**Final Project Paper**By: Niki Jahanara

**Introduction**

The purpose of this paper to is to outline the process of taking the un-normalized Federal Stimulus dataset from 2009 [(Link)](https://data.cityofnewyork.us/Business/Federal-Stimulus-Data/ivix-m77e/data) and transforming it into a relational database. The task of transforming the dataset into a relational database is accomplished by first understanding the purpose of the data, how the data was gathered, and how the data was used. The next step in this process is to find, understand, and implement the changes needed to normalize the data through the use of an Entity Relationship Diagram (ERD) and the appropriate CREATE TABLE statements. The final step in this process is to write queries that answer common, non-trivial questions that someone may ask about the data in the relational database in order to ensure that the data was correctly normalized. The process one would take to transform the un-normalized Federal Stimulus data from 2009 into a relational database is described in detail in the following sections.

**Section 1.) The Business Purpose of the Data**

**A Brief History of the American Recovery and Reinvestment Act (ARRA)**

The Federal Stimulus data from 2009 documented in the dataset is based on the usage of stimulus or recovery funds that were provided by the American Recovery and Reinvestment Act (ARRA) of 2009. According to the U.S Department of Education’s official website [(Link)](https://www2.ed.gov/policy/gen/leg/recovery/factsheet/overview.html), ARRA was a stimulus package that was signed into law on February 17th, 2009 by President Obama with the purpose of stimulating job creation during a time in U.S history known as “The Great Recession” by cutting taxes and investing hundreds of billions of dollars into energy, health care, infrastructure, and education over the following two years after ARRA was signed into law. According to the official website of the U.S Treasury [(Link),](https://www.treasury.gov/initiatives/recovery/Pages/recovery-act.aspx) the act was also an effort to save millions of jobs and to address neglected challenges within the U.S by putting down a payment for those challenges that could be fixed with funding.

**The Business Purpose of the Federal Stimulus Data**

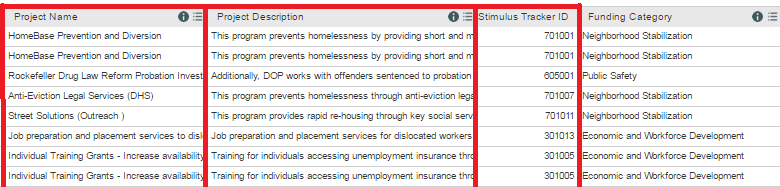
Based upon the intent of the ARRA, it can be determined that the business purpose of the Federal Stimulus data from 2009 was to keep track of every project that either requested or received federal stimulus funding, the individual amounts of funding each project received, the total amount of each project’s funding, the source of each project’s funding, the start and end dates of when the project was expected to use the funding it received as well as the actual start date should a project’s use of funds be delayed, each project’s funding contract details, each project’s vendor, and the payment details of every project that received funding after the ARRA was passed.

It can also be determined that the business purpose of tracking the details about every project that either received or requested funding from the passing of the ARRA was to ensure that the funding ARRA provided to each project was being used for the purpose that the act intended: to save jobs, to create jobs, and to reform critical areas of life in the U.S during “The Great Recession”, such as energy, education, health care, and infrastructure. The critical areas of life that the ARRA intended to reform can be seen from the diverse purposes of the projects listed in the Federal Stimulus dataset from 2009, and it is the reason why the Federal Stimulus dataset lists projects with purposes ranging from providing meals to the elderly to job preparation, to education reform, and to placement services for dislocated works and high need individuals.

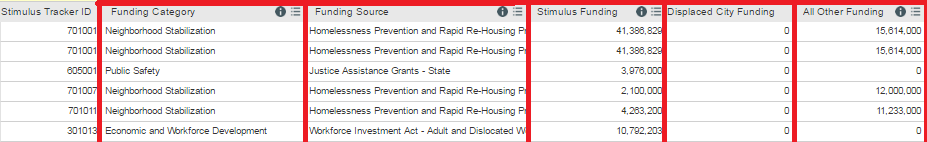
**Section 2.) A Description of the Un-normalized Data’s Form**

**A General Description of the Un-Normalized Dataset**

The un-normalized dataset is primarily organized in blocks of columns that relate to a specific portion of a project. The blocks of columns that the un-normalized data are organized into are the project’s name and description details, the project’s funding details, the project’s contract details, the project’s vendor, and the project’s payment details.

  
***Pictured Above:*** *A block of columns that relate to the* ***project*** *portion of the data.*

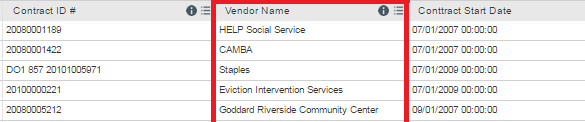
For example, the first three columns of the dataset (**Project Name**, **Project Description**, and **Stimulus Tracker ID**), shown in the image above, relate directly to the project itself and provide a way to uniquely identify a project by its stimulus tracker ID.

  
 ***Pictured Above:*** *A block of columns that relate to the* ***funding*** *portion of the data.*

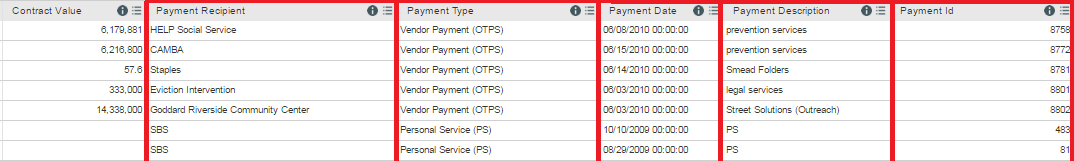
The five attributes (**Funding Category**, **Funding Source**, **Stimulus Funding**, **Displaced City Funding**, **All Other Funding**) shown in the image above that follow the project portion of the data however, relate more so to the project’s funding details. The pattern of the Federal Stimulus data being organized into blocks of columns that relate to the project’s details, the project’s funding details, the project’s contract details, the project’s vendor, and the project’s payment detail continue for the remaining columns in the Federal Stimulus dataset.



***Pictured Above:*** *A block of columns that relate to the* ***contract*** *portion of the data.*



***Pictured Above:*** *A block of columns that relate to the* ***vendor*** *portion of the data.*

******

***Pictured Above:*** *A block of columns that relate to the* ***payment*** *portion of the data.*

The fact that columns are grouped according to their relation to a specific portion of each project is an important aspect of this particular dataset because it allows the dataset to easily be normalized into entities with attributes corresponding to the columns in the dataset that relate to a specific portion of each project.

**An Important Note Regarding the Dataset Columns**

Although many of the columns relating to a specific portion of the project are grouped together, some columns are not. The ProjectStatus column, for example, is listed after the columns related to the project’s funding instead of being grouped with the ProjectName and ProjectDescription columns. The fact that dataset’s columns are not always correctly grouped together is important to consider when attempting to normalize the data into entities with the correct columns that relate to that entity.

**The Federal Stimulus Dataset’s Column/Attribute Descriptions**

To better understand which portion of the project each column in the dataset belongs to, the following list gives a general description of each of the 37 columns in the Federal Stimulus dataset and explains which portion of the project each column belongs to.

1. Project Name: The name of the project that requested/received funding.  
**Relates To:** The project itself, because it is the name of the project.

2. Project Description: The description of the project that requested/received funding.  
**Relates To:** The project itself, because it is the description of the project.

3. Stimulus Tracker ID: The unique ID of each project that requested/received funding.  
**Relates To:** The project itself, because it is the unique ID of the project.

4. Funding Category:The specific funding category of the project.  
**Relates To:** The project’s funding, because it is the category of the funding requested.

5. Funding Source:The organization that provided funding for the project.  
**Relates To:** The project’s funding, because it states the organization that provided funding for the project.

6. Stimulus Funding: The amount of stimulus funding that the project received in total.  
**Relates To:** The project’s funding, because it is the total amount of funds received by the project.

