



**Accuro**

**QHR Technologies**

**Design and Testing**

November 8<sup>th</sup>, 2019

**COSC 499 Team Members:**

Jen Labossiere - Technical Lead

Robert Rosa - Technical Manager

Samantha DesBrisay - Client Liaison

Ethan Godden - Integration Lead

# **Table of Contents**

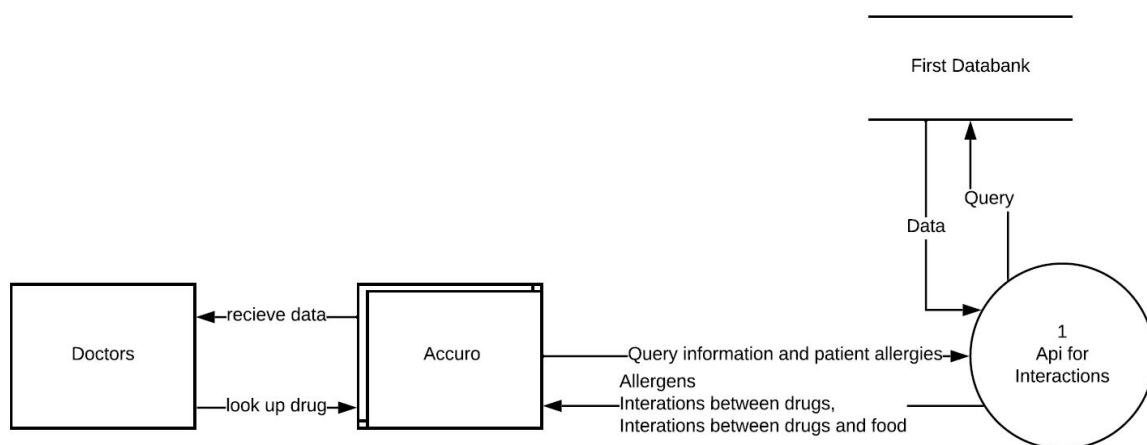
<b>Description</b>	<b>2</b>
<b>System Architecture Design</b>	<b>2</b>
Level 0 DFD	2
Level 1 DFD	3
API Description	3
<b>User Interface</b>	<b>4</b>
<b>Query Optimization</b>	<b>5</b>
Parallelism	5
Reusing Results	5
Pagination	5
Manipulating Relational Algebra	6
<b>Technical Specifications</b>	<b>7</b>
Programming Language	7
Dependency Management Software	7
Database Software	8
Testing Software	9
<b>Entity Relation Diagrams (ERD)</b>	<b>10</b>
Identifiers and Attributes Top-Level ERD	10
Drug-Drug Interaction Module ERD	13
Drug-Allergy Interaction Module ERD	15
Drug-Food Interaction Module ERD	17

## Description

The system that needs to be built is an API for gathering information about drugs from First Databank when prescribing drugs to patients in Accuro. It needs to be able to return all drugs whose name starts with a given prefix. It also needs to return all relevant drug interactions and allergies in order to properly warn physicians of all potential hazards. All allergies will come from Accuro, and all drug interactions will come from First Databank.

