex-exhibitionStartDate | | ex-ex\_descripton  ex-exhibitonEndDate | | *New:*   |  | | --- | | ns-**Exhibitions** | | ns-ex-exhibitionName  ns-ex-exhibitionStartDate  ~~ex-museumName~~  ns-ex-inInstitutionName | | ns-ex-istraveling  ns-ex-exhibitionDescription  ns-ex-exhibitonEndDate | |

Also create domain isTraveling as Boolean.  
Also drop table exhgala. Because the data in exhgala can get from exhibitionlocation. Exhgala table is redundant.

I added a ‘institution name’ attribute to my new exhibitions table. The reason for this is to making a clearly identifier which identifies the belonging of each exhibition. Hence in this way could avoid the conflicts with exhibitions hold by other museums which could have the same exhibition name and start date.

However here comes a problem when I trying to merge my old database to the new one.

One attributes from previous old table and the attribute from new designed table, both of them represent the same functionality, but their domains are different, which can not converted by psql automatically.

For example I only use varchar(100) as exhibition description in my old database, but we decide to use varchar(200) as exhibition description in the new database. Hence I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~ To test if the changes to exhibitions table worked, just query this table by selecting the added names of attributes and the deleted names of attribute. And display the table to see if the changes works. And also display the entire database to see if the exhgala table dropped.

To test the changes to the isTravelling I will query the old travelling exhibition table selecting the exhibitionname and the new exhibition table selecting the ex-exhibiitionname and ex-istravelling to confirm the exhibition name has been mapped accordingly.

4.For the **WorksLocations** table, add additional attribute called 'wk-lcmuseumNme' as primary key and it’s also a foreign key.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **workslocation** | | wl-museumetter FK-wk wl-museumnumber FK-wk wl-current\_location  wl-date\_in | | wl-date\_out  wl-time\_in  wl-time\_out | | *New:*   |  | | --- | | ns-**WorksLocations** | | ns-wl-lcName FK-lc  ns-wl-wkIDAlpha FK-wk  ns-wl-wkIDNumeric FK-wk  ~~wl-lcmuseumName FK-lc~~  ns-wklcinInstitutionName FK-lc,wk  ns-wk-workLocationStartDateTime | | ns-wk- workLocationEndDateTime | |

Because other member's museums can have the same locationname.   
Also change the domain workLocationStartDate to workLocationStartDateTime as timestamp;  
Also change the domain workLocationEndDate to workLocationEndDateTime as timestamp;

I added a ‘institution name’ attribute to my new works-Locations table. The reason for this is to making a clearly identifier which identifies the locations of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~~~ To test if the changes to workslocations table worked, just query this table by selecting the added names of attributes. And display the table to see if the changes works.~~

Test:If there is no error messages occurred after the each insert or update, that means there is no general mistake. Then I compared the number of records in old ‘**workslocation**’ table with the number of records in new ‘ns-works-locations’ table, if those two numbers of records are same, that means the transfer is successful.

Also, I picked 5 records from new ‘ns-works-locations’ table randomly, and compare them with corresponding records in old old ‘works-locations’ table, to see if the value in each attribute was transferred correctly.

5. For the **ExhibitionsWorks** table, delete the redundant attribute 'moveintime', because that information can get from the workslocations table.

~ To test if the changes to worksexhibitions table worked, just query this table by selecting the deleted name of attributes. And display the table to see if the changes works.  
  
6. For the **ExhibitionsLocations** table, add the additional attribute 'el-lcexmuseumname' as a primary key.  
Because other member's museum could have the same location name.   
Also add the attribute ‘el-lctelEndDate’ to store the end date of travelling exhibition.  
Create the domain tempExhibitionLocationStartDate as Date.  
Also add the attribute ‘el-lcexMuseumName’ to store the start date of travelling exhibition.   
Create the domain tempExhibitionLocationEndDate as Date.  
  
Because the exhibitionsLocations is a temporal table we need to keep history records of traveling exhibitions.

~~~ To test if the changes to exhibitionsLocations table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works.~~

Test:If there is no error messages occurred after the each insert or update, that means there is no general mistake. Then I compared the number of records in old ‘exhibitions-locations’ table with the number of records in new ‘ns-exhibitions-locations’ table, if those two numbers of records are same, that means the transfer is successful.

Also, I picked 5 records from new ‘ns-exhibitions-locations’ table randomly, and compare them with corresponding records in old old ‘exhibitions-locations’ table, to see if the value in each attribute was transferred correctly.

7. For the **Doors** table. Add the attribute dr-lcMuseumName as a primary key,   
because that is much easier to check which door connected to which doors in others museum.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | dr-door\_from FK-lc  dr-door\_to FK-lc | |  | | *New:*   |  | | --- | | ns-Locations-**Doors** | | ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-ld-lcinInstitutionName FK-lc  ns-ld-lcName1 FK-lc  ns-ld-lcName2 FK-lc | |  | |

added a ‘institution name’ attribute to my new works-Locations table. The reason for this is to making a clearly identifier which identifies the locations of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically. Also, I picked 5 records from new ‘ns-Locations-**Doors**’ table randomly, and compare them with corresponding records in old old ‘**Doors**’ table, to see if the value in each attribute was transferred correctly.

~ To test if the changes to doors table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works.  
  
~~8. For the~~ **~~WorksKeeper~~** ~~table. Add the attributes wkk-workKeeperStartDate and wwk-workKeeperenddate,wwk-status. Also make wkk-workKeeperStartDate and workKeeperenddate as primary keys.~~

*~~New:~~*

|  |
| --- |
| **~~WorksKeepers~~** |
| ~~wkk-workKeeperName~~  ~~wkk-wkIDNumberic FK-wk~~  ~~wkk-wkIDAlpha FK-wk~~  ~~wkk-workKeeperStartDate~~  ~~wkk-status~~ |
| ~~wkk-workKeeperAddress~~  ~~wkk-workKeeperEndDate~~  ~~wkk-email~~  ~~wkkphonenumber~~ |

~~Create the domain workKeeperStartDate as Date.  
Create the domain workKeeperEndDate as Date.  
Because the same work could be kept by the same person by different time.~~

~~~ To test if the changes to worksKeeper table worked, just query this table by selecting the added name of attributes. And query the table to display if the changes works.~~  
9. Also add a **Media** table in my database. Because other museum works have more than one materials for some works.

*New:*

|  |
| --- |
| ns-Works-Media |
| ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-wm-wkIDAlpha FK-wk  ns-wm-wkIDNumeric FK-wk  ns-wm-wkIDInstitution FK-wk  ns-wm-material |
|  |

Also add the attributesns-wm-wkIDAlpha, ns-wm-wkIDNumeric, ns-wm-wkIDInstitution, ns-wm-material into the materials table. And all of them are the primary keys.

added a ‘institution name’ attribute to my new ns-media table. The reason for this is to making a clearly identifier which identifies the medias of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same medias.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

Test: I compared the number of records in old ‘museum’ table with the number of records in new ‘ns-Works-Media’ table, if those two numbers of records are same, that means the transfer is successful. Also, I picked 5 records from new ‘ns\_item\_materials’ table randomly, and compare them with corresponding records in old old ‘museum’ table, to see if the value in each attribute was transferred correctly.

~~~ To test if the changes to Media table worked, display the database to see if the made changes works and query the table by selecting the added attributes~~  
  
~~10. Create a new table called~~ **~~TempExhibitionsLocations~~** ~~to store the temporal travelling exhibition information.~~

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *~~New:~~*   |  | | --- | | **~~TempExhibitionsLocations~~** | | ~~tel-lcname FK-lc~~  ~~tel-lcMuseumName FK-lc~~  ~~tel-tempexhibitionLocationStartDate~~ | | ~~tel-sponsor~~  ~~tel-security~~  ~~tel-insurance~~  ~~tel-tempExhibitionLocationEndDate~~  ~~tel-tempExhibitionLocationAddress~~ | |

~~Also made the attributes ‘tel-lcname’, ‘tel-lcMuseumName’, ‘tel-tempExhibitionLocationStartdate’ as primary keys  
Also made the attributes ‘tel-lcname’, ‘tel-lcMuseumName’ as foreign keys  
Also add the ‘tel-sponsor’, ‘tel-security’, ‘tel-insurance’, ‘tel-tempExhibitionLocationEnddate’, ‘tel-tempExhibitionLocationAddres  
attributes to store the corresponding information for the travelling exhibitions.~~

~~~ To test if the changes to TempExhibitionsLocations table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works.~~

11. Create a new table called **WorksState** that store the information about the works state. Add the attributes ‘ws-wkIDNumeric’, ‘ws-wkIDAlpha’, ‘ws-workStateStartDate’ as primary keys. Also add the ‘ws-state’, ‘ws-workStateEndDate’ attributes to store the state information. Also the ws-state could be stolen, damaged, restoration, in good condition.

|  |
| --- |
| **WorksState** |
| ws-wkIDNumeric FK-wk  ws-wkIDAlpha FK-wk  ws-workStateStartDate |
| ws-state  ws-workStateEndDate |

Create domain for attribute ‘ws-state’ called state as varchar(30); And create constrain for the state could be only one of (stolen, damages, restoration, in good condition)

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~ To test if the changes to worksState table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works

12. Create a new table called **WorksInsuranceValue** that store information about the works insurance information. Make the ‘wiv-wkIDNumeric’, ‘wiv-wkIDAlpha’, ‘wiv-workinsuranceValueStartDate’ as primary keys. Also add attribute ‘wiv-insuranceValue’, ‘wiv-workInsuranceVaueEndDate’ to record the information about the insurance of each works. Because the insurance value of each works could change.  
Also create domain for attribute’ wiv-insuranceValue’ called workInsuranceValueStartDate as date;  
Also create domain for attribute‘wiv-insuranceValue’ called workInsuranceValueEndDate as date;

|  |
| --- |
| **WorksInsuranceValue** |
| wiv-wkIDNumeric FK-wk  wiv-wkIDAlpha FK-wk  wiv-workInsuranceValueStartDate |
| wiv-insuranceValue  wiv- workInsuranceValueEndDate |

~ To test if the changes to worksInsuranceValue table worked, query this table by selecting the added name of attributes. And display the table to see if the changes works  
  
By using Dbvisualiser to display the E-R diagram of my new database and compare it with our planed E-R diagram. If they have same structure, then my database is correct.