7. Displaced City Funding:The amount of funds displaced by the city.  
**Relates To:** The project’s funding, because it is the amount of funds displaced by the city.

8. All Other Funding: The amount of funding received by the project from other sources.  
**Relates To:** The project’s funding, because it is the amount of funds received from other sources by the project.

9. Award Lead City Agency:The agency that granted the project the funding award. **Relates To:** The project’s funding, because it is the agency that granted the project the funding award.

10.Project Lead City Agency: The agency of the project’s city that is representing the project.  
**Relates To:** The project itself, because it is the agency representing the project.

11. Project Status: The completion status of the project.  
**Relates To:** The project itself, because it is the completion status of the project.

12. Percent of Funds Spent: The percentage of the funds that have been spent by a project.  
**Relates To:** The project’s funding, because it a percentage of project funds that have been spent.

13. Date Funds Awarded By Fed/State:The date that funds were awarded to a project. **Relates To:** The project’s funding, because it is the date that the project received funding.

14. Date Funds Announced By NYC:The date that a project’s funds were announced by NYC. **Relates To:** The project’s funding, because it is the date that the project’s funds were announced by NYC.

15. Estimated Start Date:The estimated date that project’s operations will begin.  
**Relates To:** The project itself, because it is the estimated date that the project’s operations will start.

16. Actual Start Date:The date that the project’s operations actually began. **Relates To:** The project itself, because it is the actual date that the project began operations.

17. Actual Completion Date:The date that the project’s goal was completed.  
**Relates To:** The project itself, because it is the date that a project’s goal was completed.

18. Interim Spending Deadline:The interim spending deadline of a project.  
**Relates To:** The project’s funding, because it is the interim spending deadline of a project.

19. Percent of Funds to Be Spent by Interim Spending Deadline: The percent of funds that a project must spend by its interim spending deadline. **Relates To:** The project’s funding, because it is the percent of funds that a project must spend by its interim spending deadline.

20. Final Spending Deadline: The final spending deadline for a project.  
**Relates To:** The project’s funding, because it is the final spending deadline for a project’s funds.

21. Contract Name:The name of the funding contract received by a project.  
**Relates To:** The project’s contract, because it is the name of the project’s funding contract.

22. Contract Method:The method of the funding contract received by a project.  
**Relates To:** The project’s contract, because it is the method of the project’s funding contract.

23. Contract Status: The status of the funding contract received by a project.  
**Relates To:** The project’s contract, because it is the status of the project’s funding contract.

24. Contract ID:The ID of the funding contract received by a project.  
**Relates To:** The project’s contract, because it is the ID of the contract received by a project.

25. Vendor Name: The name of the vendor that is on the project’s contract.  
**Relates To:** The project’s vendor and the project’s contract, because of the relationship between projects and vendors, as well as projects and contracts, which is described in the ERD section.

26. Contract Start Date:The start date of a project’s funding contract.  
**Relates To:** The project’s contract, because it is the start date of the project’s funding contract.

27. Contract End Date:The end date of a project’s funding contract.  
**Relates To:** The project’s contract, because it is the end date of the project’s funding contract.

28. New Or Existing Contract: This states whether the contract is new or has already been made.  
**Relates To:** The project’s contract, because it states whether the project’s contract is new or not.

29. Revised Contract Start Date: The revised date for when the project’s contract will start.  
**Relates To:** The project’s contract, because it is the revised date for the contract’s start date.

30. Revised Contract End Date: The revised date for when the project’s contract will end.  
**Relates To:** The project’s contract, because it is the revised date for the contract’s end date.

31. Contract Value: The monetary value of the project’s contract.  
**Relates To:** The project’s contract, because it is the monetary value of the project’s contract.

32. Payment Recipient: The recipient of a payment that has been made for a project.  
**Relates To:** The project’s payment, because it is the recipient of the payment for a project.

33. Payment Type: The type of payment that has been made for a project.  
**Relates To:** The project’s payment, because it is the type of payment that a project received.

34. Payment Date: The date that a payment was made to a project.  
**Relates To:** The project’s payment, because it is the date that a payment was made to a project.

35. Payment Description: The description of the payment that was made to a project.  
**Relates To:** The project’s payment, because it is the description of the payment for a project.

36. Payment ID: The ID of a payment that was made to a project.  
**Relates To:** The project’s payment, because it is the ID of the payment that a project received.

37. Payment Value: The value of the payment that was made to a project.  
**Relates To:** The project’s payment, because it is the value of the project’s payment.

**Section 3.) Normalizing the Data in BCNF & the Changes Needed to Normalize the Data**

**Boyce-Codd Normal Form (BCNF)**

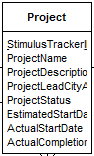
There are several forms in which a dataset can be normalized. Among the several forms is Boyce-Codd Normal Form, which is how this paper intends to normalize the Federal Stimulus dataset. In order to normalize the tables for the Federal Stimulus Data in BCNF, “All attributes should depend on the key, whole key and nothing but the key (so help me Codd).” This means that the 37 columns in the dataset must be decomposed into tables with attributes representing the columns and where each table’s attributes depend entirely on the decomposed table’s primary key.

**The Entities/Tables (Prior to Normalization in BCNF)**

The previous sections of this paper outlined five entities that can be decomposed into tables that exist within the Federal Stimulus dataset: **Project**, **Funding**, **Payment**, **Contract**, and **Vendor**. The five entities that exist within the Federal Stimulus dataset have the following attributes prior to normalization in BCNF:

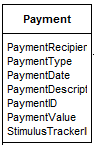
**\* Note:** The number of each attribute corresponds to the list of the columns in the previous section.

**Decomposed Table #1:** Project

**Attributes:**1. ProjectName  
2. ProjectDescription  
3. StimulusTrackerID   
10. ProjectLeadCityAgency  
11. ProjectStatus  
15. EstimatedStartDate   
16. ActualStartDate   
17. ActualCompletionDate

The Project table was made by breaking off the columns from the Federal Stimulus dataset that related to each project’s own details. It does not need a foreign key to any attributes in another table, however, due to its relationship with the other entities/tables. The relationship between the entities is explored in detail in the next section: The Entity Relationship Diagram (ERD).

**Decomposed Table #2:** Payment

**Attributes:**32. PaymentRecipient   
33. PaymentType   
34. PaymentDate   
35. PaymentDescription   
36. PaymentID  
37. PaymentValue  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**

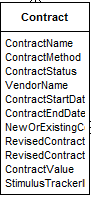
The Payment table was made by breaking off the columns from the Federal Stimulus dataset that related to the project’s payment. After breaking the Payment attributes off of the Federal Stimulus dataset, a foreign key to the StimulusTrackerID of the Project table is added due to the one-to-many relationship between the Payment and Project entities that is explained in depth in The Entity Relationship Diagram (ERD) section.

**Decomposed Table #3:** Vendor

**Attributes:**25. VendorName

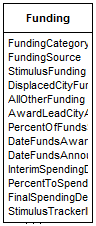
The Vendor table was made by breaking off one column, Vendor Name, from the Federal Stimulus dataset because it is the only column that relates to each project’s vendor.

**Decomposed Table #4:** Contract

**Attributes:**21. ContractName   
22. ContractMethod   
23. ContractStatus   
24. ContractID   
25. VendorName **(Foreign Key to VendorName in Vendor)**   
26. ContractStartDate   
27. ContractEndDate   
28. NewOrExistingContract   
29. RevisedContractStartDate   
30. RevisedContractEndDate   
31. ContractValue  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**

The Contract table was made by breaking off the columns from the Federal Stimulus dataset that related to the project’s contract. After breaking the Contract attributes off of the Federal Stimulus dataset, a foreign key to the StimulusTrackerID of the Project table is added due to the one-to-many relationship between the Contract and Project table and a foreign key to the VendorName attribute of the Vendor table is also added due to the one-to-many relationship between the Contract and Vendor entities. Contract’s one-to-many relationship with Project and Vendor is explained in depth in The Entity Relationship Diagram (ERD) section.