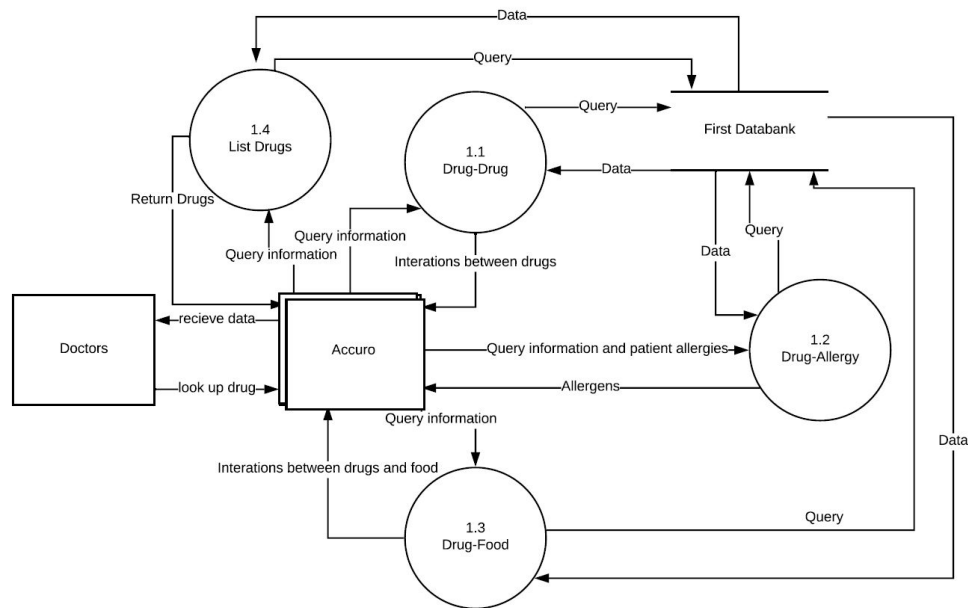
## System Architecture Design

### Level 0 DFD



Doctors (which are the only users of Accuro) will access accuro looking for a drug. After this, Accuro will send both the query information and the patient allergies to our API. The API will then query the database, receive information, and send the interactions and allergens to Accuro. Lastly, Accuro will return the information to the doctors.

## Level 1 DFD



The level 1 DFD differs from the level 0 DFD in that the API has been broken up into list drugs, drug-to-drug interactions, drug-to-allergy interaction, and drug-to-food interactions. Each will be individually queried by the database and return the necessary information to Accuro based on what a user inputs. It should be noted that we will not be receiving patient data; the client will use our API to find general drug information and then combine it with patient data on their end.

## API Description

The following is the interface we will be implementing. These are the methods that our client will see, and they will be what is connected to some user interface provided by the client.

```

interface DrugInfo {
    List<Drug> queryDrugs(String prefix);

    List<Allergy> queryAllergyInteractionsOfDrug(Drug drug, Patient patient);

    List<Food> queryFoodIterationsOfDrug(Drug drug);

    List<Drug> queryDrugInteractionsWithOtherDrugs(Drug drug);
}

```

The classes `Drug`, `Allergy`, `Patient` and `Food` are container classes that contain identifying information about a drug, allergy, or food. For example, the `Drug` class would contain an id, brand name, label name, form, and route. By making classes for these objects, it allows any user interface to display information about a specific drug, allergy or food easier because all the frontend has to do is call the getters in these classes.

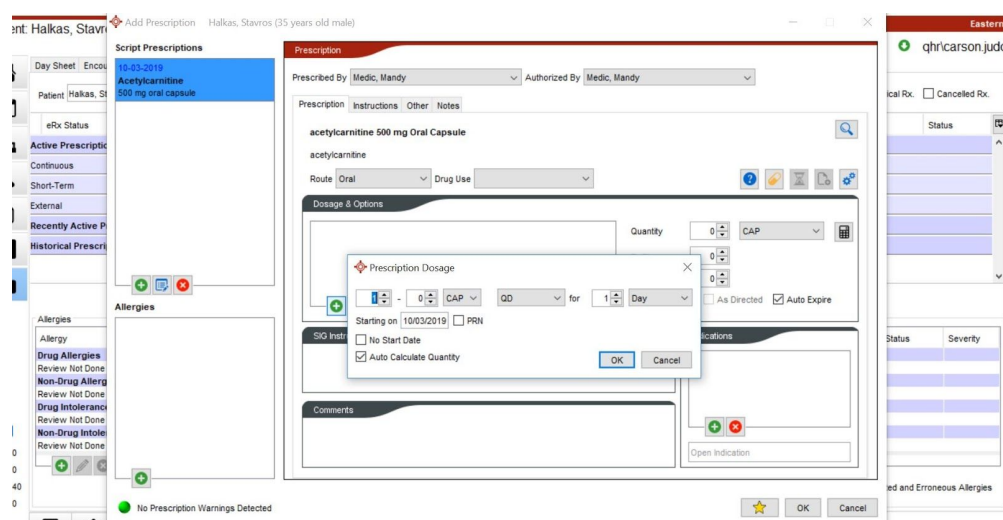
The first method corresponds with the List Drugs portion of the level 1 DFD. It takes a prefix and returns all drugs whose name starts with that prefix. For example, the method could be used to find all drugs that start with 'a'.

