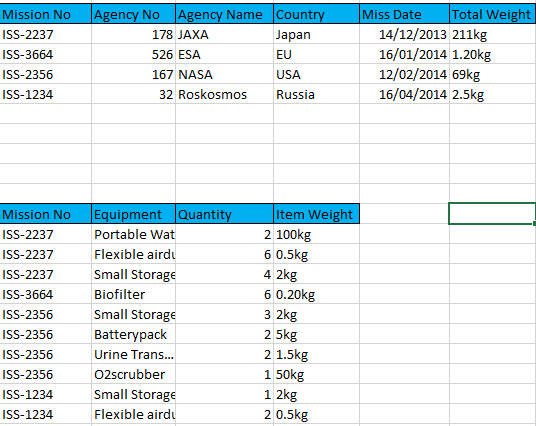
SQL IMPLEMENTATION

# INITIAL DATABASE:

# **Going to 1NF:**

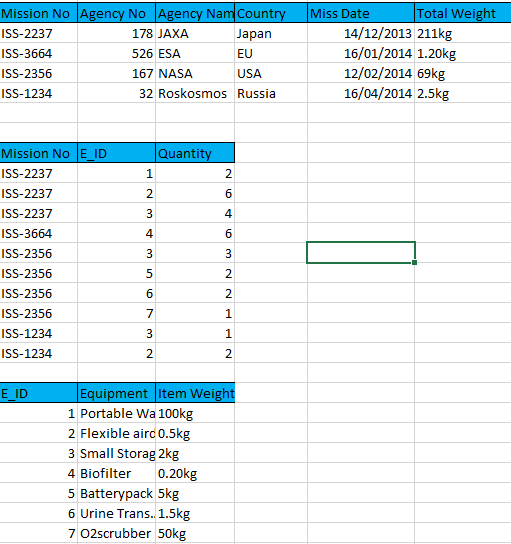
To do this, we need to make sure that each new table we create has a primary key, all values are atomic(no multiple values in a box) and there are no repeating values inside a table(same row written more than once). Evaluating our table, we can see that all the information up to the equipment column has to be repeated for each row. Therefore, we need to split this into a new table with primary key MissionNo, then we can just link that primary key to the rest of the table.



Primary key for first table is Mission No, for the 2nd table it is Mission No + Equipment. We are now in 1NF

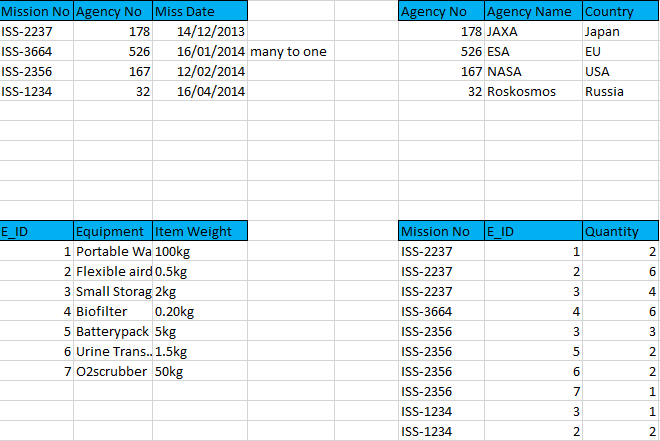
# **Going to 2NF:**

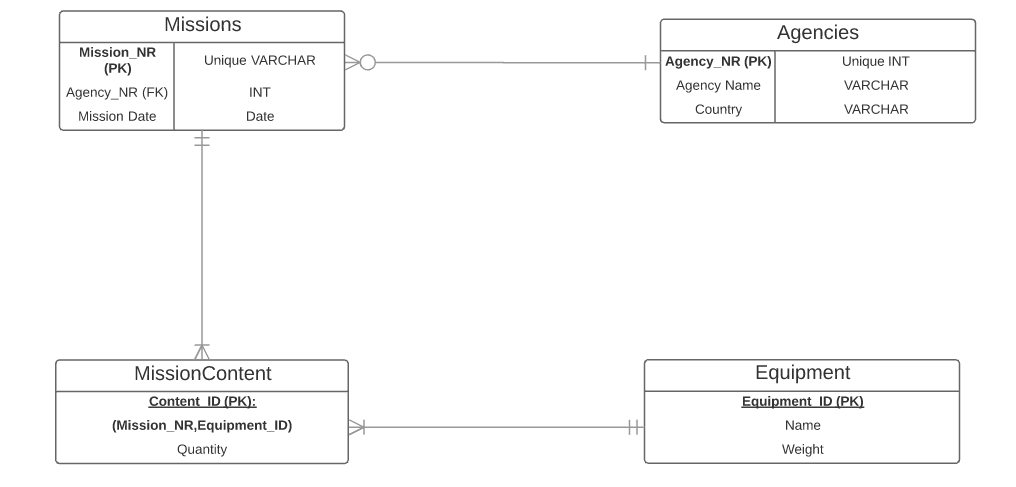
We now have a composite primary key in the second table, which means that for the table to be in 2NF, all columns in that table have to relate to both parts of the composite key. However, in our case, Item Weight only relates to the Equipment’s name, so we have to split it from that table. We will therefore create a new table where we will store Equipment Name and Item Weight and give a primary key to each row called E\_ID.