**5. Nam Pham’s report**

I will have to change some part of my database before I can transfer the data to the new one. The major difference is that all of the tables in the database now contain the name of the museums that associate with the works or the locations or exhibitions, which can be used as a identification between the group museums.

Main table:

- Works table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **All\_works** | | wk-itemalpha  wk-itemnumber | | wk-name  wk-description  wk-type  wk-subtype  wk-author  wk-date\_complete  wk-date\_accquire  wk-insurance\_value  wk-claim  wk-potray\_with | | *New group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDinstitution | | ns-wk-worksName  ns-~~wk-type~~  ns-~~wk-subtype~~  ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-Wk-ownershipStatus  ns-Wk-theme  ns-Wk-subject  ns-Wk-culture  ns-Wk-colour  ns-wk-carType  ns-wk-transmission  ~~wk-question~~  ~~Wk-answer~~ | |

For this table, my current table already look identical to it, including the primary key, although there are still some differences.

The table that the group agree on will not contain the **insurance\_value** value of the item because that value will be record as the temporal data for the new table *WorksInsuranceValue*. So, I will need to add all the current item value to that table and set the same date range for all of them. Because there aren’t any change in the insurance value, so all the end\_date value will be null.

*Test*: I will create a query that choose the idnumber and idalpha from the new database and find it whether the data of the new database catch up. For the insurance value which are now in the new table, I will also make it one part of the query even when it can cause multiple row.

- Locations table

*New*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Galleries** | | lc-name | | lc-dimension  lc-min\_capacity  lc-max\_capacity | | ns-**Locations** |
|  | ns-lc-name  ~~lc-museumName~~  ns-inInstitutionName |
|  | ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode  ~~lc-telStartDate~~  ~~lc-telEndDate~~  ~~lc-dimension~~ |

For the location table, because we will combine the data of all the group members together, we need to distinguish each member location by adding the attribute **museumName** and make it as a part of the PK. So, even when our locations the same names, it does not violet the PK unique rule. Therefore, I just need to add the name of my museum for each row of location information.

Now, even though my old table already have the attributes like: dimension, min and max capacity, we decided to add more attributes for the locations of the traveling exhibitions later. So, the **telStartdate** and **telEndDate** will be used to tell, for a temporary location, when the exhibition will be held and end at that place. So, because I have stored the date information in the *exhibition\_info* table of my old database, I will use it for this 2 attribute. However, for the locations that are in the main museum, they will never held a traveling exhibition and therefore, this 2 attributes will be null for them.

The **availableDate** is used for when the location is not occupied and can be used. It is most likely the end date of the exhibitions. So, for now I will only need to update this attribute with the date after the exhibitions end, and then later I can create a trigger or a rule that can used this attribute to plan the exhibition in the future.

Test: I will create a query that include the location name to check whether or not the data catch up. The tables that will be used from the old database to check is the galleries table and the temporary location table.

- Exhibitions table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitions\_info** | | ex-exhibition\_id | | ex-theme  ex-ex\_descripton  ex-number\_of\_items  ex-start\_date  ex-end\_date  ex-location\_name | | *New:*   |  | | --- | | ns-**Exhibitions** | | ns-ex-exhibitionName  ns-ex-exhibitionStartDate  ~~ex-museumName~~  ns-ex-inInstitutionName | | ns-ex-istraveling  ns-ex-exhibitionDescription  ns-ex-exhibitonEndDate | |

In my old table, I used the **exhibition\_id** as the set of 3 characters to counter the problem that is 2 exhibitions can have the same name. However, our group decide to go with using the **exhibitionName** and the **exhibitionStartDate** as a part of the PK to counter the problem. We also add the **museumName** attribute to distinguish our exhibitions from each other. So, for me, I have to create a new set of PK and like in the Location tables, I will add my museum name for each row of exhibitions information. I will also have to drop the **exhibition\_id** as I won’t use it in the future.

I also have to drop the number\_of\_items and location\_name as they will be store elsewhere in the database. However, I will have another attribute **istraveling** which is a Boolean. My data will have difference from my group as for a travelling exhibition, they will use it as a whole, but I divided it into 5 exhibitions for each temporary location with the different start date. So, I can add **istraveling** to tell whether or not the 5 exhibitions is actually one or not, but the way I represent the traveling exhibition will be different from my groupmates.

Test: I will create a query that use the museum name to check whether or not the data in the new database catch up or not. The table from the old database that is used to compare is exhibitions\_info table.

- TempExhibitionsLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Temporary\_location** | | tel-name\_location FK-lc | | tel-sponsor  tel-address  tel-head\_of\_security  tel-insurance | | *New:*   |  | | --- | | **TempExhibitionsLocations** | | tel-lcname FK-lc  tel-lcMuseumName FK-lc  tel-tempexhibitionLocationStartDate | | tel-sponsor  tel-security  tel-insurance  tel-tempExhibitionLocationEndDate  tel-tempExhibitionLocationAddress | |

In my old table database, I forgot to add the situation that the temporary location can be reused in the future and this table can be used as a temporal data storage. Now, our group decide to add the **tempexhibitionLocationStartDate** attribute to know the time the temporary location is used and also make it a part of the PK. Moreover, because this table is connected to the *TempExhibitionsLocations* table it also need to add the **lcMuseumName** attribute to be a part of the PK and the FK to *locations* table as the PK in the *locations* table is changed. So, what I need to do here is to change the part of add the museum attribute and change the set of PK and the FK in the *Temporary\_location* table.

The **startDate** and **endDate** attributes in this table will have the also have the same value as the one in the *locations* table, so I can easily update them.

- WorksLocation table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **work\_location** | | wl-itemalpha\_ex FK-wk wl-itemnumber\_ex FK-wk wl-current\_location  wl-date\_in | | wl-date\_out  wl-time\_in  wl-time\_out | | *New:*   |  | | --- | | ns-**WorksLocations** | | ns-wl-lcName FK-lc  ns-wl-wkIDAlpha FK-wk  ns-wl-wkIDNumeric FK-wk  ~~wl-lcmuseumName FK-lc~~  ns-wklcinInstitutionName FK-lc,wk  ns-wk-workLocationStartDateTime | | ns-wk- workLocationEndDateTime | |

For this table, because of the change of PK set in *locations* table, I will need to add the **lcmuseumName** attribute to connect it with the location table.

The other thing I need to do is to combine the 2 attributes **date\_in** and **time\_in**, so that the attribute can show both date and time. After that I can make the result attribute as a part of the PK in this table. The same will be done for **date\_out** and **time\_out**, but we don’t need it as part of the PK. No further change need to be made for this table.

- ExhibitionsLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Exhibition\_location** | | el-Ex\_id FK-ex  el-current\_location FK-lc | | el-no\_of\_items | | *New:*   |  | | --- | | **ExhibitionsLocations** | | el-lcName FK-lc  el-exName FK-ex  el-exStartDate FK-ex  el-lcexMuseumName FK-lc, ex | | el-lctelStartDate  el-lctelEndDate  el-exEndDate | |

Because the set of PK in both *Exhibitions* table and *Locations* table are changed, I will need to add the more attributes to make a new PK. All of the data for these new attributes are already available in *locations* and *exhibitions* and therefore, can be updated from those tables. Because **Ex\_id** is not needed in the new database, I will also drop it.

The no\_of\_items attribute is also not needed for this table, so I will also drop it.

The 2 new special attributes like: **lctelStartDate** and **lctelEndDate** are used to store the date of the traveling exhibition. For this dates, I can find them both in the *exhibitions* table or the *temporary location* table, therefore I can update it from there. However, for the temporary locations, the value of this attribute will be null.

- ExhibitionsWorks table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Exhibition\_works** | | ew-itemalpha\_ex FK-wk  ew-itemnumber\_ex FK-wk  ew-ex\_id FK-ex | |  | | *New:*   |  | | --- | | **ExhibitionsWorks** | | ew-exName FK-ex  ew-wkIDAlpha FK-wk  ew-wkIDNumeric FK-wk  ew-exStartDate FK-ex  ew-exMuseumName FK-ex | | ew-endDate | |

This table has to be changed due to the change of the PK set from the exhibitions table. So, after I added all the new attributes, I can update the information from the exhibitions table.

- Institutions table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **donors** | | do-itemalpha\_don FK-wk  do-itemnumber\_don FK-wk | | do-donor\_name |        |  | | --- | | **loan\_institution** | | lo-itemalpha\_loan FK-wk  lo-itemnumber\_loan FK-wk | | lo-name  lo-address  lo-phonenum  lo-email  lo-start\_date  lo-end\_date | | *New:*   |  | | --- | | **Institutions** | | in\_Name | | In\_email  In\_ phoneNumber  in\_streetAddress  In\_city  in\_countrySubdivision  In\_country  in\_postalCode | |

The *institutions* table is the combination of the donors and *loan\_institution* table. Because this 2 tables have the same attributes and can work in the same way, all we need to do is to add another attribute **status** to know whether or not the items are lend or borrowed. All the data can be taken from this 2 old tables after the merge.

When I do these 2 tables, I didn’t think to make them the temporal data storage, so I only make the item identification to be the PK. However, in this project, making it a temporal data storage mean I have to change the PK set like the above, but all the information can be update easily.

- Doors table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | dr-door\_from FK-lc  dr-door\_to FK-lc | | dr-door\_name | | *New:*   |  | | --- | | ns-Locations-**Doors** | | ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-ld-lcinInstitutionName FK-lc  ns-ld-lcName1 FK-lc  ns-ld-lcName2 FK-lc | |  | |

The doors table doesn’t have many major change.

