Joel Bravo

IST 659

**Fallout 4 Quest Database**

Part 1 – High-Level Design

Fallout 4 is an open-world role playing game developed by Bethesda Softworks and launched in November 10, 2015. The game supports a wide variety of quests, many of which are tied together through an overarching story, scattered through a persistent living world for the player character to discover and complete at their discretion, wherein the player character will be given varying types of rewards as compensation. Given that Fallout 4 is a non-linear game with multiple story paths and a large explorable area, it can be of great aid to players to be able to have a way to explore all the various quests that exist, as well as game elements that are directly tied into quests.

A Quest table will be essential in the creation of a quest database. The Quest table should contain the name of the quest, type of quest (e.g., Main Story Quest, Minuteman Quest, Miscellaneous Quest), the quest location, and the quest reward. Quest name, type, and location should all be required values, while quest reward, due to the prospect of multiple rewards, should be multivalue.

A Quest Giver table is also necessary to the database. A quest can be given by 0, 1, or many quest givers. It is possible that some quests are automatically started or discovered when interacting with an item, such as a computer terminal. A quest giver, in contrast, must give 1 or many quests. The quest giver name, sex, race, faction, and location will all need to be tracked. Quest giver name, sex, and race should be required.

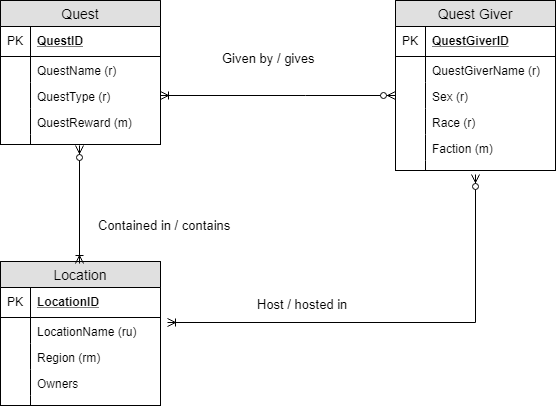
Finally, a Location table will be needed. A quest can be contained in 1 or many locations, while a location contains 0, 1, or many quests. A location can also host 0, 1, or many quest givers. Because quest givers can move around the game world on their own accord, a quest giver must be hosted in 1 or many locations. The Location table will need to track the location name, region, and owners. Location name is required and must be unique, while region is required and can is multivalued in order to accommodate the various subregions that can exist in the game world.

For simplification purposes, surrogate keys provided by the identity property in SQL server will be used as primary keys for all tables.

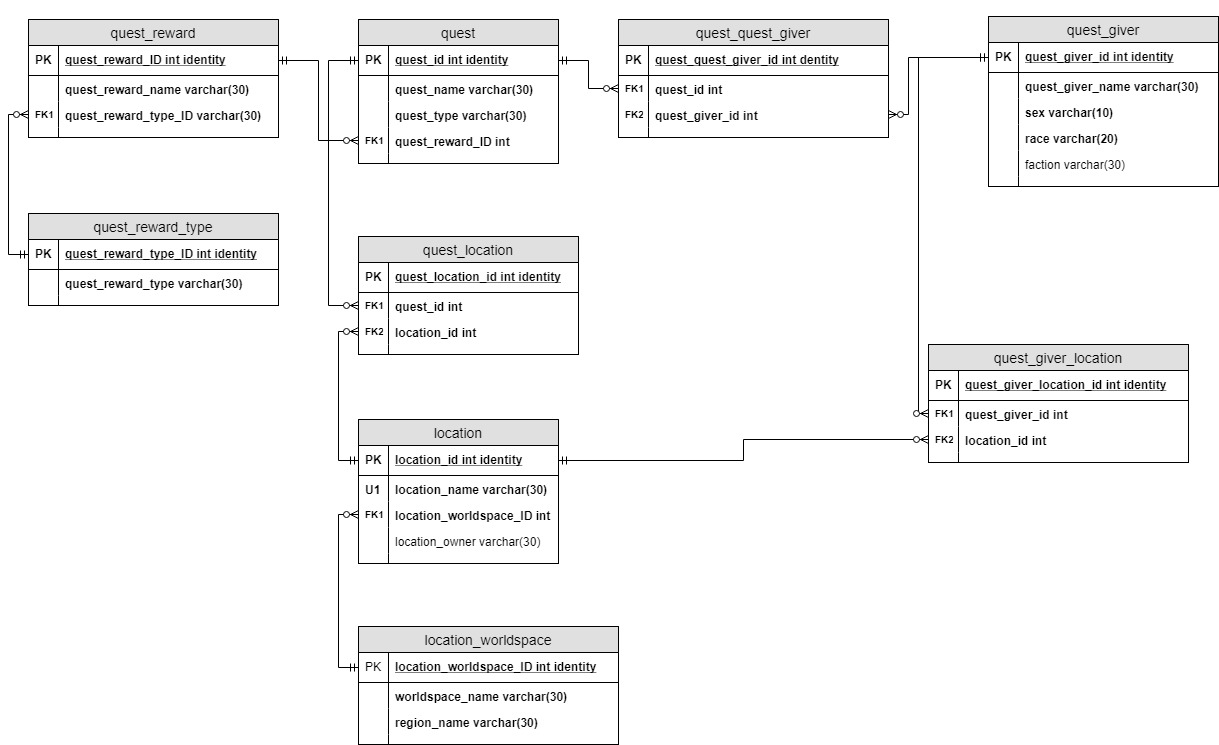
Upon completion, the database should be able to be used to answer questions such as the following:

* What quests are tied to specific factions?
* What quests provide a reward of the certain type, such as XP points or weapons?
* What quest givers of a specific faction can only be found in a specific location?

**Conceptual Model**

****

**Normalized Logical Model**



Normalizing the conceptual model brought several changes and expansions to the conceptual model. These include the following:

* All multivalue attributes were broken down into their own tables, such as quest\_reward.
* The many to many relationships that existed were converted into associative tables, such as quest\_location and quest\_quest\_giver\_location.
* As surrogate keys were in use for the logical model, the database was in 2NF.
* Because region was only transitively dependent on location\_id via worldspace, it was necessary to move worldspace and region into their own table, location\_worldspace, and link them together with worldspace as a foreign key in the location table.

In addition, after review it was decided to update the logical model by splitting quest reward type into its own table from quest reward in order to avoid anomalous data. This is referenced by the quest\_reward\_type table, which links to quest\_reward via the quest\_reward\_type\_ID foreign key.

Part 2 – Building the Database

**Data Definition Language**

With the database model finalized, I could now begin building the database. Below you will find the SQL code used in the creation of the database tables:

/\* Note the use of the DROP TABLE IF EXISTS statements. Thanks to using SQL Server 2017,

these statements will ensure that we can run these commands repeatedly without issue by

automatically dropping the named table prior to creating it if it exists \*/

DROP TABLE IF EXISTS QuestReward

CREATE TABLE QuestReward (

quest\_reward\_ID int identity primary key,

quest\_reward\_name varchar(50),

quest\_reward\_type varchar(30)

)

DROP TABLE IF EXISTS LocationWorldspace

CREATE TABLE LocationWorldspace (

location\_worldspace\_ID int identity primary key,

worldspace\_name varchar(100),

region\_name varchar(100)

)

DROP TABLE IF EXISTS Location

CREATE TABLE Location (

location\_ID int identity primary key,

location\_name varchar(50),

location\_worldspace\_ID int foreign key REFERENCES LocationWorldspace(location\_worldspace\_ID),

location\_owner varchar(50)

)

DROP TABLE IF EXISTS Quest

CREATE TABLE Quest (

quest\_ID int identity primary key,

quest\_name varchar(30),

quest\_type varchar(30),

quest\_reward\_ID int foreign key REFERENCES QuestReward(quest\_reward\_ID)

)

DROP TABLE IF EXISTS QuestLocation

CREATE TABLE QuestLocation (

quest\_location\_ID int identity primary key,

quest\_ID int foreign key REFERENCES Quest(quest\_ID),

location\_ID int foreign key REFERENCES Location(location\_ID)

)

DROP TABLE IF EXISTS QuestGiver

CREATE TABLE QuestGiver (

quest\_giver\_ID int identity primary key,

quest\_giver\_name varchar(30),

sex varchar(20),

race varchar(20),

faction varchar(30)

)

DROP TABLE IF EXISTS QuestQuestGiver

CREATE TABLE QuestQuestGiver (

quest\_quest\_giver\_ID int identity primary key,

quest\_ID int foreign key REFERENCES Quest(quest\_ID),

quest\_giver\_ID int foreign key REFERENCES QuestGiver(quest\_giver\_ID)