The *queryAllergyInteractionOfDrug* method corresponds to the Drug-Allergy portion of the DFD. It takes a drug and patient information given by Accuro and returns the allergies that could negatively interact with the drug.

The rest of the methods all correspond to the Drug-Allergy, Drug-Food, and Drug-Drug portions of the DFD. The last two methods find any negative interactions that a given drug has with food and other drugs. Note that the *queryAllergyInteractionOfDrug* is the only method that takes information stored in Accuro. The rest of the methods return results based entirely on First Databank.

## User Interface

The system does not require a user interface, but the system should be able to connect to any user interface. The following image shows the current user interface used in Accuro:



What it means to connect to any user interface is it needs to be able to follow the following DFD diagrams above. That is, the system needs a method to look up drug with a

particular dosage, route, and use. It should also have a method to find any complications with the drug such as issues with allergies, food, and other drugs. These operations are all included in the top-level API (above). Our main goal is to implement these operations in an efficient manner.

## Query Optimization

Currently, there is a working system in Accuro that does everything that our finished product would do. However, the system in Accuro is inefficient; querying large result sets can take over 30 seconds, and sometimes the system crashes because the result is too big. One such example is to query all drugs that start with the letter 'a'. Thus, one of our main challenges is to make sure that the queries involved in our system are executed in an efficient manner. The following are techniques that will be used, when possible, in order to make the queries more efficient..

### Parallelism

Inside any of the methods in the API, separate queries that do not depend on one another will be done using separate threads when available. For example, in the Drug-to-Food module shown below, the tables 'Clinical Formulation Id' and 'Drug-to-Food Interaction' do not have any attributes in common. Therefore, they could be executed in parallel when searching for types of foods that interact badly with a drug. Further, all of the methods in the top-level API will be parallel-safe. That is, Accuro can run any of the methods in the API at the same time.

### Reusing Results

Many of the queries that will take place in our system will query the same tables. For example, the Drug-Drug Interaction Master Table in the Drug to Drug Module below contains identifying information that will be used by other queries involved in collecting information about drug-to-drug interaction. In cases like this, we will be reusing results and not parallel processing.

### Pagination

The idea behind Pagination is that a certain number of specified rows are returned from the database instead of the entirety of a query's result set. In our case, the result set can be very large considering how many drugs are in the database (think about searching for all drugs with an 'a' in it). It should be noted that the code involved still does a full table scan--ie reading all tables and gathering all the results of the query--regardless of the amount of rows to be returned. Intuitively this sounds like Pagination shouldn't be faster than doing the normal query, but only a subset of the result set is actually returned, which

the API can handle better than the full amount. Currently searching for the letter 'a' crashes the system as their query returns all results.

A Pagination query would look like this:

```
SELECT firstName, lastName, address, country FROM customers ORDER BY firstName DESC OFFSET 0 ROWS
FETCH NEXT 1000 ROWS ONLY;
```

Where OFFSET 0 ROWS means to get rid of 0 tuples, and FETCH NEXT 1000 ROWS ONLY means to return only 1,000 rows after the OFFSET. To continuously give more results, the database would just be queried again with different values for rows returned--aka the next set of rows would need OFFSET 1000 ROWS FETCH NEXT 1000 ROWS.

There are other ways to tackle this problem, so it should be said that we will only use Pagination if it works better for the API.

## Manipulating Relational Algebra

Another idea is to manipulate the form of a query. In COSC404, we learn about query optimization heuristics. They aren't guaranteed to make the query faster, but the ideas behind them attempt to make the intermediate steps in queries as small as possible (which end up being faster some of the time). Since our tables are so large--aka our intermediate steps are also large--we want to try these techniques to see if they will help.