I will need to drop the **doorname** as it is considered unnecessary.

Then, because the PK set in the locations table is changed, I will need to add the **museum name** as part of the table PK. This will also be a way to distinguished the doors between the museum. All the data can be taken from *locations* table.

- Media table

|  |
| --- |
| ns-Works-Media |
| ~~dr-lcMuseumName FK-lc~~  ~~dr-lcName1 FK-lc~~  ~~dr-lcname2 FK-lc~~  ns-wm-wkIDAlpha FK-wk  ns-wm-wkIDNumeric FK-wk  ns-wm-wkIDInstitution FK-wk  ns-wm-material |
|  |

For this table, it is identical to my current one and have the same primary key as foreign key. Therefore, I can transfer my data to this table easily.

New tables:

- WorksState table

|  |
| --- |
| **WorksState** |
| ws-wkIDNumeric FK-wk  ws-wkIDAlpha FK-wk  ws-workStateStartDate |
| ws-state  ws-workStateEndDate |

This table is used to check the current state of the item which including: in good condition, stolen, damaged, etc and the time range they are in that condition. So, for all of my current item now, I will just make them into ‘in good condition’ state.

- WorksInsuranceValue table

|  |
| --- |
| **WorksInsuranceValue** |
| wiv-wkIDNumeric FK-wk  wiv-wkIDAlpha FK-wk  wiv-workInsuranceValueStartDate |
| wiv-insuranceValue  wiv- workInsuranceValueEndDate |

This table will keep the record of the table insurance value that will be changed by the time. However, because all my insurance value have not been changed yet, I will only keep the current insurance and only insert new one when it is changed.

**Part E:**

**Kevin Noonan:**

Once our team had decided upon a table file to create our new database I began to copy in the additions I had made to insert my data. I found as I was copying my data and testing it out that I was going to have to update some data from previous assignments in the beginning of the text file. I will go over the data that needed to be updated first and then cover the insertions into the new database.

**Updates to the old database:**

The first thing I discovered was that I had named my own museum wrong in my previous database so I update the works and owners table to fix this error. I realized as I was trying to insert my exhibitions table that I did not declare a traveling exhibition to be associated with the works that went to temporary locations. Due to this, I created a new exhibition and set the works to that exhibition in my exhibition works table. I then inserted each traveling location with the exhibition into my exhibition location table. This solved my exhibition error and I was able to create queries properly using the designed references to the tables. The next error was discovered as I was trying to copy my database into the team database. I realized that some of my donors names were duplicates to my team members. I decided to update the donors names so that I would not get the duplicate key error in the team database.

One thing I did know that I had to change from assignment three was the addition of an end date timestamp to my works locations table. This was difficult to add to the table because of the temporal data style. In order to solve the problem I had to create an order number to assign each row in the table a number. Additionally each row had to be ordered from the work id, num id and the start date. After these were ordered a temporary table was made which was a copy of the works locations table. In order to set the end date though I had to set the order number to the previous row. Once this was setup I was able to add in the end date and the start date of the next time the location of the work happens to it’s end date. After this was done I drop the order column as it’s meaningless to the data.

Some smaller errors I found were the locations availability dates. They were not aligning with the dates that they were supposed to so I had to update the locations to the proper dates. A known fix that needed to happen was the acquisition date of my works not being of type date. To fix this I altered the table and the column and changed it to type date with a cast. While I was doing my locations table I noticed that I could not properly query my temporary locations and this was because I had not properly reference my temp location table in my previous database. I referenced the proper foreign keys to the table so that I was able to query properly. One of the major issues I had to solve prior to the insertion of my data was in temp location table. It didn’t have the correct information so I had to manually add the works that were associated with the location so it could be queried properly in the future. Another update I had to make to my previous database was with the on loan table. They didn’t properly reference my works id’s so I referenced the table to the works table and added in the work id’s. The final update was an issue with my works locations table. In previous assignments I had given a location to ‘Sold’, ‘Onloan’ and ‘Temploc’ these were incorrect so I gave them a static value ‘Not in Museum’ and deleted them from the locations table.

**Institutions Table:**

Following my design from part D, I inserted all the institutions and their information and it went exactly as planned. There were no additional changes to made to the design and all of the data was inputted without error. I tested this table by using a query on the entire table to make sure that the data was properly inputted.

**Works Table:**

Following my design from part D, i inserted the institutions and the proper null values and excluded the values that were being added to temporal tables. The only addition I made was to check to make sure that the institution name was the name of my museum. Other then that there were no more changes and the data was inputted without error. I tested the data by querying the table with the institutions table so that each institution had it’s associated info

**Locations Table:**

As i said it in part D each value for the temporary locations in locations had to be manually

Implemented. I first inserted the museums galleries that actually took place in the museum. I then created my manual insert values for each individual temporary location and travelling period. I then updated each individual temporary location and travelling period to the appropriate availability date as they were all the same date before. After this was fixed the data was complete and I was able to insert it into the new schema without error. I tested this table by querying it with the institutions table because with this I was able to see the information for the location and institution in one view.

**Exhibitions Table:**

The exhibitions table went as planned by inserting the false value for the non traveling exhibitions and then creating an update after for the traveling exhibition and setting the is traveling value to true. The data inputted perfectly fine and the values that were reperesented were correct no other changes needed to be made. I tested this table by querying it with the institutions table so that the proper information for the institute was displayed for each exhibition.

**Transactions Table:**

The transactions insertion went as expected. I created five different insert statements with each one representing a type value that consisted of ‘Buy’, ‘Sell’, ‘Borrow in’, ‘Lend in’, ‘Lend Out’. The only thing that needed to be changed was because of a duplicate key error i needed to make sure that the works ids were eqaul to either the old works table ids or the on loan table work ids. The data inputted fine and it was all correct. The data was tested by querying it with the institutions table to see each giver and recievers info along with the work that they had type.

**Works Ownership Table:**

Following the design in part D this went as planned except for a duplicate key error. I had to make sure that the work ids were equal to the work ids from the owners table. After this the data inputted fine. The data was tested by querying the table with the institutions table. This way we could check each owners information along with the work that they owned.

**Works Exhibitions Table:**

Following the design in part D this insertion went as planned after solving the exhibition issue that I explained in the beginning. With the exhibitions properly located each work was able to be associated with its respected exhibition including the traveling exhibition. With this the data was inserted perfectly fine. I tested this data by querying the table with the works table to make sure that each work in the works table had an exhibition to go along with it.

**Exhibitions Locations Table**

Following the design in part D this insertion went as planned. I created two separate inserts one for if the exhibition was travelling and one if it was not. The only addition I had to make was a change to the time on the insert where the exhibition was travelling. I had to add twenty-one days to my availability date to represent the period the exhibition was on display. Once this was solved the data was correct and the insertion worked. I tested this data by querying the table with the locations table to check to see if the exhibition was in a museum or if it was traveling.

**Works Locations Table:**

Creating this table did not go as expected according to my part D. This was a result of the exhibitions error that was talked about in the beginning. After I had solved that error I realized i need to create two separate inserts. One insert was for regular locations with their respective works and the other was for the traveling exhibitions locations and their respective works. After this was figured out I had to create some updates to adjust the times on the traveling periods. This was easy enough and I just created six different updates for six different traveling periods. Once this was figured out the data inserted just fine. I tested this data by querying the table with the locations table and making sure that each work was with it’s location traveling or not traveling.

**Works Value Table:**

Inserting this table went as expected according to my part D design except for a duplicate key issue. This was resolved by making the old works table id’s equal to the new works table ids. Other then that there were no other issues and data was inserted into the table. I tested this data by querying with the works table so that each insurance value was matched with works information and its respective ids.

**Works State Table:**

Following my design in part D inserting the state table worked perfectly. There were no errors and the insurance value matched up exactly with its respective work id’s. No errors were involved in the creation of this table. I tested this data by querying with the works table so that each state was associated with the works information and it’s respective ids.

**Works Media Table:**

Following my design in part D inserting the media table worked perfectly. There were no errors and the media matched up exactly with its respective work id’s. No errors were involved in the creation of this table. I tested this data by querying with the works table so that the media was with it’s work ids information.

**Locations Doors Table:**

The insertion of this table didn’t go as planned in my part D. Rather I decided to insert the location name separately and then update the values when their respective match was set to the other room. Using this method I added the doors I wanted and worked without error. I tested this data by querying with the locations table and checked that each location was with it’s museum name.

**Transferring to the team database:**

After all of the tables were created I copy them all to text files with a text file called ‘copy to text.txt’. I then took those files and created another file called ‘texttodatabase.txt’ which copied all the text files into the new database. This method had a few errors that I had to resolve with naming issues mention in the beginning of this report but after they were fixed the text file worked perfectly fine and my data we inserted into the team database.

**Daniel Morris:**

After I had fixed all of the issues and discrepancies from my assignment 3 I started by first changing the temporal type in my works locations table. Originally I had my works locations table as a closed - open temporal type but our group decided that we were going to go with a closed - closed temporal data style. I discovered that doing this was very problematic and difficult. I ended up having to order the table by the id, and start date and assign each row a number. I then made a temporary table where each row was set to the row previous to it in the original table. This allowed me to compare the two tables and set the start date of the next location of a work to its end date and set the end date of a work to null if the next work in this ordered table is not the same ID.

Upon fixing this issue I stumbled upon more errors from my assignment 3. I discovered that the works I loaned out were actually almost all in exhibitions and could not be loaned out. This led me to manually select different works that were not currently in an exhibition to be loaned out.

Once that was fixed I then imported our team’s create tables file to create the new schema in my database. This file contained all of the domain and table definitions for our teams schema.

Once the schema was inserted I began by filling my institutions table. I decided to insert the data into this table manually because of the simple fact that I did not have this data in my previous structure. I had to make institutions for my museum, the cartoon characters my museum borrowed from and the museums my museum lent works to. It was also necessary to create a unknown institution because of reasons to be discussed later on.