)

DROP TABLE IF EXISTS QuestGiverLocation

CREATE TABLE QuestGiverLocation (

quest\_giver\_location\_ID int identity primary key,

quest\_giver\_ID int foreign key REFERENCES QuestGiver(quest\_giver\_ID),

location\_ID int foreign key REFERENCES Location(location\_ID),

)

DROP TABLE IF EXISTS QuestRewardType

CREATE TABLE QuestRewardType (

quest\_reward\_type\_ID int identity primary key,

quest\_reward\_type varchar(30)

)

There is a deliberate order in how the statements are written. Any CREATE TABLE that involves a dependency on a foreign key would be created after its parent table is created. For example, QuestQuestGiver has quest\_giver\_ID as a foreign key, which references the QuestGiver table. As such, QuestGiver has to be created first.

**Data Creation**

With our tables created, we can now begin inserting our data via SQL statement. See below for the INSERT statements used. For the sake of brevity, I am including the INSERT statements for three tables:

/\*location\_name references a quest location within a game region and worldspace.

location\_worldspace\_ID acts as a FK to specify region and worldspace affiliation.

Note that Unclaimed and Independent are NOT the same. Unclaimed is a location that has no owner or is abandoned,

while Indpendent signifies that the location is not affialated with any major faction.

\*/

('Vault 111', 1, 'Unclaimed'),

('Diamond City', 16, 'Diamond City'),

('Valentine Detective Agency', 16, 'Nick Valentine'),

('Fort Hagen', 3, 'Conrad Kellog'),

('Crater of Atom', 14, 'Children of Atom'),

('C.I.T. Ruins', 13, 'The Institute'),

('Railroad HQ', 18, 'The Railroad'),

('The Institute', 13, 'The Institute'),

('Museum of Freedom', 2, 'Commonwealth Minutemen'),

('The Castle', 19, 'Commonwealth Minutemen'),

('Abnernathy Farm', 1, 'Independent'),

('Saugus Ironworks', 1, 'Forged'),

('Cambridge Police Station', 13, 'Brotherhood of Steel'),

('The Prydwen', 17, 'Brotherhood of Steel'),

('Boston Common', 16, 'Independent'),

('The Switchboard', 1, 'The Institute'),

('Libertalia', 1, 'Raiders'),

('Mass Fusion Building', 16, 'The Gunners'),

('Vault 81', 1, 'Vault 81'),

('Covenant', 1, 'Independent'),

('USS Constitution', 20, 'Independent'),

('Cabot House', 18, 'Independent'),

('Goodneighbor', 16, 'Goodneighbor'),

('Atom Cats Garage', 1, 'Atom Cats'),

('The Slog', 1, 'Commonwealth Minutemen'),

('Waltz Consumer Electronics', 13, 'Unclaimed'),

('The Mechanists Lair', 17, 'The Mechanist'),

('Nakano Residence', 1, 'Independent'),

('The Last Plank', 21, 'Harbormen'),

('Acadia', 23, 'Acadia'),

('The Nucleus', 24, 'Church of the Children of Atom'),

/\* Notice that this location has the same name as its region? This is accurate per Bethesda's documentation.

Why make it potentially confusing? Guess it isn't a Bethesda game without the potential for bugs!

\*/

('Vault 88', 1, 'Vault 88'),

('Nuka-World Transit Center', 1, 'The Gunners'),

('Nuka Station', 31, 'Overboss'),

('Fizz Top Grille', 31, 'Overboss'),

('The World of Refreshment', 29, 'Unclaimed')

/\* The heart of the database. This is a selection of quests from Fallout 4 and its DLCs

That is meant to provide a repsenstation of the scope of the game and what the database

hopes to achieve fully.