The rules are as follows:

1. Deconstruct conjunctive selections into a sequence
2. Move selection operations down the query tree for the earliest possible execution
  - a. Reduces the amount of tuples in joins
3. Replace Cartesian product operations that are followed by a selection condition by join operations
  - a. Since a join results in less tuple reads (cartesian products produces much larger tables to read)
4. Execute first selection and join operations that will produce the smallest relations
  - a. The first selection and join with the smallest relation will give you a smaller amount of tuples for the next selection

To start, let's show an example of selecting all first names from a table Names:

```
SELECT firstName FROM Names
```

We can use relational algebra to represent this type of query which we later optimize. The relational algebra for the above would be:

$$\Pi_{firstName}(Names)$$

To show that we can optimize certain queries, consider:

```
SELECT name, major
FROM Student, Enrolled, Class, Faculty
WHERE facultyId = fid AND studentId = sid AND className = cname AND fname = 'Jones'
```

Which gives the following relational algebra:

$$\Pi_{name,major}(\sigma_{facultyId=fid \text{ AND } studentId=sid \text{ AND } className=cname \text{ AND } fname='Jones'}(Student \bowtie Enrolled \bowtie Class \bowtie Faculty))$$

After using our rules, we end up with:

$$\Pi_{name,major}(\sigma_{studentId=sid}(Student \bowtie (\sigma_{className=cname})(Enrolled \bowtie (\sigma_{facultyId=fid})(Class \bowtie (\sigma_{fname='Jones'})(Faculty))))))$$

Although it may look like a more difficult query, the amount of tuple reads from the database is minimized. Thus, it can be faster when querying a large database with many results.

## Technical Specifications

This section describes what technologies our system will use. The client has said that any open-source software is allowed to be used in order to improve the final product.

### Programming Language

The client wants the Accuro API to be programmed in Java 8. The reason for this is Accuro, the software owned by our client, is built using Java 8. Thus, the system needs to be written in Java 8 in order to be compatible.

### Dependency Management Software

The system that will be built uses multiple dependencies including a database driver, an object orientated querying library, and testing software. Without dependency management software, every jar file would have to be downloaded and added to the build path of the project, and whenever the dependency changes, the dependency has to be redownloaded and added to the build path again.

To avoid this, the system will use Gradle as the dependency management software. Gradle allows users to simply add dependencies by adding them to a gradle.build file. If



the dependency changes, all one has to do is change the version number in the gradle.build file.

The main reason for using Gradle is that the client uses Gradle, so it makes sense for compatibility. However, there are alternatives like Maven and Ant, but these alternatives are nowhere near as large as Gradle. Gradle allows you to include dependencies from either Maven or Ant, and Gradle is also compatible with other languages such as JavaScript and C++ . Thus, Gradle is being used because QHR uses Gradle, and because it has wider applications than the alternatives.

## Database Software

The main goal of this project is to create an API for querying the database, so, a database driver is needed to query the database. The database management system used by the client is SQL Server, so a SQL server driver is needed to be compatible.

Along with the database driver, the JOOQ open source library will be used to construct queries. The main reason we are using JOOQ is because Accuro's current system is spaghetti code. As a result, it is very hard to maintain. We are using JOOQ to construct cleaner and more maintainable queries. JOOQ constructs queries with methods rather than string concatenations. The query

```
SELECT name FROM drugs
```

in JOOQ would look like

```
DSL.using(SQLDialect.SQLSERVER)
    .select("name")
    .from("Drugs")
    .getSQL();
```

which will make large queries more readable.

There are two major alternatives: one of them is JDBC. To construct a query in JDBC, we have to use string concatenation or a string builder. This is fine for small queries, but our system will involve very large queries. This would make it very hard to read and edit the queries later on, especially if we are combining queries from multiple places in the implementation. This is also one of the big problems we are trying to solve as their current system is full of spaghetti code. Thus, we will not use JDBC for constructing queries.

The other alternative to JOOQ is QueryDSL. QueryDSL has many of the same benefits as JOOQ as they both use methods to construct queries. However, JOOQ has many more features than QueryDSL. For example, JOOQ has a feature that takes in a database schema and generates Java classes to interact with that database. While we won't

be using all of JOOQs features, it gives our client more options in the future than if we use QueryDSL.