The next table I inserted into was the works table. This insertion firstly involved inserting attributes from my old works table and inserting them straight into the new schema. I then selected works from my old works table that under my new attribute stated that they were aboriginal. I used this to add to the new schema attribute wk\_culture. To test this table I selected all works from my old works table where the isAboriginal attribute was set to ‘Yes’ and the new works table where the wk\_culture attribute was set to ‘Australian Aboriginal’. These queries matched each other.

The next table I inserted was the Works\_Value table. This table was very simple only selecting the ID, ‘MCA’, the date of acquisition, and the value from the old works table. To test this table I queried both tables ordering by their ID’s and checked that the value, and acqDates were the same. They were.

The next table I inserted was the Works\_Media table. This was the simplest table to transfer since it was exactly the same as my current media table except for adding an attribute that was set to ‘MCA’. To test this I simply queried the whole table and checked that every row had ‘MCA’ as it’s institution name.

The next table I inserted was the Works\_Ownership table. To insert into this table I used two insert statements. The first one selected all works from my museum that were owned by the museum and used their acquisition date to set the date where the museum owned these works. The second insert statement selected all works from my database that were not owned by my museum. The owner’s date of these works was obtained by the owners table from my previous schema. To test this table I queries the new table where the owner is ‘MCA’, and the works table ordering by ID to see if the date of acquisition matched the date of ownership. It did. I then did the same against the owners table to see if their dates matched. They did.

The next table I inserted was the Transactions table. I inserted into this table using 5 insert statements. The first one added the transactions for the museum buying 15 works on October 25th. This implementation was not dealt with correctly in my assignment 3 so I had to use this date to decipher which works they were. I decided to make all these works come from one owner so that I would not have to manually enter 15 different owners. The second one inserts all the data that is borrowed in from various cartoon characters, this insert simply joins works with worksOwnership to find the date and ownership status of the work. The third and fourth inserts simply add the lend in and lend out functionality of the workLoan table selecting either the end date or the start date. The fifth insert adds the works that were moved to the location ‘Sold’ in the worksLocations table and sets the tr\_type to be ‘Sold’ as well. To test this table I simply joined this table with the worksOwnership table and confirmed that that the Buy was implemented correctly. I then joined this table with the workLoan table to confirm that the loan operations were also implemented correctly.

The next table I implemented was the Locations table. Firstly I inserted all the data from my rooms table. I then needed to update the height of all the locations since I did not implement this into my rooms table previously. I then manually inserted all the information for the locations of the institutions that I lent works out to since I will need to store the location of the works there. After that I had to also manually enter the data for the traveling exhibition’s locations since my assignment 3 did not properly accommodate for this.

The next table I implemented was the exhibitions table. To insert into this I simply inserted directly from my exhibitions table. I then set the traveling exhibitions isTraveling attribute to true. Since this table is very small I just tested it by selecting the whole table and comparing.

The next table I implemented was the Exhibitions\_Locations table. Once again I started by inserting the data from my exhibitionsLocations table. I then manually entered the locations and dates that each of the traveling exhibition locations held the exhibition since this data was not implemented into my assignment 3. To test this table I queried it ordering by the start date and made sure that none of the start and end dates overlapped when the exhibitions were in the same location.

The next table I implemented was the Exhibitions\_Works table. This table was very simple to implement since I just had to have one insert statement adding my museum name and exhibition start and end date to this table. To test this I joined this table with my exhibitions table and made sure the start date and end dates matched.

The next table I implemented was the LocationsDoors table. This table was very easy to implement since I just had to select two attributes from my current doors table and add my museum name. To test this table I simply selected the whole table and made sure the data was correct.

The final table I implemented was the Works\_Locations table. I started implementing this table by first inserting everything from my worksLocations table. I then realized that the first entry of every work had an end date of null, I fixed this by setting the end date of these worksLocations entries to the start date of the first exhibition the specified work is in. I then added all the locations of the traveling exhibitions that were not implemented in my worksLocations table. I did this by selecting from the exhibitions locations table. To test this table I selected all columns from works\_locations ordering by ID, startDate then making sure that each work does not run into discrepancies where it is in two locations at once for example, none occurred.

**Qi:**

-Institutions table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Owners** | |  | | ownwkIDLetter FK-wk  ownwkIDNumber FK-wk  ownName |        |  | | --- | | **LoanedWorks** | | lwwkIDLetter FK-wk  lwwkIDNumber FK-wk | | lwInstituteName  lwAddress  lwPhoneNumber  lwEmail  lwStartDateTime  lwEndDateTime | | *New group table:*   |  | | --- | | **ns\_Institutions** | | ns\_in\_inName | | ns\_in\_email  ns\_in\_phoneNumber  ns\_in\_streetAddress  ns\_in\_city  ns\_in\_countrySubdivision  ns\_in\_country  ns\_in\_postalCode | |

My Owners table and LoanedWorks table will be merged to a new table called Institution table since those two old tables contain information regarding either owners or obligors for different works, which have similar functionality. I added 15 potentially borrowed works and 15 long-term borrowed works to this table by querying my old works table with type equals to ‘potential works or borrowed works’.

I tested Institution table by querying all data in Institution table and all data in wk-IDAlpha, wk-IDNumeric from Works table to check if all works stored in Works table have associated record in Institution table.

-Works-Ownership table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Owners** | |  | | ownwkIDLetter FK-wk  ownwkIDNumber FK-wk  ownName |        |  | | --- | | **LoanedWorks** | | lwwkIDLetter FK-wk  lwwkIDNumber FK-wk | | lwInstituteName  lwAddress  lwPhoneNumber  lwEmail  lwStartDateTime  lwEndDateTime | | *New group table:*   |  | | --- | | **ns\_Works\_Ownership** | | ns\_wo\_inName  ns\_wo\_IDAlpha  ns\_wo\_IDNumeric  ns\_wo\_inNameOwner  ns\_wo\_woStartDate | | ns\_wo\_woendDate | |

My Owners table and LoanedWorks table are merged to a new table called ns\_Works\_Ownership table since those two old tables contain information regarding either owners or obligors for different works, which have similar functionality.

I tested Institution table by querying all data in Institution table and all data in wk-IDAlpha, wk-IDNumeric from Works table to check if all works stored in Works table have associated record in Institution table. I also checked whether there are 15 works ownership changes on 2016-10-25 as that was the date that I purchased 15 new works.

-Works table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Works** | | wkIDLetter  wkIDNumber | | wkName  wkType  wkSubtype  wkAuthor  wkCompletionDate  wkAcquisitionDate  wkInsuranceValue  wkDescription  wkOwnership  wkCharacter | | *New group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDInstitution | | ns-wk-worksName  ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-wk-theme  ns-wk-subject  ns-wk-culture  ns-wk-colour  ns-wk-carType  ns-wk-transmission | |

I updated my Works table from the following aspects. Removed wkInsuranceValue attribute from Works table and made a new temporal table named WorksValue table to keep track of changes in insurance value from time to time. Added new attributes of all types, subtypes and custom types from team mates to Works table named ns-wk-theme, ns-wk-subject, ns-wk-culture, ns-wk-colour, ns-wk-carType, ns-wk-transmission. Since ns-wk-theme, ns-wk-subject, ns-wk-culture apply to my museum, I just put my type, subtype and custom attribute data into those three columns. ns-wk-IDInstitution is a newly added attribute. I put my museum name “Qi’s Museum”, as all works in my museum are all related to me.

I tested changes in Workstable by querying all data in newly added attribute wk-answer, along with wk-IDAlpha, wk-IDNumeric from old Works table and new Works table to check whether those data match up. And I queried all ns-wk-IDInstitution to see if they are all “Qi’s Museum’.

- WorksValue table

*New group table:*

|  |
| --- |
| **ns\_WorksValue** |
| ns\_wv\_IDAlpha FK-wk  ns\_wv\_IDNumeric FK-wk  ns\_wv\_inName FK-wk  ns\_wv\_startDate |
| ns\_wv\_insuranceValue  ns\_wv\_endDate |

A new table WorksInsuranceValuementioned above was created so that the changes in insurance value can be recorded. Since there is no changes for insurance value in my own database yet, I can just use old insurance value from Works table and set start date as works acquisition date, along with end date to be null.

I tested for changes in WorksInsuranceValue table by querying all data in WorksInsuranceValue table to check if are temporal data are stored correctly.

- WorksState table

*New group table:*

|  |
| --- |
| **Works\_State** |
| ns\_ws-IDNumeric FK-wk  ns\_ws-IDAlpha FK-wk  ns\_wsStartDate  ns\_inName |
| ns\_ws-state  ns\_ws-wsEndDate |

A new table WorksStatewas created to keep track of different work states, namely, damaged, stolen, restoration, in good condition. As there is no changes in works states yet, I can just put every works inside with state information to be ‘in good condition’.

I tested changes in WorksState table by querying all data in this table and Works table to make sure that number of works state equals to number of works.

- Locations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Locations** | | lcName  lcMuseum | | lcDimension  lcMinNumWork  lcMaxNumWork  lcIsAvailable | | *New group table:*   |  | | --- | | ns-**Locations** | | ns-lc-lcname  ns-lc\_inName FK-in | | ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode | |

I made the following changes for Locations table. Updated domain to lc-dimension to three separate attributes, namely, ns-lc-length, ns-lc-height, ns-lc-width. Add a new attribute ns-lc-availableDate to take place of Boolean attribute named lcIsAvailable. I queries Exhibitions Locations table to get the available date if the location names are the same. I added availability date, sponsor, security, insurance, street address, city, country subdivision, country and postal code to this table in order to keep track of information of travelling exhibitions. I queried WorksLocations table to get available date for each locations (ns\_lc\_availabilityDate for galleryD is 2016-12-09). I also add ed5+6 travelling exhibition locations to this table and 6 loaned works.