\*/

INSERT INTO Quest (quest\_name, quest\_type, quest\_reward\_ID) VALUES

('War Never Changes', 'Main Quest', 1109),

('Out of Time', 'Main Quest',1112),

('Jewel of the Commonwealth', 'Main Quest', 1110),

('Unlikely Valentine', 'Main Quest', 1113),

('Getting a Clue', 'Main Quest', 1116),

('Reunions', 'Main Quest', 1117),

('Dangerous Minds', 'Main Quest', 1110),

('The Glowing Sea', 'Main Quest', 1110),

('Hunter/Hunted', 'Main Quest', 1110),

('The Molecular Level', 'Main Quest', 1110),

('Institutionalized', 'Main Quest', 1110),

('When Freedom Calls', 'Minuteman Main Quest', 1120),

('Taking Independence', 'Minuteman Main Quest', 1123),

('Old Guns', 'Minuteman Main Quest', 1124),

('Returning the Favor', 'Minuteman Side Quest', 1126),

('Out of the Fire', 'Minuteman Side Quest', 1128),

('Call to Arms', 'Brotherhood of Steel Main Quest', 1134),

('Show no Mercy', 'Brotherhood of Steel Main Quest', 1139),

('Road to Freedom', 'Railroad Main Quest', 1110),

('Tradecraft', 'Railroad Main Quest', 1150),

('Synth Retention', 'Institute Main Quest', 1110),

('Mass Fusion', 'Institute Main Quest', 1110),

('Here Kitty, Kitty', 'Side Quest', 1110),

('Human Error', 'Side Quest', 1185),

('Last Voyage of the U.S.S. Constitution', 'Side Quest', 1186),

('Special Delivery', 'Side Quest', 1110),

/\*Miscellaneous Quest per Bethesda is not the same as Side Quest.

It appears Misc Quest is reserved for short quests that do not receive their own entry in the quest log \*/

('Atom Cats', 'Miscellaneous Quest', 1201),

('Giddyup n Go', 'Miscellaneous Quest', 1110),

/\*The following are quests tied to specific Downloadable Content, hence the addtion of the DLC name to the

quest type value\*/

('Mechanical Menance', 'Automatron Main Quest', 1214),

('Restoring Order', 'Automatron Main Quest', 1215),

('Far From Home', 'Far Harbor Main Quest', 1217),

('Walk in the Park', 'Far Harbor Main Quest', 1218),

('Where You Belong', 'Far Harbor Main Quest', 1110),

('Best Left Forgotten', 'Far Harbor Main Quest', 1110),

('Vault-Tec Calling', 'Vault-Tec Workshop Main Quest', 1222),

('All Aboard', 'Nuka-World Main Quest', 1224),

('Taken for a Ride', 'Nuka-World Main Quest', 1226),

('An Ambitious Plan', 'Nuka-World Main Quest', 1228),

('A World of Refreshment', 'Nuka-World Main Quest', 1229)

**Data Manipulation and Answering Questions**

With the tables and data in place, I can now use the databases to start manipulating data further and answer some of the business questions that that were originally considered.

At the most basic level, I can use a SELECT alongside a JOIN to start answering some basic questions. For example, perhaps a user wants to know what rewards and tied to a specific quest. The following SELECT statement can help answer this:

SELECT

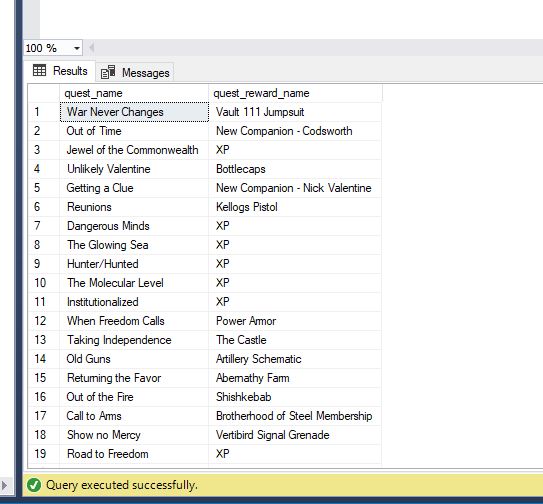
Quest.quest\_name,

QuestReward.quest\_reward\_name

FROM Quest

INNER JOIN QuestReward ON Quest.quest\_reward\_ID = QuestReward.quest\_reward\_ID

Running the statement gives us the following results:



While running SELECT statements works just fine, it may prove more efficient and secure to create views or stored procedures that can do the same thing. For example, if someone wanted to create a game guide and needed to have an easy way to reference the quests associated with major NPCs, we can easily create views that can do so. Let’s assume a user wants to know every quest offered by the NPC Nick Valentine, who is also a companion of the player character. Creating a view to do is simple enough, as seen below:

/\* Rather than let users run amok in the tables, views can be used to provide access to specific information \*/