Another downside to QueryDSL is that it's not being as actively developed as JOOQ. As of writing this, the QueryDSL github page has two commits in the last nine months. Compare this with JOOQ which has two paid levels and hundreds of commits over the last year. Therefore it makes more sense to use JOOQ because it will be compatible for a longer period of time.

## Testing Software

In order to make sure that our API is working correctly, we will be using TestNG for unit and integration testing, and we will use JMockit to mock external dependencies, as per QHR.

TestNG is a framework that is very similar to JUnit; it was inspired off of JUnit. However, the main reason we are using TestNG is because it has everything JUnit has plus more. For example, TestNG supports dependency testing while JUnit does not. This can be seen in the following code:

```
@Test
public void testMethod1() {
    System.out.println("Method 1");
}

@Test(dependsOnMethods = {"testMethod1"})
public void testMethod2() {
    System.out.println("Method 2");
}
```

TestNG allows you to put an annotation at the top of tests that show they are dependent on other tests. This means if *testMethod1* fails, then *testMethod2* will be ignored. This allows for better integration testing.

Another alternative for unit and integration testing is the framework Spock. Spock is an all in one solution. It allows you to do unit testing, integration testing, and mock objects. However, Spock does not have dependency testing like TestNG, and Spock tests are written in the language Groovy, which is a language built off of Java. This means anyone reading our code will have to understand Groovy as well as Java, and we would need a different compiler for just our tests. Thus, we are using TestNG because it has the most features and because it does not require much overhead to include it in our system.






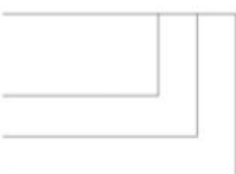
Since our system is heavily dependent on database queries, we need a mocking framework in order to test our units of code without the outside influences like the state of the First Databank database. To do this, we will be using the mocking framework JMockit. The only reason we use this over alternatives like Mockito and EasyMock is

because our client is currently switching over to JMockit from Mockito. Also, JMockit supports everything that both Mockito and JMockit supports, so there is no huge benefit to using EasyMock or Mockito over JMockit.

## Entity Relation Diagrams (ERD)

The following sections describe the main components of the database that our API will be querying. The database has many sections, but there are only three sections that we care about: the Drug-Drug Interaction Module, Drug-Allergy Interaction Module, and the Drug-Food Interaction Module. We will use entity relation diagrams to show how tables in these modules interact.

Entity relation diagrams are a top-down approach to database design that models the data as entities, attributes, and relationships. These relationships are described using various lines, which descriptions can be found in the image on the right hand side. These lines describe interactions between entities such as one to one and one to many. Where the key that determines the interaction between those entities is used as the description of the relationship.

ERD Line Symbol	Meaning
	Exactly one (required)
	Zero or one (optional)
	One or multiple (required)
	Zero, one, or multiple (optional)
	More than one (required)
	Recursive loop

### Identifiers and Attributes Top-Level ERD

The identifiers and attributes top-level entity relation diagram is broken into three sections: packaged products (green) generic formulation/ingredient data (yellow), and miscellaneous therapeutic classification data (purple). Clinical Formulation ID Table sends GCN\_SEQNO to the IDDF Canada Drug Product Table to obtain the description of federal regulatory and drug product information for a particular drug. From there the IDDF Canada Drug Product Table uses ILBLRID to obtain manufacturer or label information from the IDDF Canada Labeler (MFG) Identifier Description Table. Clinical Formulation ID Table can also use the GCN\_SEQNO, TC and GTC to obtain different therapeutic data about a particular drug. From Clinical Formulation ID Table, we obtain the drug category, strength, dosages, routes, ingredients, and attributes.