I tested Locations table by querying all data in lc-dimension to see if length, height and width are all recorded; querying all data in ns-ls-availabilityDate to see if it is the day after the last exhibition that happens in the location.

* WorksLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **WorksLocations** | | wlIDLetter FK-wk wlIDNumber FK-wk wlLocation  wlStartDAte | | wkEndDate  wllcMuseum | | *New:*   |  | | --- | | ns\_**Works\_Locations** | | ns\_wl\_lcName FK-lc  ns\_wl\_IDAlpha FK-wk  ns\_wl\_IDNumeric FK-wk  ns-wl\_inName FK-lc,wk  ns-wl-wlStartDateTime | | ns-wl-wlEndDateTime | |

Since I have museum name in my own WorksLocations table, there is no big changes needed. All I did was to use data from wllcMuseum and store it to ns-wklcinInstitutionName as they contain identical information. I made a mistake when I did my assignment 1-3 by putting 2015-01-01 as the start date for every work that initially goes to storage. I fixed this part by querying all works acquisition date in works table and set the initial date going to storage to the acquisition date.

I tested this table by querying all information in my old WorksLocations table and see if data ns-WorksLocations. What’s more, I checked whether the first works location record for each work is identical to works acquisition date.

- ExhibitionsLocations table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **ExhibitionsLocations** | |  | | elLocation FK-lc  elExhibition FK-ex  elexStartDate FK-ex  elexEndDate  ellcMuseum FK-lc | | *New group table:*   |  | | --- | | ns\_**Exhibitions\_Locations** | | ns\_el-lcName FK-lc  ns\_el-exName FK-ex  ns\_el-exStartDate FK-ex  ns\_el-inName FK-lc, ex | | ns\_el-travelingStartDate  ns\_el-travelingEndDate  ns\_el-exEndDate | |

For my ExhibitionsLocations table, I added two attributes, ns\_el-exStartDate, ns\_el-exEndDate, to keep track of all the changes in locations for travelling exhibitions.

I tested ExhibitionsLocations table by querying all data in ns\_el-lcName, ns\_el-exStartDate, ns\_el-exEndDate from ExhibitionsLocations table and all data in tel-lcName, tel-lctempExhibitionLocationStartDate, tel-lctempExhibitionLocationEndDate in TempExhibitionsLocations table to check when el-lcName and tel-lcName are the same, which means that it is a travelling exhibition, data in el-lctelStartDate and tel-lctempExhibitionLocationStartDate, el-lctelEndDate and tel-lctempExhibitionLocationEndDate match up.

- Exhibitions table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **Exhibitions** | | exName  exStartDate | | exDescription  exEndDate | | *New group table:*   |  | | --- | | ns-**Exhibitions** | | ns-ex-exName  ns-ex-exStartDate  ns-ex-inName | | ns-ex-isTraveling  ns-ex-exDescription  ns-ex-exEndDate | |

Two changes were made to Exhibitions table. Added a new attribute ns-ex-inName to keep track of museum name for normal exhibitions and travelling exhibitions. Add a new Boolean attribute ex-isTravelling to distinguish travelling exhibitions from normal exhibitions.

I testedExhibitions table by querying all data in ex-museumName from Exhibitions table and lc-museumName from Locations table to see if they are of the same amount; querying all data in ex-museumName, ex-exhibitionStartDate, ex-isTravelling from Exhibitions table and all data in lc-museumName, lc-telStartDate from Locations table to check if when ex-museumName is the same as lc-museumName and ex-exhibitionStartDate is five days after lc-telStartDate, ex-isTravelling is true; otherwise, ex-isTravelling is false.

- ExhibitionsWorks table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **ExhibitionsWorks** | |  | | ewExhibition FK-ex  ewIDLetter FK-wk  ewIDNumber FK-wk  ewStartDate FK-ex  ewEndDate | | *New group table:*   |  | | --- | | ns\_**Works\_Exhibitions** | | ns\_we-IDAlpha FK-wk  ns\_we-IDNumeric FK-wk  ns\_we\_exName FK-ex  ns\_we\_inName FK-ex, wk  ns\_we\_exStartDate FK-ex | | ns\_we\_exEndDate | |

For ExhibitionsWorks table, I added a new attribute named ns\_we-inName for museum name of normal exhibitions and travelling exhibitions.

I tested ExhibitionsWorkstable by querying all data in ew-exName, ew-exStartDate, ew-exMuseumName from ExhibitionsWorks table and all data in ex-exhibitionName, ex-exhibitionStartDate, ex-museumName from Exhibitions table to check if when ew-exName and ex-exhibitionName, ew-exStartDate and ex-exhibitionStartDate are identical, ex-exhibitionName and ex-museumName are same.

- Doors table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | drLocation1 FK-lc  drLocation2 FK-lc  drLcMuseum FK-lc | |  | | *New:*   |  | | --- | | ns\_Locations\_**Doors** | | ns\_ld\_inName FK-lc  ns\_ld\_lcName1 FK-lc  ns\_ld\_lcName2 FK-lc | |  | |

Since my Doors table contains every data needed for team Locations Doors table, I can just use old data in Doors table.

I tested this table would be query Doors table and ns\_locations\_Doors table to see whether they are identical.

- Media table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Media** | | meIDLetter FK-lc  meIDNumber FK-lc  meMaterial | |  | | *New:*   |  | | --- | | ns\_Works\_Media | | ns\_wm-IDAlpha FK-wk  ns\_wm-IDNumeric FK-wk  ns\_wm-inName FK-wk  ns\_wm-material | |  | |

There is a wm\_inName attribute in team Works\_Media table, which is the institution name.   
So I need to update ns\_Works\_Media table with all old data from my Media table along with "Qi's Museum" for wm\_inName attribute. I didn't set any primary keys to my Media table so I didn't realize there are two duplicate rows of data. So I have to use distinct keyword to get unique data from my Media table.

I tested this table by querying all data in Works Media table and Media table to see if they are identical and if institution's name are all my museum name.

**- Loading the data into the team database**

I used copy to command to export all tables and then use copy from command to import my tables to team database. There were only three duplicate data, which were all in Institutions table. Those are 3 cartoon names. Since those three names are used for owners that I bought works from. I didn’t influence my works ownership table. So I just removed duplicate cartoon names from my Institutions table.

**Nam:**

To change my current database to the team database, I have to rely on the information I have on my old database. So, after my team agree on the database design, we have created the text files that can create the tables and domains of the team database. Then, I will use that files to create the team database and use the SQL code to transfer all my data from the old design to the new one. Then after I done everything, I will copy all the data to text files, fix some errors and update them on the team database. This is the more detail of how I have done the work in each table:

**1)** **Institutions TABLE:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **donors** | | do-itemalpha\_don FK-wk  do-itemnumber\_don FK-wk | | do-donor\_name | |  | | **loan\_institution** | | lo-itemalpha\_loan FK-wk  lo-itemnumber\_loan FK-wk | | lo-name  lo-address  lo-phonenum  lo-email  lo-start\_date  lo-end\_date | | *New:*   |  | | --- | | **Institutions** | | in\_Name | | In\_email  In\_ phoneNumber  in\_streetAddress  In\_city  in\_countrySubdivision  In\_country  in\_postalCode | |

As you can see, the institutions table is the combination of 2 tables donors and loan\_institution from my old data base.

For the loan\_institution table, it is easy to get the information from it when it already has the name of the institution, the phone number and the email. The only think I had to change in this table is that I will have to divide the address to smaller part like: street address, city, country subdivision, country, and postal code, but I can update it manually very fast because there are only 5 loan institutions. It also doesn’t have any duplicate names so, but I still use distinct just in case.

For the donors table, as you can see, it lacks everything when it only has the institutions name. Not only that, some name is duplicate because one institution can lend you 2 items. So, right after I get the name for all the institutions, I had to update the other information for them. And because there are 32 donors, it was a long process. Though, most of my donors is not real, so the information is fake anyway.

Because the institution table has the institution name as primary key, I have me to realize there is duplicate value inside my donors table when I insert from it. After everything was done, I just need to query it to show all the table and get the number of rows.

**2)** **Works TABLE:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current table:*   |  | | --- | | **All\_works** | | wk-itemalpha  wk-itemnumber | | wk-name  wk-description  wk-type  wk-subtype  wk-author  wk-date\_complete  wk-date\_accquire  wk-insurance\_value  wk-claim  wk-potray\_with | | *New group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDinstitution | | ns-wk-worksName  ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-Wk-ownershipStatus  ns-Wk-theme  ns-Wk-subject  ns-Wk-culture  ns-Wk-colour  ns-wk-carType  ns-wk-transmission | |

For this new table, the first problem occur is that the new attribute that will represent type and subtype. This is the result from combining the type and subtype of all the team members and now, the table will have new attributes which represent the team members type and subtype with a new meaningful name. Though, this is not a big proplem because I can make them null when I use the insert transaction.

Beside from the problem above, all my data from the old table is sufficient enough for the new one and it didn’t take me much time to transfer all the data from the old table to the new one.

The other change is that now, because I have combine thee museum with other teammate, my items now had to associate with my museum name from now on. However, this is also not a big issue when it can be easily done in the insert form.

So, even when this new table have connection with the institutions table, because I have insert all the data in one swoop, it doesn’t cause any problem.

Finally, I do the query and have all the data in organize and don’t cause any error.

**3)** **Locations TABLE:**

*New:*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Galleries** | | lc-name | | lc-dimension  lc-min\_capacity  lc-max\_capacity |        |  | | --- | | **Temporary\_location** | | tel-name\_location FK-lc | | tel-sponsor  tel-address  tel-head\_of\_security  tel-insurance | | **Locations** |
| ns-lc-name  ns-inInstitutionName |
| ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode |

The locations table of the team database now is a combination of mu 2 previous tables: galleries and temporary\_location and they cause me to have some change.