/\*For example, the following view could be used to answer the question of which quests are offered

by Nick Valentine, one of the major NPCs in the game \*/

CREATE VIEW nicks\_quests AS

SELECT Quest.quest\_name, QuestGiver.quest\_giver\_name

FROM Quest

INNER JOIN QuestQuestGiver ON Quest.quest\_ID = QuestQuestGiver.quest\_ID

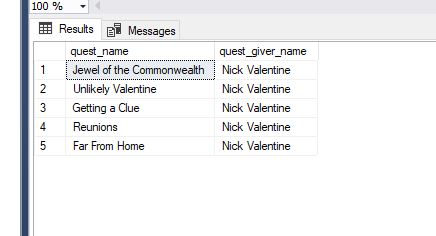
INNER JOIN QuestGiver ON QuestQuestGiver.quest\_giver\_ID = QuestGiver.quest\_giver\_ID

WHERE QuestGiver.quest\_giver\_name = 'Nick Valentine'

/\* Running the view nicks\_quests will then generate the information we requested \*/

SELECT \* FROM nicks\_quests

And the result:



Another example could involve creating a view that lists quests and their reward and reward type:

/\* Alternatively, perhaps we want to get a list of all quests and their reward and reward type \*/

CREATE VIEW quest\_loot AS

SELECT

Quest.quest\_name,

Quest.quest\_type,

QuestReward.quest\_reward\_name,

QuestRewardType.quest\_reward\_type

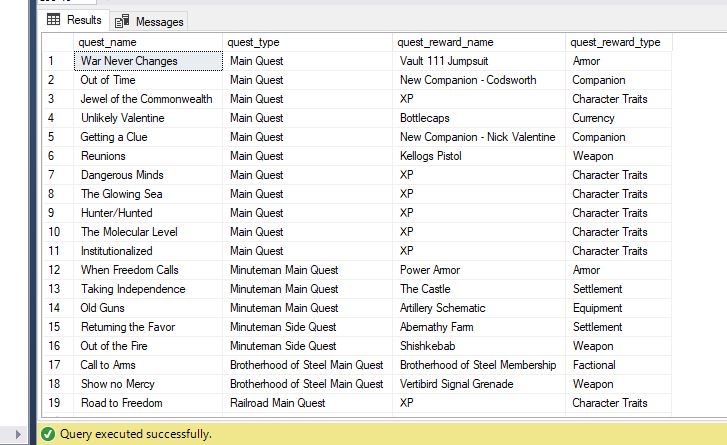
FROM Quest

INNER JOIN QuestReward ON Quest.quest\_reward\_ID = QuestReward.quest\_reward\_ID

INNER JOIN QuestRewardType ON QuestReward.quest\_reward\_type\_ID = QuestRewardType.quest\_reward\_type\_ID

SELECT \* FROM quest\_loot

And the result:



Building upon the previous examples, stored procedures could be used to help simplify routine tasks, like adding in additional data:

/\*There will also be scenarios where a DBA will want to insert more entries into tables, but prevent users from having full access

or avoid having to repetitively enter the same SQL statements. Creating a stored procedure, in this case for

entering a new quest giver provided by a mod called Tales of the Commonwealth, would be an ideal solution \*/

GO

CREATE PROCEDURE UpdateQuestGiver @NPC\_name varchar(20), @NPC\_sex varchar(10), @NPC\_race varchar(20), @NPC\_faction varchar(40)

AS

BEGIN

INSERT INTO QuestGiver (quest\_giver\_name, sex, race, faction) VALUES

(@NPC\_name, @NPC\_sex, @NPC\_race, @NPC\_faction)

END

GO

EXEC UpdateQuestGiver 'Paladin Brend', 'Female', 'Human', 'Brotherhood of Steel'

/\*Running the procedure results in a new entry for Paladin Brend, who will automatically get a primary key.