We now have three tables, we created a bridge table with the composite key of Mission No and E\_ID and the only other attribute in that table directly relates to the composite key. We are now in 2NF.

# **Going to 3NF:**

To get to 3NF, we have to go through each attribute and see if it directly relates to the primary key. If it doesn’t, then we need to split it from that table. In our case, we can see that Agency Name and Country do not relate to the mission no, they relate to Agency ID, so we need to create another table that holds that information.

We end up with 4 tables. If we check each attribute, we see that all of them relate directly to their primary key, which means we are now in 3NF and our database is normalized. This gives us the E-R diagram:

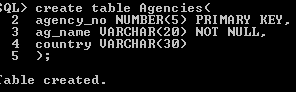
There is a one-to-many relationship between agencies and missions, meaning that one agency can participate in multiple missions, but a mission only has one agency.

There is a many-to-many relationship between missions and equipments, meaning that each equipment can belong to many missions, and each mission can have many equipments. To illustrate this relation, we have a bridge table called MissionContent, which has the composite PK of Mission\_No and E\_ID and holds the quantity attribute.

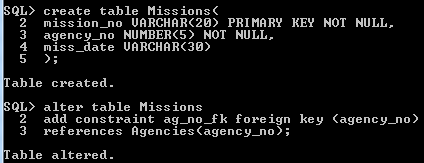
Note: I took away the Total Weight column, because it represents an aggregate function of already existing data and it has too many dependencies on other values. Instead of holding a column with that info, we can query it, as I will show below in an example.

# SQL IMPLEMENTATION:

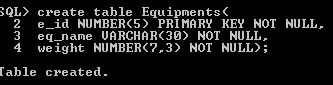
Create Agencies table:



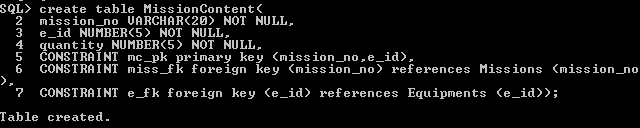
Create Missions table:



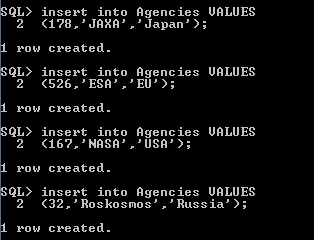
Create Equipments table:



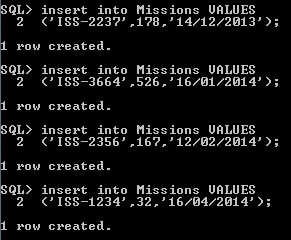
Create MissionContent table:



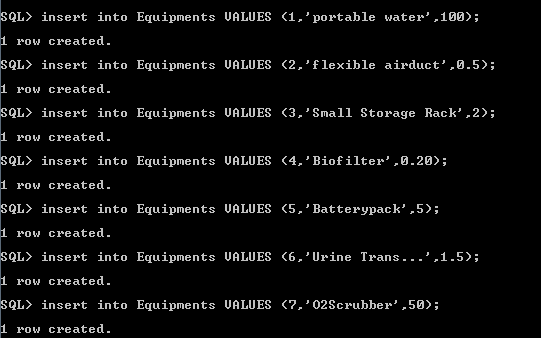
INSERT VALUES into Agencies:



Insert Values into Missions:



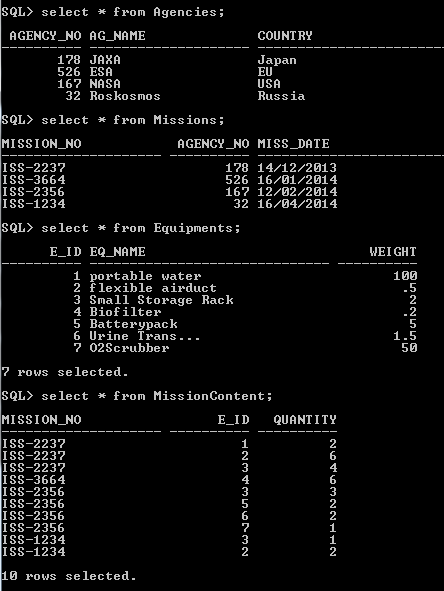
Insert Values into Equipments:



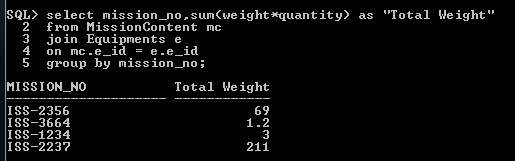


Insert Values into MissionContent:

Show Tables:



Now we are going to compute the total weight column that we previously removed from the database, due to too many dependencies:



We now have a fully working RDBMS with proof of running queries.