In my old database, the temporary location is the sub-table of the galleries table and therefore, the galleries table have the names of all the temporary locations. So, when I insert the data into the new table, it will also contain the name of the temporary location, but it won’t take other information because I need to change them before I insert the new one.

For the galleries table, I can only take the name of the location, and the min and max capacity because the dimension will be divided by 3 values in the new table. Not only that, I also add my museum name when I was performing the insert. After I have done the insert for this normal location, I had to update the dimension into 3 new values and also the availability date, which is the next day of its last exhibition (I can get this information from the old exhibitions locations table). So, for this table, I had to double check the value before I can update it to the new data.

For the temporary location, I have already got all theirs name in the new table, the next thing I need to do is to update the value manually because now the address one again had to be separated into new attributes. The update did not take that much time when I can get them from the old tables, even when I had to do it manually.

Finally, I do the query to show the whole table and have all the data in organize and don’t cause any error.

**4)** **Exhibitions TABLE:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitions\_info** | | ex-exhibition\_id | | ex-theme  ex-ex\_descripton  ex-number\_of\_items  ex-start\_date  ex-end\_date  ex-location\_name | | *New:*   |  | | --- | | **Exhibitions** | | ns-ex-exhibitionName  ns-ex-exhibitionStartDate  ns-ex-inInstitutionName | | ns-ex-istraveling  ns-ex-exhibitionDescription  ns-ex-exhibitonEndDate | |

In my old table, it got most of the necessary information for new one like the name of the exhibition, thee description, the start and end date. So, I only need to deal with 2 new attribute which are: isTraveling and the museum name. The museum once again, is not a major problem in my data when I can deal with it in the insertion like the other previous tables. The one I had to deal with was the isTraveling attribute.

Because isTraveling will show whether or not the exhibitions is going to be traveling or not and it is a Boolean, I have to do the insertion 2 times for 2 kinds of exhibition. It is not a major problem because this 2 insertion have the same format, but the only difference is the value of isTraveling is true or false.

This table have not much to check when it doesn’t have much data. I only need to have a query that show the whole table to check whether it is organized or not.

**5)** **Works\_Locations TABLE:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **work\_location** | | wl-itemalpha\_ex FK-wk  wl-itemnumber\_ex FK-wk  wl-current\_location  wl-date\_in | | wl-date\_out  wl-time\_in  wl-time\_out | | *New:*   |  | | --- | | **WorksLocations** | | wl-lcName FK-lc  wl-wkIDAlpha FK-wk  wl-wkIDNumeric FK-wk  wklcinInstitutionName FK-lc,wk  wk-workLocationStartDateTime | | wk- workLocationEndDateTime | |

Because my old table have the separate attribute for the time and the date, I had to combine them together, so that I can insert them into the new database. To do this, I had to change my old table by adding a new attribute that now can store the timestamp value. Then I had to do the most time-consuming part of my database, that is I had to combine and update them manually. Though it is lucky that there are many rows have the same time and date value because there are many items have the same time and date to go into the exhibition or go to the new locations, but it was not enough when I had to update for 2 attributes (timestamp when it is in and timestamp when it is out). Every time I done an update, I had to go to another look for the new date that haven’t got combine yet and sometimes, I miscounted the row and had to do it over again. However, thanks to this, I also found out some error in my old data and then I can delete them from the database.

After I have done all the combining, I finally can the insert the data from that old table of mine to the new one. There is not so much difference between my old and the new table now when the only thing I had to do is adding my museum name for each of the row.

Because this is the most massive table that I have, I have some queries to check a certain of works to see whether theirs date is reasonable and possible. I got some error occur and I had to fix some part of the data.

**6)** **Works\_Exhibitions TABLE:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Exhibition\_works** | | ew-itemalpha\_ex FK-wk  ew-itemnumber\_ex FK-wk  ew-ex\_id FK-ex | |  | | *New:*   |  | | --- | | **ExhibitionsWorks** | | ew-exName FK-ex  ew-wkIDAlpha FK-wk  ew-wkIDNumeric FK-wk  ew-exStartDate FK-ex  ew-exMuseumName FK-ex | | ew-endDate | |

My old table does not contain the information of date period of the exhibitions as well as the name of the exhibition, therefore, I had to change my old table (just like in works\_location) to have more attribute that contain those data. So, I add the exhibition name column and the start date and end date column, then I get the update from the exhibitions table. Because they have the same exhibition\_id, it is easy to update that and this process won’t take long like the previous one.

After I got all the information I need in the old tables, I will simply do the insert from my tables to the new one, and of course, I will also insert my museum name.

I only need the query showing the whole table and compare it with the old one to find some problem it may have.

**7)** **Works\_Value TABLE:**

This table is the new table and only contain the value of my item. And because my works value had not been changed, it only need one record for each item. So, I just need to get the value, the work identification and their acquisition date to insert into the new table.

However, I still need to do some end date update in the end because some of my items have been sold to another institution, therefore, the end data of this item value is the date it be sold because from now on, I won’t have any connection with it (except if they sold it back to me).

**8)** **Works\_State TABLE:**

This table will show the current state of the items in a period of time. But I haven’t change anything to my works state so like the works value table, I will just update all the state to ‘in good condition’ while inserting the identification and theirs acquisition date.

However, I still need to do some end date update in the end because some of my items have been sold to another institution, therefore, the end data of this item value is the date it be sold because from now on, I won’t have any connection with it (except if they sold it back to me).

**9)** **Works\_Media TABLE:**

My media table look exactly like the team database table. The only difference is that now I had to add my institution name and nothing else. So, it is an easy to check situation when I only need a query to show the whole table.

**10)** **Exhibitions\_Locations TABLE:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Exhibition\_location** | | el-Ex\_id FK-ex  el-current\_location FK-lc | | el-no\_of\_items | | *New:*   |  | | --- | | **ExhibitionsLocations** | | el-lcName FK-lc  el-exName FK-ex  el-exStartDate FK-ex  el-lcexMuseumName FK-lc, ex | | el-lctelStartDate  el-lctelEndDate  el-exEndDate | |

Like the works exhibitions table, my old exhibition location table does not contain the information about the exhibition name and its period of time. The period of time for the traveling exhibition at each location also does not concern me much because in my old exhibitions table, I have keep record of all of them.

So, the first thing I do is to add more columns into my old table and then update them to get the new information from the exhibitions table. This is the exact thing that I do at the works\_exhibitions table, which have the same format as the old exhibition location table.

After I got all information in the exhibition location table, I just need to use insert to get the data to the new table and I will also add my museum name in this insert.

In the end the separate date for the traveling exhibition, which can be update manually.

I only need the query showing the whole table and compare it with the old one to find some problem it may have.

**11)** **Locations\_doors TABLE:**

My doors table look exactly like the team database table. The only difference is that now I had to add my institution name and nothing else. So, it is an easy to check situation when I only need a query to show the whole table.

**12)** **Works\_Ownership TABLE:**

In my old works table, it has the claim attribute which can show that whether the items is OWNED, BORROWED, POTENTIALLY BORROWED or LOAN. Therefore, based on this information, I can update my ownership table.

First the owned and loan means that my works is already mine, therefore, I will be its owner. So, by using the acquisition date as start date of the ownership, I have upload the item as mine.

Then there is the owner of the BORROWED work. The name of this institutions is in my donors table along with their identification of the item they lend us, therefore I can easily track their name down and insert into the table. However, because I do not know the date their ownership start, so I just choose randomly the date 2016-6-1.

I only need the query showing the whole table and compare it with the old one to find some problem it may have.

**13)** **Transactions TABLE:**

This is the new table that will record my museum transaction with other institutions including my team database. However, I have lost track of all my database at this point, so I will only update the transaction when I am selling the item to other institutions.

**Part2**: adding the data to the group database:

* After I got all the data from the new tables, I copy those data and paste them on the text file.
* From here, I have to change the file to the style so that it can be copy into the database by sql.
* First I replace all the character '|' to the tab, which is required for a file to be update on the database.
* Next, I delete or the unnecessary space between the column, so that when the file is upload, the primary key will be the same and will not cause error about unrecognizable primary key.
* Next I have to add the character '\N' that can show it is a NULL data
* After the first upload, my works file got problem when some of the works have strange characters and therefore, cannot be load. I have to find and delete those characters before my file can have good run. There is a total 9 works that have that error.
* Next, I have the error when I am loading the work-ownership file because my table column sequence is different from the rest of my group. That happens because the team data file I use to create the table is older than my teammate, and therefore, it have different presentation of that table. So, i have to change the position of the column before I can add it to the team database.
* Finally, after I done, I query all of the table to check that my database is in there.

**Wei Zhang’s report for E**

Descriptions of what changes did for my own database  
5.1. For the **Works** table, deleted the attribute media, because we created the media table to put media information in there.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Old table:*   |  | | --- | | **museum** | | muletter  munumber | | muname  mudescription  mutype  musubtype  Muauthor  mumedia  mudate\_complete  mudate\_accquire  muinsurance\_value  muownership  mucharacteristic | | *Current group table:*   |  | | --- | | ns-**Works** | | ns-wk-IDAlpha  ns-wk-IDNumeric  ns-wk-IDinstitution | | ns-wk-Creator  ns-wk-completionDate  ns-wk-acquisitionDate  ns-wk-workDescription  ns-Wk-ownershipStatus  ns-Wk-theme  ns-Wk-subject  ns-Wk-culture  ns-Wk-colour  ns-wk-carType  ns-wk-transmission | |

I added a ‘institution name’ attribute to my new works table. The reason for this is to making a clearly identifier which identifies the belonging of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

Also I added new attribute ns-Wk-theme,ns-Wk-subject, ns-Wk-culture, ns-Wk-colour, ns-wk-carType, ns-wk-transmission according to the new designed classification systems to all works.

However here comes a problem when I trying to merge my old database to the new one.

One attributes from previous old table and the attribute from new designed table, both of them represent the same functionality, but their domains are different, which can not converted by psql automatically.