**Decomposed Table #5:** Funding

Attributes:   
4. FundingCategory  
5. FundingSource   
6. StimulusFunding  
7. DisplacedCityFunding   
8. AllOtherFunding   
9. AwardLeadCityAgency  
12. PercentOfFundsSpent   
13. DateFundsAwardedByFedOrState   
14. DateFundsAnnouncedByNYC  
18. InterimSpendingDeadline   
19. PercentToSpendByIntDeadline   
20. FinalSpendingDeadline  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**

The Funding table was made by breaking off the columns from the Federal Stimulus dataset that related to the project’s funding. After breaking the Funding attributes off of the Federal Stimulus dataset, a foreign key to the StimulusTrackerID of the Project table is added due to the one-to-many relationship between the Funding and Project entities that is explained in depth in The Entity Relationship Diagram (ERD) section.

**The Changes Needed to Normalize the Data in BCNF**

Although five entities have been identified within the Federal Stimulus dataset so far, and five tables have been made with the attributes of those five entities so far, the tables are still un-normalized in BCNF. This is because in BCNF “all attributes should depend on the key, whole key and nothing but the key (so help me Codd)”, but none of the five tables have a primary key assigned to it yet for its other attributes to rely on. Although one may think that the solution to this problem is to simply assign the ID attribute of each entity as its primary key, the true solution requires changes to be made to three out of the five entities so far in order for the attributes of each table to rely solely on the key and have the relational database be normalized in BCNF.

* **Change #1:** A **surrogate primary key must be added to the Payment table** in order for the Payment table’s attributes to rely solely on the key, the whole key, and nothing but the key.

Why? Because the Payment table does not have an attribute that can uniquely identify any of its tuples. Although one might think that PaymentID is a candidate key, there are over 400 tuples where PaymentID is NULL, which disqualifies it from being the primary key of the Payment table.

* **Change #2:** A **surrogate primary key must be added to the Contract table** in order for the Contract table’s attributes to rely solely on the key, the whole key, and nothing but the key.

Why? Because the Contract table does not have an attribute that can uniquely identify any of its tuples. Although one might think that ContractID is a candidate key, there are tuples where ContractID is NULL, which disqualifies it from being the primary key of the Contract table.

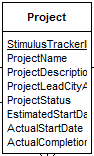
* **Change #3:** A **surrogate primary key must be added to the Funding table** in order for the Funding table’s attributes to rely solely on the key, the whole key, and nothing but the key.

Why? Because the Funding table does not have an attribute that can uniquely identify any of its tuples. There is no FundingID attribute in the Federal Stimulus dataset that uniquely identifies each fund that a Project receives. Therefore, a surrogate primary key must be added to the Funding table in order for its attributes to rely solely on the key, the whole key, and nothing but the key.

\* **Note:** The SQL commands to implement the three changes described above are shown in Section 5, The Create Table Statements. \*\*

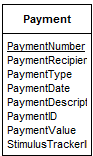
**The Entities/Tables (After Normalization in BCNF)**

**BCNF Decomposed Table #1:** Project

**Attributes:**1. ProjectName  
2. ProjectDescription  
3. StimulusTrackerID **(Primary Key)**  
10. ProjectLeadCityAgency  
11. ProjectStatus  
15. EstimatedStartDate   
16. ActualStartDate   
17. ActualCompletionDate

The Project table is now in BCNF because its attributes rely entirely on the StimulusTrackerID primary key because it that uniquely identifies each project and each project’s attributes, such as its ProjectName, ProjectDescription and ProjectStatus.

**BCNF Decomposed Table #2:** Payment

**Attributes:   
NEW:** PaymentNumber **(Surrogate Primary Key)**32. PaymentRecipient   
33. PaymentType   
34. PaymentDate   
35. PaymentDescription   
36. PaymentID  
37. PaymentValue  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**  
 The Payment table is now in BCNF because its attributes rely entirely on the PaymentNumber surrogate primary key, which uniquely identifies each payment received by a StimulusTrackerID and the specific Payment’s attributes, such as the PaymentDate and PaymentValue, that belongs to each unique payment.

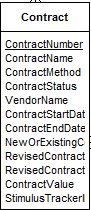
**BCNF Decomposed Table #3:** Vendor



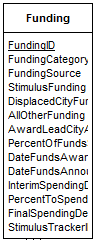
**Attributes:**25. VendorName **(Primary Key)**

The Vendor table is now in BCNF because the table only has one attribute. Therefore, the primary key of the table, VendorName, relies entirely on itself to identify each unique vendor.

**BCNF Decomposed Table #4:** Contract

**Attributes:   
NEW:** ContractNumber **(Surrogate Primary Key)**21. ContractName   
22. ContractMethod   
23. ContractStatus   
24. ContractID   
25. VendorName **(Foreign Key to VendorName in Vendor)**   
26. ContractStartDate   
27. ContractEndDate   
28. NewOrExistingContract   
29. RevisedContractStartDate   
30. RevisedContractEndDate   
31. ContractValue  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**  
  
 The Contract table is now in BCNF because its attributes rely entirely on the ContractNumber surrogate primary key, which uniquely identifies each contract received by a StimulusTrackerID and the specific Contract’s attributes, such as the ContractName and ContractValue, that belongs to each unique contract.

**BCNF Decomposed Table #5:** Funding

Attributes:   
**NEW**: FundingID **(Surrogate Primary Key)**   
4. FundingCategory  
5. FundingSource   
6. StimulusFunding  
7. DisplacedCityFunding   
8. AllOtherFunding   
9. AwardLeadCityAgency  
12. PercentOfFundsSpent   
13. DateFundsAwardedByFedOrState   
14. DateFundsAnnouncedByNYC  
18. InterimSpendingDeadline   
19. PercentToSpendByIntDeadline   
20. FinalSpendingDeadline  
3. StimulusTrackerID **(Foreign Key to StimulusTrackerID in Project)**

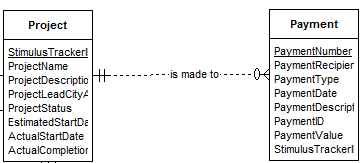
The Funding table is now in BCNF because its attributes rely entirely on the FundingID surrogate primary key, which uniquely identifies each fund received by a StimulusTrackerID and the specific Funding’s attributes, such as the FundingSource and FundingCategory, which belongs to each unique fund.

**Section 4.) The Entity Relationship Diagram (ERD)**

In this section, the relationships between the five entities (**Project, Payment, Funding, Contract, Vendor**) in the Federal Stimulus dataset so far and why Payment, Funding, Contract each have a foreign key to Project’s primary key, as well as why Contract has a foreign key to Vendor’s primary key, will be explained through the use of an Entity Relationship Diagram (ERD).

An ERD is essential to the normalization of any dataset because a many-to-many relationship between any pair of entities reveals additional tables that must be made in order to achieve normalization, as well as the fact that one-to-many relationships between entities require the primary key of entity in the one-side to be added as a foreign key to the entity in the many-side. Examples of the result of one-to-many and many-to-many relationships between entities are shown in the figures below for the Federal Stimulus dataset.

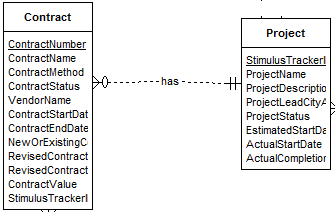
**Project** and **Payment’s** One-to-Many Relationship



As shown in the figure above, the relationship between the **Project** entity and the **Payment** entity is **a one-to-many relationship**. This is because a Project can have many Payments made to it, while a Payment belongs to one and only one Project. It is important to also note from the minimum cardinality of the relationship, however, that a Project does not have to have to have Payment made to it, whereas a Payment must belong to a Project.