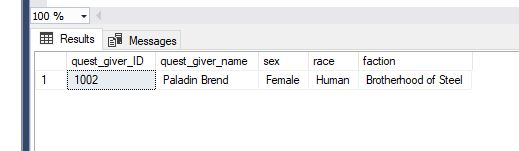
As an aside, Tales from the Commonwealth is a massive, fantastic mod that is very much professional grade in its

design and voice acting. These guys definitely need to start making RPGs of their own. \*/

We can confirm the addition by running a quick SELECT statement:

SELECT \* FROM QuestGiver

WHERE quest\_giver\_name = 'Paladin Brend'

****

It’s possible that users will need to make some updates or changes to data that has already been inserted. A simple UPDATE statement will work fine enough:

/\*After speaking with some stakeholders, we realize that a mistake has been made in the QuestGiver table. The quest giver Curie is listed as a Human when her Race value should be set to Synth. Thankfully, it is easy enough to rectify this mistake.\*/

UPDATE QuestGiver

SET race = 'Synth'

WHERE quest\_giver\_name = 'Curie'

UPDATE statements can potentially be dangerous, however, as not specifying any conditions when there should be can easily result in unintended changes and lots of heartache. So to avoid such scenarios and simplify things, a stored procedure may make more sense. Consider the scenario listed below:

/\*Building on the above, a common subplot within the game involves characters that appear human but are secretly Synths, sometimes without even the character realizing it. Since this revelation can be hidden well by the game, in some cases not visible unless one inspects the character's body for loot, quest givers whose race is set to Human originally made need to be updated later to Synth. To avoid repetition, we can package the above statement as a stored procedure \*/

GO

CREATE PROCEDURE QuestGiverSynthStatus @NPC\_name varchar(20)

AS

BEGIN

UPDATE QuestGiver

SET race = 'Synth'

WHERE quest\_giver\_name = @NPC\_name

END

GO

By setting the conditions already, in this case locking the changes only to quest\_giver\_name, the user can simply enter the name of the NPC that needs their race value adjusted and the procedure will handle the rest. Consider the NPC Sturges, who as seen below was originally set to Human as their race value:



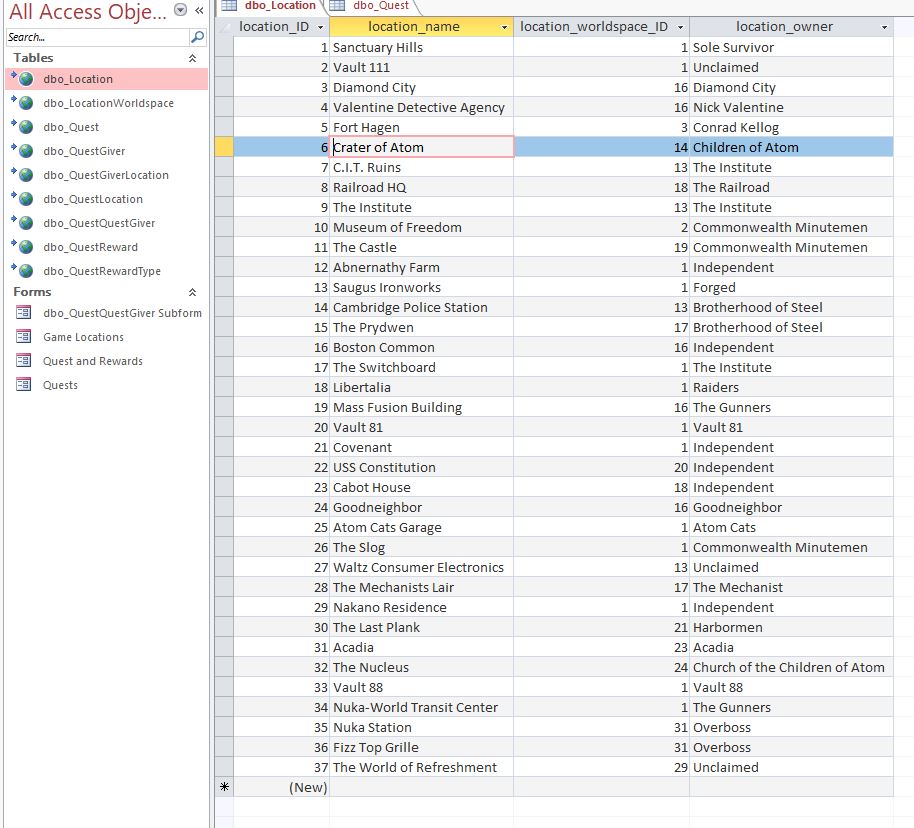
Not immediately apparent to the player character is that he is actually a Synth. Utilizing the QuestGiverSynthStatus procedure let us easily correct this:

EXEC QuestGiverSynthStatus 'Sturges'