For example I only use year as completion date in my old database, but we decide to use year-date as completion date in the new database. Hence I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

To test the media attribute changes in the table query this table to select the ns-wk-IDAlpha, s-wk-IDNumeric, ns-wk-acquisitionDate, and media from the old Works table and ns-wk-IDAlpha, s-wk-IDNumeric, ns-wk-acquisitionDate from the new media table. I will check to make sure these values match up. To test the wk-question and wk-answer attributes I will select the wkIdAlpha, wkIdNum, and wkchatacteristic from the old Works table and wk-IDAlpha, wk-IDNumeric, and wk-answer from the new Works table. I want to make sure these values match. I will also select every wk-question from the new Works table to check if it is set to “Is the work abstract?”.

5.2. For the **Locations** table, added additional attribute called lc-telStartDate as the location start to use date,add additional attribute called lc-telStartDate as the location end to use date  
And also create the domain for ‘lc-museumname as ‘museumname’ and make the domain type as varchar(50)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Locations** | | lc-name | | lc-dimension  lc-min\_capacity  lc-max\_capacity | | *New:*   |  | | --- | | ns-**Locations** | | ns-lc-name  ns-inInstitutionName | | ns-lc-length  ns-lc-height  ns-lc-width  ns-lc-minNumWorks  ns-lc-maxNumWorks  ns-ls-availabilityDate  ns-lc-sponsor  ns-lc-security  ns-lc-insurance  ns-lc-streetAddress  ns-lc-city  ns-lc-country  ns-lc-postalCode | |

Because in this way we can distinguish the same location name with different museum name. Add additional attribute called ‘lc-availabledate’ to indicate when and where the location will be available to use in future. Also create the domain for attribute ‘lc-availableDate’ as ‘availableDate’ as date.

Also I added new attribute ns-lc-length, ns-lc-height, ns-lc-width according to the new designed dimensions to the locations.

Also I added new attribute ns-lc-sponsor, ns-lc-security, ns-lc-insurance, ns-lc-streetAddress, ns-lc-city, ns-lc-country, ns-lc-postalCode. Because we want to merge the travelling- location information in the locations table.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

To test the changes to the lc-availabledate I will query the old location table selecting the length and width, and the new Locations table selecting the dimension to confirm the date has been mapped accordingly. To test the lc-telStartDate I will query all of the temporary locations in my traveling exhibition and confirm that their start dates and end dates add up to the total duration of the exhibition.  
  
3.For the **Exhibitions** table, add attribute ‘ex-isTravelling’ to indicate whether the exhibition is travelling exhibition or not.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **exhibitions** | | ex-exhibitionName  ex-exhibitionStartDate | | ex-ex\_descripton  ex-exhibitonEndDate | | *New:*   |  | | --- | | ns-**Exhibitions** | | ns-ex-exhibitionName  ns-ex-exhibitionStartDate  ns-ex-inInstitutionName | | ns-ex-istraveling  ns-ex-exhibitionDescription  ns-ex-exhibitonEndDate | |

Also create domain isTraveling as Boolean.  
Also drop table exhgala. Because the data in exhgala can get from exhibitionlocation. Exhgala table is redundant.

I added a ‘institution name’ attribute to my new exhibitions table. The reason for this is to making a clearly identifier which identifies the belonging of each exhibition. Hence in this way could avoid the conflicts with exhibitions hold by other museums which could have the same exhibition name and start date.

However here comes a problem when I trying to merge my old database to the new one.

One attributes from previous old table and the attribute from new designed table, both of them represent the same functionality, but their domains are different, which can not converted by psql automatically.

For example I only use varchar(100) as exhibition description in my old database, but we decide to use varchar(200) as exhibition description in the new database. Hence I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~ To test if the changes to exhibitions table worked, just query this table by selecting the added names of attributes and the deleted names of attribute. And display the table to see if the changes works. And also display the entire database to see if the exhgala table dropped.

To test the changes to the isTravelling I will query the old travelling exhibition table selecting the exhibitionname and the new exhibition table selecting the ex-exhibiitionname and ex-istravelling to confirm the exhibition name has been mapped accordingly.

5.4.For the **WorksLocations** table, add additional attribute called 'wk-lcmuseumNme' as primary key and it’s also a foreign key.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **workslocation** | | wl-museumetter FK-wk wl-museumnumber FK-wk wl-current\_location  wl-date\_in | | wl-date\_out  wl-time\_in  wl-time\_out | | *New:*   |  | | --- | | ns-**WorksLocations** | | ns-wl-lcName FK-lc  ns-wl-wkIDAlpha FK-wk  ns-wl-wkIDNumeric FK-wk  ns-wklcinInstitutionName FK-lc,wk  ns-wk-workLocationStartDateTime | | ns-wk- workLocationEndDateTime | |

Because other member's museums can have the same locationname.   
Also change the domain workLocationStartDate to workLocationStartDateTime as timestamp;  
Also change the domain workLocationEndDate to workLocationEndDateTime as timestamp;

I added a ‘institution name’ attribute to my new works-Locations table. The reason for this is to making a clearly identifier which identifies the locations of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

Test:If there is no error messages occurred after the each insert or update, that means there is no general mistake. Then I compared the number of records in old ‘**workslocation**’ table with the number of records in new ‘ns-works-locations’ table, if those two numbers of records are same, that means the transfer is successful.

Also, I picked 5 records from new ‘ns-works-locations’ table randomly, and compare them with corresponding records in old old ‘works-locations’ table, to see if the value in each attribute was transferred correctly.

5.5

For the **ExhibitionsWorks** table, delete the redundant attribute 'moveintime', because that information can get from the workslocations table.

~ To test if the changes to worksexhibitions table worked, just query this table by selecting the deleted name of attributes. And display the table to see if the changes works.  
  
5.6. For the **ExhibitionsLocations** table, add the additional attribute 'el-lcexmuseumname' as a primary key.  
Because other member's museum could have the same location name.   
Also add the attribute ‘el-lctelEndDate’ to store the end date of travelling exhibition.  
Create the domain tempExhibitionLocationStartDate as Date.  
Also add the attribute ‘el-lcexMuseumName’ to store the start date of travelling exhibition.   
Create the domain tempExhibitionLocationEndDate as Date.  
  
Because the exhibitionsLocations is a temporal table we need to keep history records of traveling exhibitions.

Test:If there is no error messages occurred after the each insert or update, that means there is no general mistake. Then I compared the number of records in old ‘exhibitions-locations’ table with the number of records in new ‘ns-exhibitions-locations’ table, if those two numbers of records are same, that means the transfer is successful.

Also, I picked 5 records from new ‘ns-exhibitions-locations’ table randomly, and compare them with corresponding records in old old ‘exhibitions-locations’ table, to see if the value in each attribute was transferred correctly.

5.7. For the **Doors** table. Add the attribute dr-lcMuseumName as a primary key,   
because that is much easier to check which door connected to which doors in others museum.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Current:*   |  | | --- | | **Doors** | | dr-door\_from FK-lc  dr-door\_to FK-lc | |  | | *New:*   |  | | --- | | ns-Locations-**Doors** | | ns-ld-lcinInstitutionName FK-lc  ns-ld-lcName1 FK-lc  ns-ld-lcName2 FK-lc | |  | |

added a ‘institution name’ attribute to my new works-Locations table. The reason for this is to making a clearly identifier which identifies the locations of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same work ID.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically. Also, I picked 5 records from new ‘ns-Locations-**Doors**’ table randomly, and compare them with corresponding records in old old ‘**Doors**’ table, to see if the value in each attribute was transferred correctly.

~ To test if the changes to doors table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works.

5.8 Also add a **Media** table in my database. Because other museum works have more than one materials for some works.

*New:*

|  |
| --- |
| ns-Works-Media |
| ns-wm-wkIDAlpha FK-wk  ns-wm-wkIDNumeric FK-wk  ns-wm-wkIDInstitution FK-wk  ns-wm-material |
|  |

Also add the attributesns-wm-wkIDAlpha, ns-wm-wkIDNumeric, ns-wm-wkIDInstitution, ns-wm-material into the materials table. And all of them are the primary keys.

added a ‘institution name’ attribute to my new ns-media table. The reason for this is to making a clearly identifier which identifies the medias of each work. Hence in this way could avoid the conflicts with works from other museums which could have the same medias.

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

Test: I compared the number of records in old ‘museum’ table with the number of records in new ‘ns-Works-Media’ table, if those two numbers of records are same, that means the transfer is successful. Also, I picked 5 records from new ‘ns\_item\_materials’ table randomly, and compare them with corresponding records in old old ‘museum’ table, to see if the value in each attribute was transferred correctly.

5.9. Create a new table called **WorksState** that store the information about the works state. Add the attributes ‘ws-wkIDNumeric’, ‘ws-wkIDAlpha’, ‘ws-workStateStartDate’ as primary keys. Also add the ‘ws-state’, ‘ws-workStateEndDate’ attributes to store the state information. Also the ws-state could be stolen, damaged, restoration, in good condition.

|  |
| --- |
| **WorksState** |
| ws-wkIDNumeric FK-wk  ws-wkIDAlpha FK-wk  ws-workStateStartDate |
| ws-state  ws-workStateEndDate |

Create domain for attribute ‘ws-state’ called state as varchar(30); And create constrain for the state could be only one of (stolen, damages, restoration, in good condition)

I will cast in my sql every time when an attribute need to be converted into the new table but could not converted by psql automatically.