Because of Payment and Project’s one-to-many relationship, the primary key of the one-side entity (Project’s StimulusTrackerID) is added as a foreign key to the to many-side’s entity, which is Payment, in order to represent the fact that each Payment belongs to a single Project.

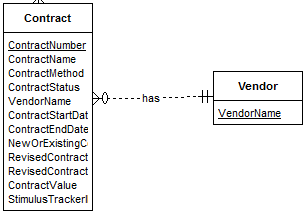
**Project** and **Contract’s** One-to-Many Relationship



As shown in the figure above, the relationship between the **Project** entity and the **Contract** entity is **a one-to-many relationship**. This is because a Project can have many Contracts, while a Contract belongs to one and only one Project. It is important to also note from the minimum cardinality of the relationship, however, that a Project does not have to have to have any Contracts, whereas a Contract must belong to a Project.

Because of Contract and Project’s one-to-many relationship, the primary key of the one-side entity (Project’s StimulusTrackerID) is added as a foreign key to the to many-side’s entity, which is Contract, in order to represent the fact that each Contract belongs to a single Project.

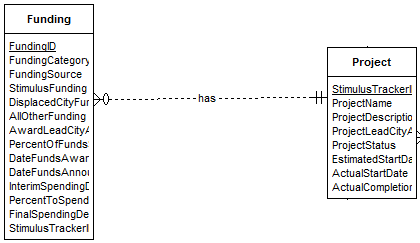
**Contract** and **Vendor’s** One-to-Many Relationship



As shown in the figure above, the relationship between the **Vendor** entity and the **Contract** entity is **a one-to-many relationship**. This is because a Vendor can have many Contracts, but a Contract can have one and only one Vendor. It is important to also note from the minimum cardinality of the relationship, however, that a Vendor does not have to have to have any Contracts, whereas a Contract must have a Vendor.

Because of Contract and Vendor’s one-to-many relationship, the primary key of the one-side entity (Vendor’s VendorName) is added as a foreign key to the to many-side’s entity, which is Contract, in order to represent the fact that each Contract has a single Vendor.

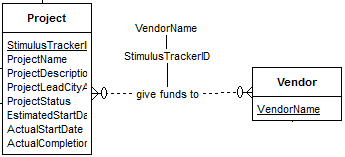
**Project** and **Funding’s** One-to-Many Relationship



As shown in the figure above, the relationship between the **Project** entity and the **Funding** entity is **a one-to-many relationship**. This is because a Project can have many funds, while a fund belongs to one and only one Project. It is important to also note from the minimum cardinality of the relationship, however, that a Project does not have to have to have any Funding, whereas a fund must belong to a Project.

Because of Funding and Project’s one-to-many relationship, the primary key of the one-side entity (Project’s StimulusTrackerID) is added as a foreign key to the to many-side’s entity, which is Funding, in order to represent the fact that each Funding belongs to a single Project.

**Project** and **Vendor’s** Many-To-Many Relationship



As shown in the figure above, the relationship between the **Project** entity and the **Vendor** entity is **a many-to-many relationship**. This is because a Vendor can provide funding to many Projects, and a Project can receive funds from more than one Vendor. It is important to also note from the minimum cardinality of the relationship, however, that a Project does not have to receive funds from a Vendor, and a Vendor does not have to give funds to any Projects.

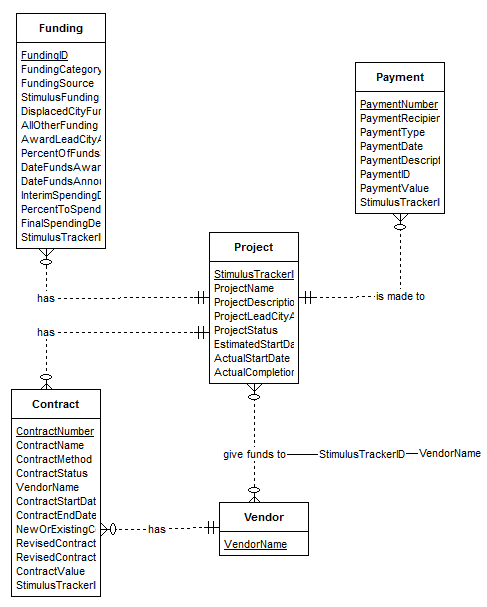
****

Because of Project and Vendor’s many-to-many relationship, an entirely new table must be made that has a composite primary key composed of the primary keys of Project and Vendor. The new table, Project\_Vendor, is necessary because it uniquely identifies which Vendor has given funds to which Project.

An entirely new table must be created for this many-to-many relationship, and many-to-many relationships between entities in general, because putting a foreign key to the primary key of Project in the Vendor table, or putting a foreign key to the primary key of Vendor in Project, would create an unnecessary data redundancy. Therefore, in order to avoid a redundancy in the relational database, a new table composed of the primary keys of each entity in the relationship must always be made for entities with a many-to-many relationship.

The SQL CREATE TABLE commands for how to make the tables for each of the six entities, including the newly discovered Project\_Vendor entity, are shown in the next section of this paper, which is “The CREATE TABLE Statements” section.

**The Overall ERD of the Federal Stimulus Dataset**

****

**A Note About Project\_Vendor’s Representation On the ERD**

Although the Project\_Vendor entity that is a result of the many-to-many relationship between Project and Vendor does not need to be explicitly shown in the overall ERD of the Federal Stimulus dataset, it is represented on the ERD as an extension of Project and Vendor’s relationship by the composite primary key that it will have: StimulusTrackerID and VendorName.

**Section 5.) The CREATE TABLE Statements**

In order to begin creating tables for the now six entities (**Project, Payment, Vendor, Contract, Funding, and Project\_Vendor**) of the Federal Stimulus Dataset, we must first load the Federal Stimulus dataset into a table that represents the entire un-normalized dataset. The name of this table will be Original\_Federal\_Data\_Import so that it is clear that this table will be used for the import of the Federal Stimulus data.

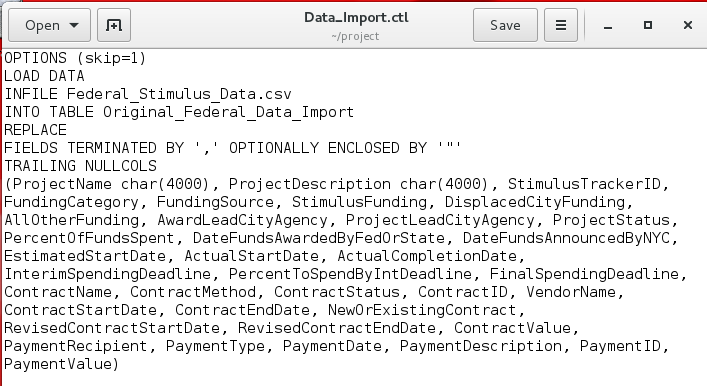
**CREATE TABLE: Original\_Federal\_Data\_Import**

This is the CREATE TABLE statement to create the Original\_Federal\_Data\_Import table with the appropriate data types for each of its attributes:

CREATE TABLE Original\_Federal\_Data\_Import (  
ProjectName varChar2(300),  
ProjectDescription varChar2(700),  
StimulusTrackerID char(25),  
FundingCategory varChar2(150),  
FundingSource varChar2(150),  
StimulusFunding NUMBER,  
DisplacedCityFunding varChar2(100),  
AllOtherFunding varChar2(100),  
AwardLeadCityAgency varChar2(50),  
ProjectLeadCityAgency varChar2(50),  
ProjectStatus varChar2(100),  
PercentOfFundsSpent char(19),  
DateFundsAwardedByFedOrState varChar2(100),  
DateFundsAnnouncedByNYC varChar2(100),  
EstimatedStartDate varChar2(100),  
ActualStartDate varChar2(100),  
ActualCompletionDate varChar2(100),  
InterimSpendingDeadline varChar2(100),  
PercentToSpendByIntDeadline char(19),  
FinalSpendingDeadline varChar2(100),  
ContractName varChar2(200),  
ContractMethod varChar2(200),  
ContractStatus varChar2(200),  
ContractID varChar2(200),  
VendorName varChar2(200),  
ContractStartDate varChar2(100),  
ContractEndDate varChar2(100),  
NewOrExistingContract varChar2(100),  
RevisedContractStartDate varChar2(100),  
RevisedContractEndDate varChar2(100),  
ContractValue NUMBER,  
PaymentRecipient varChar2(200),  
PaymentType varChar2(100),  
PaymentDate varChar2(150),  
PaymentDescription varChar2(240),  
PaymentID char(25),  
PaymentValue NUMBER  
);

**Loading the Federal Stimulus Data**

To begin loading the Federal Stimulus Data, the dataset must be downloaded from the NYC OpenData website as a .csv file. After the .csv file has been downloaded, a .ctl file taking into account the fact that the first line of the .csv file must be skipped, because it contains the data’s column names, and the fact that the fields in the .csv file are terminated by commas but also may be enclosed in quotes, needs to be created.

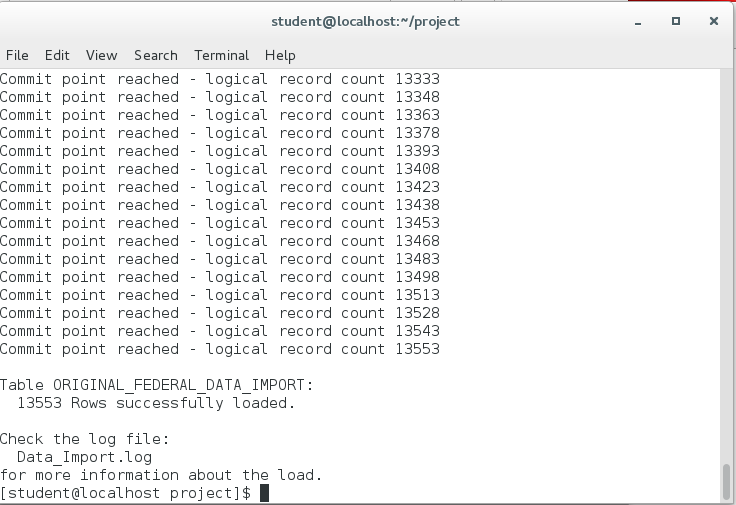


\* **Note**: The .ctl file used in this example was placed in the **project directory.**

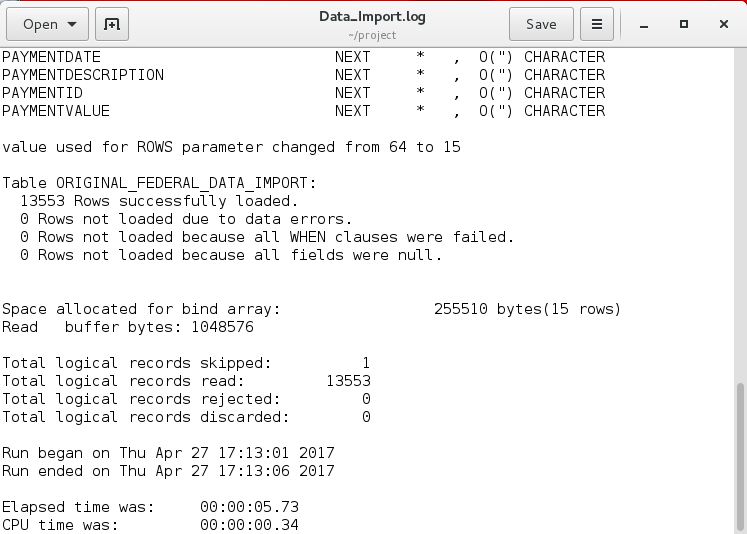
After the creation of the .ctl file, the following sqlldr command must be run at the command prompt of the terminal in order to load all of the Federal Stimulus data into the Original\_Federal\_Data\_Import table, assuming that the .csv and .ctl file is located in the student account’s project directory:

**sqlldr student/password DATA=Federal\_Stimulus\_Data.csv CONTROL=Data\_Import.ctl LOG=Data\_Import.log**

The result of running the above sqlldr command is shown in the figure below:



The Federal Stimulus dataset has 13553 rows, which means that the above sqlldr command successfully loaded every row in the dataset. The Data\_Import log file created from the sqlldr command also confirms that all 13553 rows of the Federal Stimulus dataset were loaded.



Now that the Original\_Federal\_Data\_Import table has been loaded with every tuple in the Federal Stimulus dataset, the normalized tables for each entity of the Federal Stimulus data can be created through CREATE TABLE AS SELECT statements.

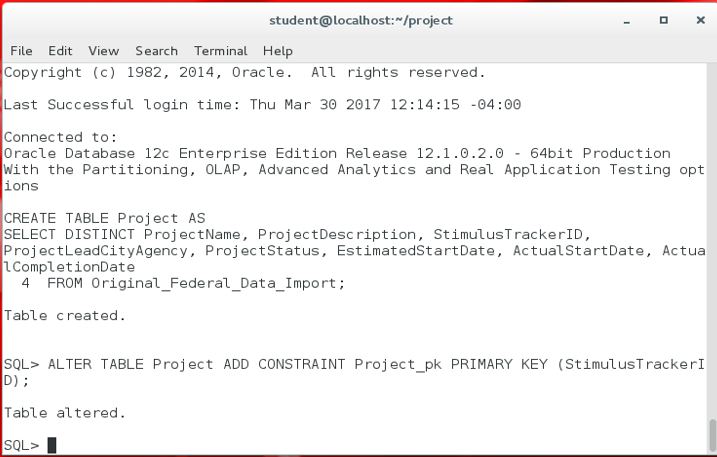
**CREATE TABLE: Project**

Create Table Statement:

CREATE TABLE Project AS  
SELECT DISTINCT ProjectName, ProjectDescription, StimulusTrackerID, ProjectLeadCityAgency, ProjectStatus, EstimatedStartDate, ActualStartDate, ActualCompletionDate  
FROM Original\_Federal\_Data\_Import;

Alter Table Statement:

ALTER TABLE Project ADD CONSTRAINT Project\_pk PRIMARY KEY (StimulusTrackerID);

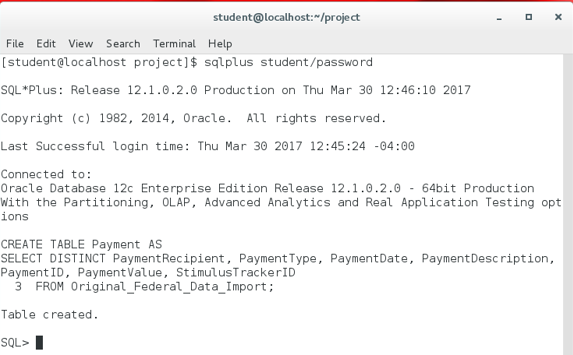


***Pictured Above:*** *Running the CREATE TABLE and primary key ALTER TABLE statements for Project.*

**CREATE TABLE: Payment**

Create Table Statement:

CREATE TABLE Payment AS  
SELECT DISTINCT PaymentRecipient, PaymentType, PaymentDate, PaymentDescription, PaymentID, PaymentValue, StimulusTrackerID  
FROM Original\_Federal\_Data\_Import;



***Pictured Above:*** *Running the CREATE TABLE statement for Payment.*

Alter Table Statement:

The first change needed to normalize the data, which was discussed in the “Normalizing the Data in BCNF & the Changes Needed to Normalize the Data” section, is to create a surrogate primary key for the Payment table named PaymentNumber. The commands to create said surrogate primary key are shown below.