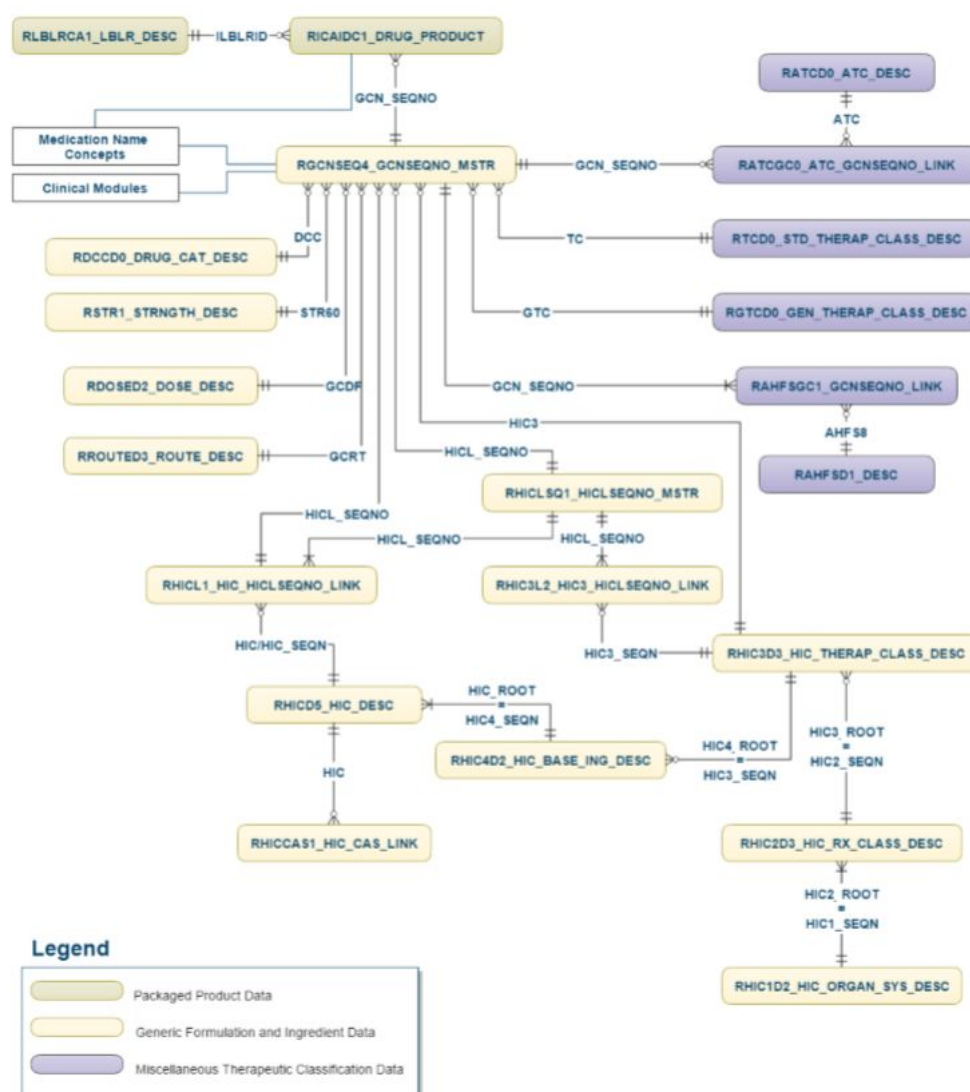


Table Name	English Name	Description
RGCNSEQ4	Clinical Formulation ID Table	Provides attributes of a Clinical Formulation ID (GCN_SEQNO) drug formulation
RICAIDC1	IDDF Canada Drug Product Table	Describes federal regulatory and drug product information
RLBLRCA1	IDDF Canada Labeler (MFG) Identifier Description Table	Provides manufacturer/labeler information
RATCD0	ATC Description Table	Relates an ATC to its text description
RATCGC0	GCN_SEQNO ATC Relation Table	Links a clinical formulation to its anatomic therapeutic classification