~ To test if the changes to worksState table worked, just query this table by selecting the added name of attributes. And display the table to see if the changes works

12. Create a new table called **WorksInsuranceValue** that store information about the works insurance information. Make the ‘wiv-wkIDNumeric’, ‘wiv-wkIDAlpha’, ‘wiv-workinsuranceValueStartDate’ as primary keys. Also add attribute ‘wiv-insuranceValue’, ‘wiv-workInsuranceVaueEndDate’ to record the information about the insurance of each works. Because the insurance value of each works could change.  
Also create domain for attribute’ wiv-insuranceValue’ called workInsuranceValueStartDate as date;  
Also create domain for attribute‘wiv-insuranceValue’ called workInsuranceValueEndDate as date;

|  |
| --- |
| **Value** |
| wiv-wkIDNumeric FK-wk  wiv-wkIDAlpha FK-wk  wiv-workInsuranceValueStartDate |
| wiv-insuranceValue  wiv- workInsuranceValueEndDate |

~ To test if the changes to worksInsuranceValue table worked, query this table by selecting the added name of attributes. And display the table to see if the changes works  
  
By using Dbvisualiser to display the E-R diagram of my new database and compare it with our planed E-R diagram. If they have same structure, then my database is correct.

**Team Database Creation:**

**Part F:**

The trigger used:

The trigger will be divided for its function to the whole table:

- The trigger that is only used for update the new value of one table:

·  **CheckValue on works\_Value table:**

When the value of the item change, it means that the end date of the previous value will also going to end. This trigger will help you to update the final date of the previous value by giving it the start date of the new value as long as the end date of the previous value is NULL.

· **ChangState on works\_ State table:**

When the state of the item change, it means that the end date of the previous state will also going to end. This trigger will help you to update the final date of the previous state by giving it the start date of the new state, as long as the end date of the previous state is NULL.

·  **UpdateWorkLocationEndDate on works\_location table:**

When the location of the item change, it means that the end date of the previous location will also going to end. This trigger will help you to update the final date of the previous state by giving it the start date of the new location, as long as the end date of the previous location is NULL.

- The triggers that will be used to plan for a whole new exhibition, in other work, they are related to each other:

· **changeWorkLocation1 on works\_exhibitions table:**

This trigger is not necessary needed, however, it is still efficient and can reduce the works of the trigger after it. In other word, let say you are planning for a new exhibition and you know all the works that you can use in that exhibitions. The first thing you do is to insert the information of this new exhibition to the exhibitions table. However, at this step, you must not declare the isTraveling attribute to be true or false because it is an essential part for the trigger after. Next, you will add the identification of your items with the name of the exhibition to the works\_exhibitions table, then this is the time that the trigger does its job. It adds the work identification along with other necessary information to the works\_locations table. However, the location for this row will only be a temporary one and it will be the storage of each individual museum. Right after we done with the next triggers, this temporary location will be updated and change into the location that the exhibition is held.

·  **Ex\_schedule on exhibitions table:**

Now, the next step to set up an exhibition is to update the value isTraveling of the exhibition. After we done the update to tell whether the new exhibition is a traveling exhibition or not, this trigger will be activated. Sadly, this trigger will only perform the setting location for the normal exhibition, because the temporary locations for traveling exhibition do not have min and max capacity or available date. Moreover, the location for the traveling exhibition can be multiple, therefore, it should be insert manually to the exhibitions\_locations.

Back to the trigger, it will do the job to find the most suitable location for your exhibition based on the available date and the location capacity. After it finds the best location to set the exhibition, it will add all the information to the exhibitions locations table and activate the next trigger.

·  **updateWorkLocation on exhibitions locations table:**

After we insert a new row to the exhibition location table by the Ex\_schedule trigger (for the normal exhibition) or by hand (for the traveling exhibition), this will activate another trigger that will change the data to the works\_location table.

This trigger can recognize the whether the exhibition got insert in the exhibitions\_locations table is a traveling or not by checking the traveling date. If is null, it is a normal exhibition, but if it is not, it is a traveling one. Base on this information, the trigger will have different action to store the information because these 2 kinds exhibitions are different. Fow example, the date the item go to the location will be 2 days before the exhibitions held, but for traveling exhibition, the date it come to the temporary location is the traveling date, not the exhibition start date. So, for this trigger, it helps us to sort out the kinds of exhibitions.

- The trigger that bring the new works to the storage:

· **changeWorkLocation2 on works table:**

When the new item is added to the museum, it will always first go to the storage, however, this is not the case with the POTENTIALLY BORROWED item when they are not technically in our museum yet. However, this trigger can recognize whether this item is just BORROWED or not by looking at its acquisition date. If this date is NULL, then it means it is just POTENTIALLY BORROWED and therefore, no need to add into the works\_location table. The opposite will be done for the other. So, the item will only go into the museum storage if only if we update its acquisition date.

**Part G:**

Our current database structure implements transactions. Below is a listing of these transactions, what they are used for, and what tables are affected by them.

|  |  |  |
| --- | --- | --- |
| Name of Transaction | Description of Transaction | Tables affected by Transaction |
|  |  |  |
| InsertExhibition | Adds a new exhibition to the database for a specified institution. | Exhibitions |
| InsertExhibitionWork | Adds a new specified work to a specified exhibition | Works\_Exhibitions |
| InsertInstitution | Adds a new specified institution to the database | Institutions |
| InsertLocation | Adds a new specified location to the database | Locations |
| InsertExhibitionLocation | Adds a new specified exhibition to a specified location | exhibitions\_locations |
| InsertLocationsDoors | Adds a door in between two specified locations | locations\_doors |
| InsertWorkNewLocation | Adds a new location to the works\_locations table with the specified work and current date | works\_locations |
| InsertWorkNewState | Updates the state of a work to the specified state. Sets the previously assigned end date to the current date | works\_state |
| InsertWorkNewValue | Updates the value of a work to the specified value. Sets the previously assigned end date to the current date | works\_value |
| InsertWorksMedia | Adds another media attribute to a specified work | works\_media |
| LendOutWork | Lends out a work to a specified museum | transactions |
| LendInWork | Institution gets back a work that it had previously lent out to the specified museum | transactions |
|  | New Multi-Table Transactions: |  |
| AddPotentiallyBorrowedWork | Adds a work to the works table that could be potentially borrowed from the specified museum. | Works,  Works\_Ownership,  Works\_State,  Works\_Value |
| Buy Work | Adds a work to the museums database that it purchased from a specified institution. | Transactions,  Works,  works\_Ownership,  Works\_state,  works\_value |
| Sell Work | Sells a specified work to a specified institution | Transactions,  works\_locations |
| BorrowInWork | Add a work into our database that is borrowed from another specified institution | Transactions,  Works,  works\_Ownership,  Works\_state,  works\_value |
| BorrowOutWork | Give back a work to an institution who we borrowed a work from | Transactions,  works\_locations |

**Part H:**

Our current database structure implements 21 queries. Below is a listing of these queries, what they are used for, and what tables are affected by them.

|  |  |  |
| --- | --- | --- |
| Name of Query | Description of Query | Tables affected by Query |
| CurrentStorage | The current storage query produces for the head of the museum a listing of all works (including their identifier, name, and insurance value) that are currently in storage. | Works  Works\_value  Works\_locations |
| ExhibitionWorksCount | The exhibition works count query will produce for the public a listing of all exhibitions (including name, description, location, and number of works). | Works\_exhibitions  Exhibitions  Exhibitions\_locations |
| ExhibitWorks | The exhibit works query will produce for the public a listing of all publicly available data on all works in each of the exhibitions sorted by exhibition and by the name of the work. | Works  Works\_Exhibitions  Works\_Media |
| SpacesInGalleries | The spaces in galleries query will produce for the curator a listing of the amount of additional works that could be added to each exhibit based on the unused capacity of the galleries that they are currently in. This listing should just have the amount of additional works that could be added and the name of the exhibit. | Locations  Works\_exhibitions  Exhibitions  Exhibitions\_locations |
| WorksAvailability | The works availability query will produce for a curator a listing of works sorted by when they are available for use in a new exhibition and by classification and by name of the work | Works  Works\_locations |
| ExhibitLocations | The exhibit locations query will produce a listing of all locations of exhibitions’ locations | Exhibitions\_locations |
| AllDoors | The all doors query will produce a listing of all doors. | Locations\_doors |
| ExhibitionDates | The exhibition dates query will produce a listing of all exhibitions with start and end dates | Exhibitions |
| ExhibitWorksSmall | The exhibit works small query will produce a limited number of attributes listing of all works in all exhibitions | We\_exname |
| WorksLocations | The works locations query will produce a listing of all work locations | Works\_locations |
| CurrentWorksOwnership | The current works ownership query will produce a listing of all current work ownership status | Works\_ownership |
| WorksOwnership | The works ownership query will produce a listing of all work ownership status | Works\_ownership |
| GalleryAvailability | The gallery availability query will produce a list of when each gallery from all museums will be available | Locations |
| WorkTypes | The work types query will produce a query of all types that the work belongs to | Works |
| ExhibitWorksCountWithRoomMax | The exhibit works with room max query will produce a report of all current and future exhibitions including the name of the exhibiton, the dates it is happening (starting and ending dates), the maximum recommended capacity of the  locations it is using, and the current number of works planned for it | Works\_exhibitions  Exhibitions  Exhibitions\_locations  Locations |
| LocationsBetweenDates | The locations between dates query will produce a query that lists the different locations that a given work was/is/will be in between two dates | Works\_locations |
| ExhibitionWorksBetweenDates | The exhibition works between dates will produce a query that lists all the works found in an exhibition between two dates | Works\_exhibitions |
| LocationExhibitionsBetweenDates | The location exhibitions between dates will produce a query that lists all the exhibitions that make use of a location between two dates | Exhibitions\_locations |
| TransactionTypes | The TransactionTypes query will produce a listing of all transactions that are of a specified type. | transactions |
|  | **New Multi-Table Queries:** |  |
| CurrentBorrowedWorksLocations | The current borrowed works location will determine the current location of all borrowed works in our museums | Transactions  Works\_locations |
| CurrentExhibitionSubjectCount | The current exhibition subject count will determine how many works of a specified subject are in current exhibitions | Works  Works\_exhibitions |
| WorksTravelingToSK | The works traveling to SK query will determine what works will be in saskatchewan and at what dates | Work  Works\_locations  Locations |