**1. Create a sequence:**

CREATE SEQUENCE idSequence START WITH 100000  
INCREMENT BY 2 NOCACHE;

**2. Add the surrogate key attribute to Payment:**

ALTER TABLE Payment ADD (PaymentNumber INT);

**3. Update PaymentNumber to equal the idSequence:**

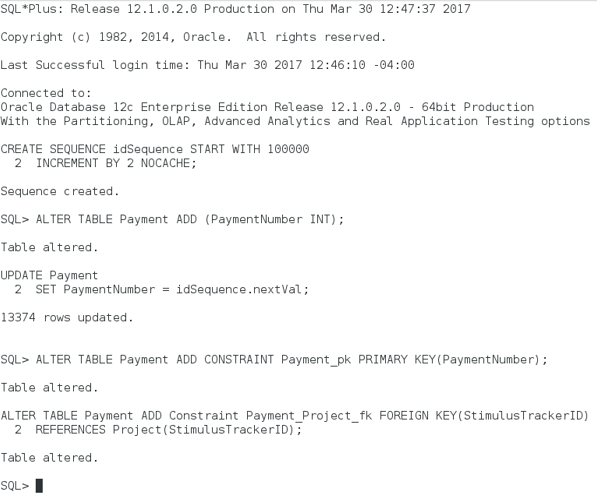
UPDATE Payment  
SET PaymentNumber = idSequence.nextVal;

**4. Alter the Payment table to make PaymentNumber the primary key:**

ALTER TABLE Payment ADD CONSTRAINT Payment\_pk PRIMARY KEY(PaymentNumber);

Because Payment’s StimulusTrackerID is a foreign key to Project’s StimulusTrackerID primary key, the Payment table was also altered with a foreign key constraint.

ALTER TABLE Payment ADD Constraint Payment\_Project\_fk FOREIGN KEY(StimulusTrackerID)  
REFERENCES Project(StimulusTrackerID);

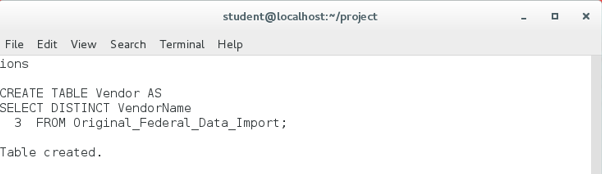
****

***Pictured Above:*** *Running the CREATE SEQUENCE and ALTER TABLE statements for Payment.*

**CREATE TABLE: Vendor**

Create Table Statement:

CREATE TABLE Vendor AS  
SELECT DISTINCT VendorName  
FROM Original\_Federal\_Data\_Import;

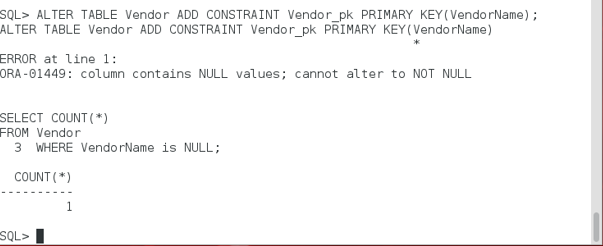


***Pictured Above:*** *Running the CREATE TABLE statement for Vendor.*

Alter Table Statement:

ALTER TABLE Vendor ADD CONSTRAINT Vendor\_pk PRIMARY KEY(VendorName);

**IMPORTANT NOTE:** Attempting to alter the Vendor table to have the VendorName attribute be its primary key results in an error stating that VendorName contains NULL values. A Count(\*) query on the number of VendorName null values reveals that there is only 1.

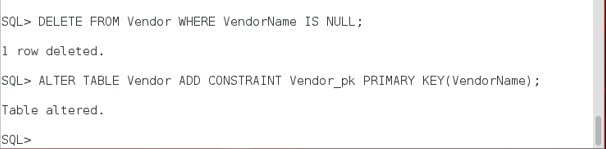


***Pictured Above:*** *ALTER TABLE’s NULL values error for Vendor and Count(\*) query for NULLs.*

This can be fixed by deleting the 1 null value with the following statement:

DELETE FROM Vendor WHERE VendorName IS NULL;

Because the single null value was removed from VendorName, it is now able to be altered as the primary key of the Vendor table:



***Pictured Above:*** *Running the DELETE FROM and now successful ALTER TABLE statement for Vendor.*

**CREATE TABLE: Contract**

Create Table Statement:

CREATE TABLE Contract AS  
SELECT DISTINCT ContractName, ContractMethod, ContractStatus, ContractID, VendorName, ContractStartDate, ContractEndDate, NewOrExistingContract, RevisedContractStartDate, RevisedContractEndDate, ContractValue, StimulusTrackerID  
FROM Original\_Federal\_Data\_Import;

Alter Table Statement:

The second change needed to normalize the data, which was discussed in the “Normalizing the Data in BCNF & the Changes Needed to Normalize the Data” section, is to create a surrogate primary key for the Contract table named ContractNumber. The commands to create said surrogate primary key are shown below.

**1. Add the surrogate key attribute to Contract:**

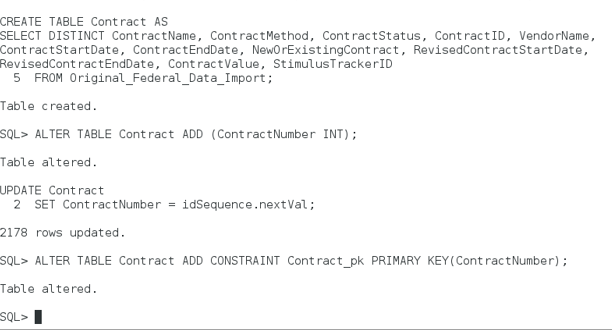
ALTER TABLE Contract ADD (ContractNumber INT);

**2. Update ContractNumber to equal the idSequence (which was created earlier):**

UPDATE Contract  
SET ContractNumber = idSequence.nextVal;

**3. Alter the Contract table to make ContractNumber the primary key:**

ALTER TABLE Contract ADD CONSTRAINT Contract\_pk PRIMARY KEY(ContractNumber);

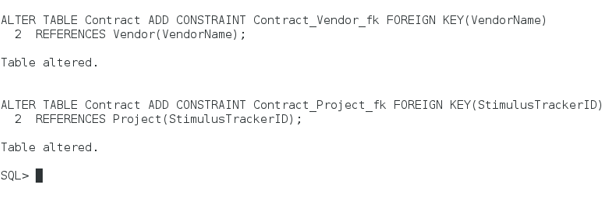


***Pictured Above:*** *Running the CREATE TABLE and primary key ALTER TABLE statements for Contract.*

Because StimulusTrackerID is a foreign key to Project’s StimulusTrackerID primary key and VendorName is a foreign key to Vendor’s VendorName primary key, the Contract table was also altered with foreign key constraints for those two attributes.