**Implementation**

While I could certainly have users work on their analysis by just using SQL Server directly, this is not ideal from an end user standpoint. Rather than making things over complicated, a streamlined, easy to use front end that connects to the database would be a good choice. Microsoft Access provides everything we need for a front-end. After creating the requisite ODBC to link Access to SQL server, we can then bring in our tables and link them together via their relationships with one another. At this point, a user could now begin to add in data to the various tables we have imported in. Consider the example below:

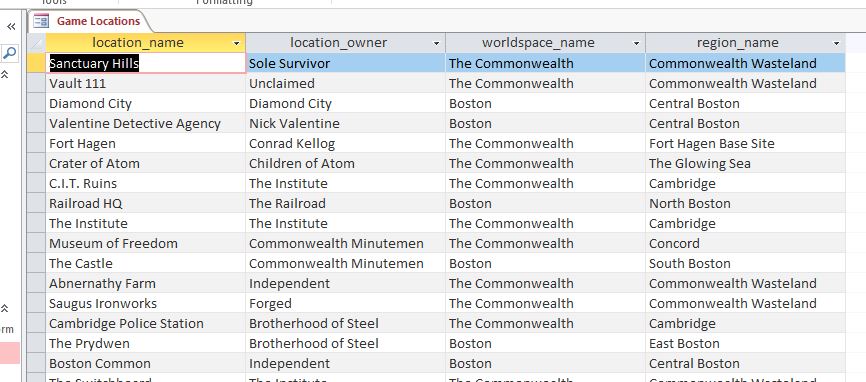


Here we have the Location table in Datasheet view. All locations are listed for users to view, along with the means to enter new locations down below. As Access is linked to SQL Server, any changes made, such as new data entries will automatically be updated in SQL Server.

Within Access, I can also create Forms that can help better answer questions that users may have. For instance, we can create a form that can show us which quests are tied to which factions:

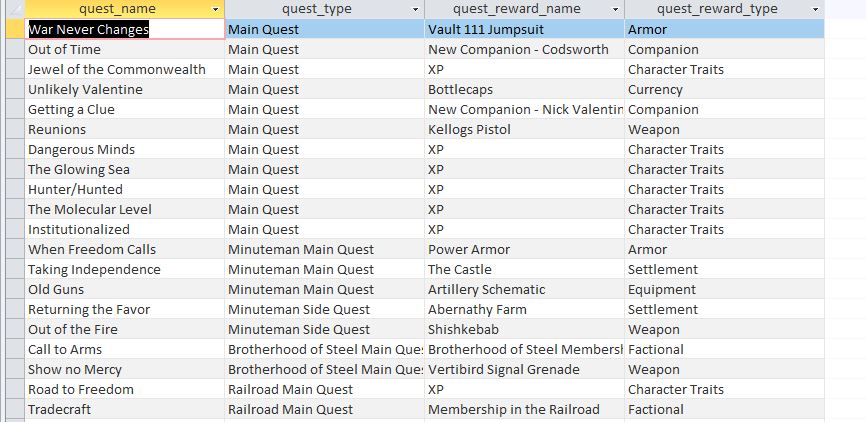


It’s possible to be more ambitious as well. For example, a form could be created that shows all locations alongside their associated region and worldspace:



Since the primary and foreign keys don’t need to be viewed, I have hidden them to greater readability.

A similar form can also be created to show the game’s quests, quest types, reward, and reward type:



**Reflection**

Aside from the fun I had creating this database, it was an illuminating experience taking a relatively simple problem, in this case finding a way to list the quests from Fallout 4, and running through the entire process of creating a living database. From developing the initial business questions and physical model to creating the front-end connection to my database, this project served as a great example of how every single step in the database creation process is informed by and informs all the others. Slicing up the project into individual layers really let me concentrate on the task at hand and made something that could appear daunting much more manageable. Professionally, I can definitely see the value in implementing similar projects in a layered fashion, especially as I deal with stakeholders and domain experts.

This database is ultimately a small scale version of what it could be due to time and scope constraints. With more time, I could easily see myself expanding its scope to include more areas, such as including enemy types, and player character skills. I would also like to look into other front-end tools beyond Access, such as potentially Tableau, once my skill sets grow to be able to master that level of detail. I can easily see myself gradually working on this database as a sandbox to test my knowledge as I progress through Syracuse’s data science program.