ALTER TABLE Contract ADD CONSTRAINT Contract\_Vendor\_fk FOREIGN KEY(VendorName)  
REFERENCES Vendor(VendorName);

ALTER TABLE Contract ADD CONSTRAINT Contract\_Project\_fk FOREIGN KEY(StimulusTrackerID)  
REFERENCES Project(StimulusTrackerID);



***Pictured Above:*** *Running the two foreign key ALTER TABLE statements for Contract.*

**CREATE TABLE: Project\_Vendor**

Create Table Statement:

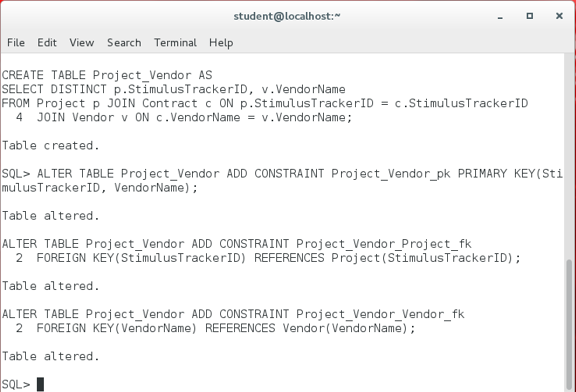
CREATE TABLE Project\_Vendor AS  
SELECT DISTINCT p.StimulusTrackerID, v.VendorName  
FROM Project p JOIN Contract c ON p.StimulusTrackerID = c.StimulusTrackerID  
JOIN Vendor v ON c.VendorName = v.VendorName;

Alter Table Statement:

ALTER TABLE Project\_Vendor ADD CONSTRAINT Project\_Vendor\_pk PRIMARY KEY(StimulusTrackerID, VendorName);

ALTER TABLE Project\_Vendor ADD CONSTRAINT Project\_Vendor\_Project\_fk  
FOREIGN KEY(StimulusTrackerID) REFERENCES Project(StimulusTrackerID);

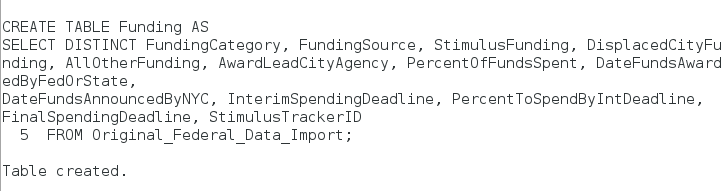
ALTER TABLE Project\_Vendor ADD CONSTRAINT Project\_Vendor\_Vendor\_fk  
FOREIGN KEY(VendorName) REFERENCES Vendor(VendorName);

  
***Pictured Above:*** *Running CREATE TABLE and ALTER TABLE statements for Project\_Vendor.*

**CREATE TABLE: Funding**

Create Table Statement:

CREATE TABLE Funding AS  
SELECT DISTINCT FundingCategory, FundingSource, StimulusFunding, DisplacedCityFunding, AllOtherFunding, AwardLeadCityAgency, PercentOfFundsSpent, DateFundsAwardedByFedOrState,  
DateFundsAnnouncedByNYC, InterimSpendingDeadline, PercentToSpendByIntDeadline,  
FinalSpendingDeadline, StimulusTrackerID  
FROM Original\_Federal\_Data\_Import;



***Pictured Above:*** *Running the CREATE TABLE statement for Funding.*

Alter Table Statement:

The third change needed to normalize the data, which was discussed in the “Normalizing the Data in BCNF & the Changes Needed to Normalize the Data” section, is to create a surrogate primary key for the Funding table named FundingID. The commands to create said surrogate primary key are shown below.

**1. Add the surrogate key attribute to Funding:**

ALTER TABLE Funding ADD (FundingID INT);

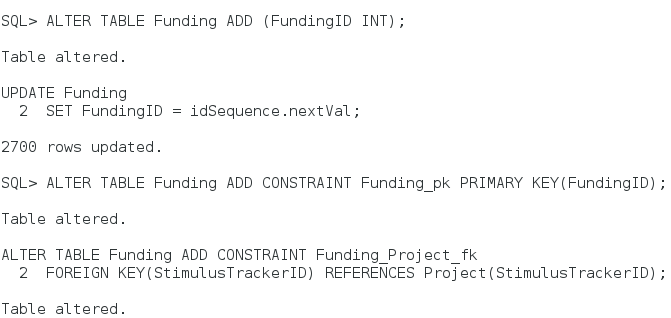
**2. Update FundingID to equal the idSequence (which was created earlier):**

UPDATE Funding  
SET FundingID = idSequence.nextVal;

**3. Alter the Funding table to make FundingID the primary key:**

ALTER TABLE Funding ADD CONSTRAINT Funding\_pk PRIMARY KEY(FundingID);

Because StimulusTrackerID is a foreign key to Project’s StimulusTrackerID primary key, the Funding table was also altered with a foreign key constraint.  
  
ALTER TABLE Funding ADD CONSTRAINT Funding\_Project\_fk   
FOREIGN KEY(StimulusTrackerID) REFERENCES Project(StimulusTrackerID);



***Pictured Above:*** *Running the primary key and foreign key ALTER TABLE statements for Funding.*

**Section 6.) Each Table’s Count and Why They Vary**

**Table #1: Project’s Count**

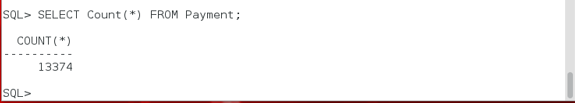
**Count Statement:** SELECT Count(\*) FROM Project;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 279 rows in the Project table because there are only 279 unique projects in the entire dataset that have either requested or received a payment to be funded. This means that the multiple times some projects either requested or received funding is the reason why there were 13553 total rows in the Federal Stimulus data instead of 279, as each Project did not receive or request funding only one time. When the Project table was created, it was created to only have distinct rows of Project, resulting in 279 rows because the duplicate values of each project were ignored and because there were only ever 279 unique projects in the entire Federal Stimulus dataset.

**Table #2: Payment’s Count**

**Count Statement:** SELECT Count(\*) FROM Payment;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 13374 rows in the Payment table because there are only 13374 unique payments in the entire Federal Stimulus dataset. Not every Project that requested funding received a Payment, which is why the count of the Payment table is significantly lower than the count of the Federal Stimulus dataset. This resulted in many rows in the Federal Stimulus dataset having NULL values in its Payment related columns to represent the fact that a Project did not receive a Payment. Because the Payment table was created to have distinct rows of Payment data, the 179 rows in the Federal Stimulus Dataset that had NULL values for all of its Payment columns where therefore ignored and not placed into the Payment table when it was created, resulting in the Payment table having a count lower than the Federal Stimulus dataset’s count.

**Table #3: Vendor’s Count**

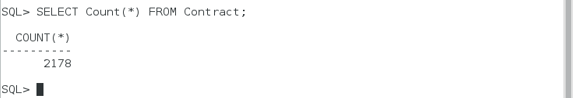
**Count Statement:** SELECT Count(\*) FROM Vendor;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 954 rows in the Payment table because there are only 954 unique Vendors in the entire Federal Stimulus dataset. The count of the Vendor table is significantly lower than the total amount of rows in the dataset because a single Vendor could have appeared in the dataset hundreds of times, resulting in hundreds of duplicate Vendors throughout the Federal Stimulus dataset. When creating the Vendor table, every unique Vendor was placed into the table only one time, which is why there are only 954 vendors in the Vendor table.

**Table #4: Contract’s Count**

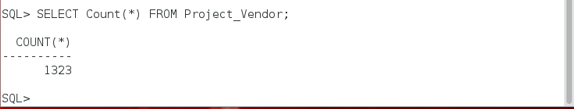
**Count Statement:** SELECT Count(\*) FROM Contract;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 2178 rows in the Contract table because there are only 2178 unique contracts in the entire Federal Stimulus dataset. Not every Project that requested funding received a Contract, which is why the count of the Contract table is significantly lower than the count of the Federal Stimulus dataset. This resulted in many rows in the Federal Stimulus dataset having NULL values in its Contract related columns to represent the fact that a Project did not receive a Contract. Because the Contract table was created to have unique rows of Contract data, the thousands of rows in the Federal Stimulus Dataset that had NULL values for all of its Contract columns where therefore ignored and not placed into the Contract table when it was created, resulting in the Contract table having a count significantly lower than the Federal Stimulus dataset’s count.

**Table #5: Project\_Vendor’s Count**

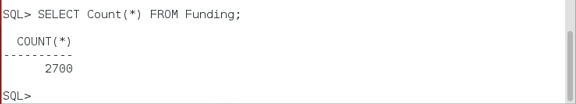
**Count Statement:** SELECT Count(\*) FROM Project\_Vendor;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 1323 rows in the Project\_Vendor table because there are only 1323 unique pairs of Vendors that gave funding to a Project in the entire Federal Stimulus dataset. The count of the Project\_Vendor table is significantly lower than the count of the Federal Stimulus dataset for two reasons. The first reason that Project\_Vendor’s count is 1323 is because not every Project had a Vendor give funds to it. This means that Projects that had no Vendors were excluded entirely from this count because Projects with no Vendors were excluded from the table when the Project\_Vendor table was created. The second reason is because if a Vendor provided funding to a Project multiple times, the Project\_Vendor table only stored the first unique instance of the Vendor providing funding for a Project and discarded the duplicate Project, Vendor rows. This means that although a Project may have received funding from a Vendor hundreds of times, the representation of the Project and Vendor is in the Project\_Vendor table once, which is why the count of the Project\_Vendor table is significantly lower than the Federal Stimulus dataset’s count.

**Table #6: Funding’s Count**

**Count Statement:** SELECT Count(\*) FROM Funding;



Although there are 13553 rows in total in the Federal Stimulus dataset, there are only 2700 rows in the Funding table because there are only 2700 unique funds in the entire Federal Stimulus dataset. Because the Funding table was created to have distinct rows of Funding data, the thousands of rows in the Federal Stimulus Dataset that had duplicate values for its Funding columns where therefore ignored and not placed into the Funding table when it was created. The duplicate rows of Funding data are a result of each Project’s Funding attributes, such as its StimulusFunding representing the Project’s total amount of StimulusFunding, being repeated in the Federal Stimulus dataset every time the Project requested or received a Payment towards the Project’s efforts. Therefore, because thousands of duplicate rows of Funding data were excluded when the Funding table was created, it resulted in the Funding table having a count significantly lower than the Federal Stimulus dataset’s count.

**Section 7.) Examples of Questions that the Data Answers**

Because the Federal Stimulus dataset is now normalized in BCNF, the tables resulting from the dataset’s normalization in BCNF can be used to effectively and accurately answer questions that one may have about the Federal Stimulus dataset.

An example of one question that one may have about the Federal Stimulus data is the following:

Question #1: How many payments has each project received?

The SQL to answer the above question is the following:

**SELECT substr(outer.ProjectName,0,55) as "Project Name",   
(SELECT Count(PaymentNumber) FROM Payment inner WHERE inner.StimulusTrackerID = outer.StimulusTrackerID) as "Payment Count"  
FROM Project outer;**

The above SQL query begins by selecting ProjectName from the outer Project table which is appropriately aliased as “outer” because it is the table outside of the correlated scalar subquery. Next, a correlated scalar subquery is used to return the count of PaymentNumber from Payment, which is aliased as inner because it is the table in the scalar subquery, where the “inner” Payment tuple’s StimulusTrackerID is equal to “outer” Project’s StimulusTrackerID in order to connect the two tables on their shared attribute, StimulusTrackerID, which is a foreign key in Payment to Project’s primary key of StimulusTrackerID. The scalar subquery in this example is correlated because it would not be able to run by itself due to its WHERE clause linking it to the outside Project table. Both of the attributes being SELECT-ed are also aliased in quotes to allow for there to be a space in the output’s column name.

**Note:** The substr method surrounding outer.ProjectName in the SELECT clause is used to output ProjectName up to 55 characters so that the output of the query is more organized.

Running the above SQL query results in the following output:



The fact that the query returned 279 rows means that the query returned the correct Payment count of each Project because there are exactly 279 Projects in the Project table.

An example of another question that one may have about the Federal Stimulus data is the following:

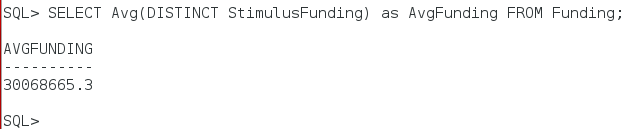
Question #2: Which projects have received greater than the average amount of stimulus funding, and what are those projects stimulus funding amount?

To approach this question, one should first find the average amount of stimulus funding from the Funding table.

The average amount of stimulus funding can be found with the following query where the all distinct StimulusFunding in Funding is put into the Avg() aggregate so that the average amount of StimulusFunding can be calculated:

**SELECT Avg(DISTINCT StimulusFunding) as AvgFunding FROM Funding;**

Running the above SQL query results in the following output:



The output of this query means that this question only wants Projects that have a StimulusFunding amount greater than 30,068,665.3 to be output. Rather than hardcode that number into the following query, however, the query itself is placed into the WHERE clause as an uncorrelated subquery to filter out the Projects that the question wants to be output.

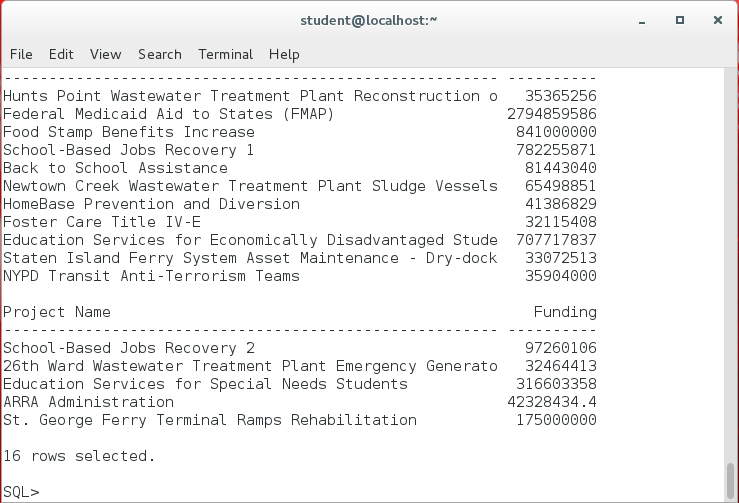
**SELECT DISTINCT substr(p.ProjectName,0,55) as "Project Name", f.StimulusFunding as Funding  
FROM Project p JOIN Funding f ON p.StimulusTrackerID = f.StimulusTrackerID  
WHERE f.StimulusFunding > (SELECT Avg(DISTINCT StimulusFunding) FROM Funding);**

The above SQL query begins by selecting ProjectName, aliased as “Project Name”, from Project, which is aliased as “p” in the FROM clause, and selecting StimulusFunding, aliased as Funding, from Funding, which is aliased as “f”, and JOIN-ed to the Project table on their shared attribute, StimulusFundingID. The JOIN between Project and Funding is an inner JOIN due to the term JOIN representing an inner JOIN by default, which means that only those tuples that have a match are included in the JOIN.

Next, the query that was used to find the average amount of StimulusFunding from Funding is placed as an uncorrelated subquery in the WHERE clause of the above SQL query. By doing this, the WHERE clause can filter out any Projects that fail to have a StimulusFunding amount greater than the average StimulusFunding returned by the uncorrelated subquery.

**Note:** The substr method surrounding ProjectName in the SELECT clause is used to output ProjectName up to 55 characters so that the output of the query is more organized.

Running the above SQL query results in the following output:



Out of the 279 projects that are in the Project table, only 16 Projects have a StimulusFunding amount greater than the average of the total amount of StimulusFunding awarded to Projects, and the StimulusFunding that each of those 16 Projects received is listed beside its ProjectName.

**Final Project Paper Checklist:**

**Section 1** - The Business Purpose of the Data– **pg. 1**

**Section 2** - A Description of the Un-normalized Data’s Form– **pg. 2-5**

**Section 3** - Normalizing the Data in BCNF & the Changes Needed to Normalize the Data - **pg. 6-10**

**Section 4** - The Entity Relationship Diagram (ERD) – **pg. 11-14**

**Section 5** - The CREATE TABLE Statements – **pg. 15-25**

**Section 6** - Each Table’s Count and Why They Vary - **pg. 25-27**

**Section 7** - Examples of Questions that the Data Answers - **pg. 28-30**