RTCD0	Standard Therapeutic Class Description Table	Relates the Standard Therapeutic Class Code to its text description
RGTC0	Generic Therapeutic Class Description Table	Relates the Generic Therapeutic Class Code to its text description
RAHFSGC1	GCN_SEQNO AHFS Code Relation Table	Links a clinical formulation to its therapeutic classification
RAHFSD1	AHFS Description Table	Relates the AHFS Therapeutic Class to its text description
RDCCD0	Drug Category Description Table	Relates the Drug Category Code to its text description
RSTR1	Drug Strength Component Table	Provides attributes of a drug's potency
RDOSED2	Dosage Form Description Table	Relates the various dosage form codes to their descriptions/abbreviations
RROUTED3	Route of Administration Description Table	Relates the various routes of administration codes to their descriptions/abbreviations
RHICLSQ1	Ingredient List Identifier Description Table	Relates the HICL_SEQNO to the generic drug ingredient list
RHICL1	HICL_SEQNO-HIC Relation Table	Links individual ingredients to an ingredient list
RHIC3L2	HICL_SEQNO-HIC3 Relation Table	Links an ingredient list to a parent therapeutic class
RHICD5	Hierarchical Ingredient Code Description Table	Relates the HIC_SEQN to its text description and provides attributes of that relationship
RHIC3D3	Hierarchical Ingredient Code Specific Therapeutic Class Table	Provides attributes of a specific therapeutic class
RHIC4D2	Hierarchical Base Ingredient Code Table	Provides attributes of a base ingredient
RHICCAS1	HIC-Chemical Abstracts Service Registry Number Relation Table	Links an active ingredient to its chemical ingredient
RHIC2D3	Hierarchical Ingredient Code Pharmacological Class Table	Provides attributes of a pharmacological class

The drug to drug interaction module ERD shows how drugs interact with one another. All of routed medication, clinical effects, and drug-drug Interaction Monograph link to the drug-drug master table. The drug-drug master table links to the specific agent causing the interaction. The severity level links to both drug-drug and Monograph master tables as well as the exception table. The reference category links to the monograph text table which links to both the drug-drug and monograph master tables. The exception table links to the clinical formulation table which links to the drug-drug master table. The display action table links to the exception table as well as the monograph master table.

[illegible]

Table Name	English Name	Description
RDDIMRM0	Drug to Drug Interaction Module Routed Medication Table	Links a routed medication to a drug interaction
RADIMMA5	Drug-Drug Interaction Master Table	Provides attributes of a drug interaction
RADIMMO5	Drug-Drug Interaction Monograph Text Table	Provides the text for the professional drug interaction monograph.
RDDIMAG0	Drug-Drug Interaction Agent Description Table	Relates the Drug-Drug Expanded Interaction Code to the specific agents involved
RADIMSL1	Drug-Drug Interaction Severity Levels Table	Relates the Drug-Drug Interaction Severity Level to its text description.
RDDIMRD0	Drug-Drug Reference Category Description Table	Relates the Drug-Drug Interaction Reference Category Line Identifier to its text description
RADIMMO5	Drug-Drug Interaction Monograph Text Table	Provides the text for the professional drug interaction monograph
RADIGE0	Drug-Drug Interaction Clinical Formulation Exception Table	Drug-drug interaction exceptions
RGCNSEQ4	Clinical Formulation ID Table	Links a unique drug formulation to a slightly broader clinical formulation
RADIDA0	Drug-Drug Interaction Display Action Table	Provides the recommended alerting level for interactions
RADIMEF0	Clinical Effects Description Table	Clinical Effect text description
RADIMIE4	Drug-Drug Interaction-Clinical Effects Relation Table	Links Clinical Effect text description to Clinical Effect code
RDDIMAG0	Drug-Drug Interaction Agent Description Table	Relates the Drug-Drug Expanded Interaction Code to the specific agents involved in the interaction.
RADIMGC4	GCN_SEQNO Drug-Drug Interaction Code Relation	Links drugs or drug classes to the drug interactions in which they participate.

RMIRMID2	MED Routed Medication Table	Provides attributes of the routed medication.
----------	-----------------------------	---

## Drug-Allergy Interaction Module ERD

The clinical formulation master table links individual ingredients to an ingredient list via two different tables, one of which is done for allergy screening purposes. The second one goes to the Hierarchical Ingredient Code Description Table which uses HIC\_SEQN to link the specific allergen group code to the cross-sensitive allergen group code. The cross-sensitive allergen group code can then be used in numerous tables to obtain a text description of the allergy, inactive indicator, status code and allergen history.

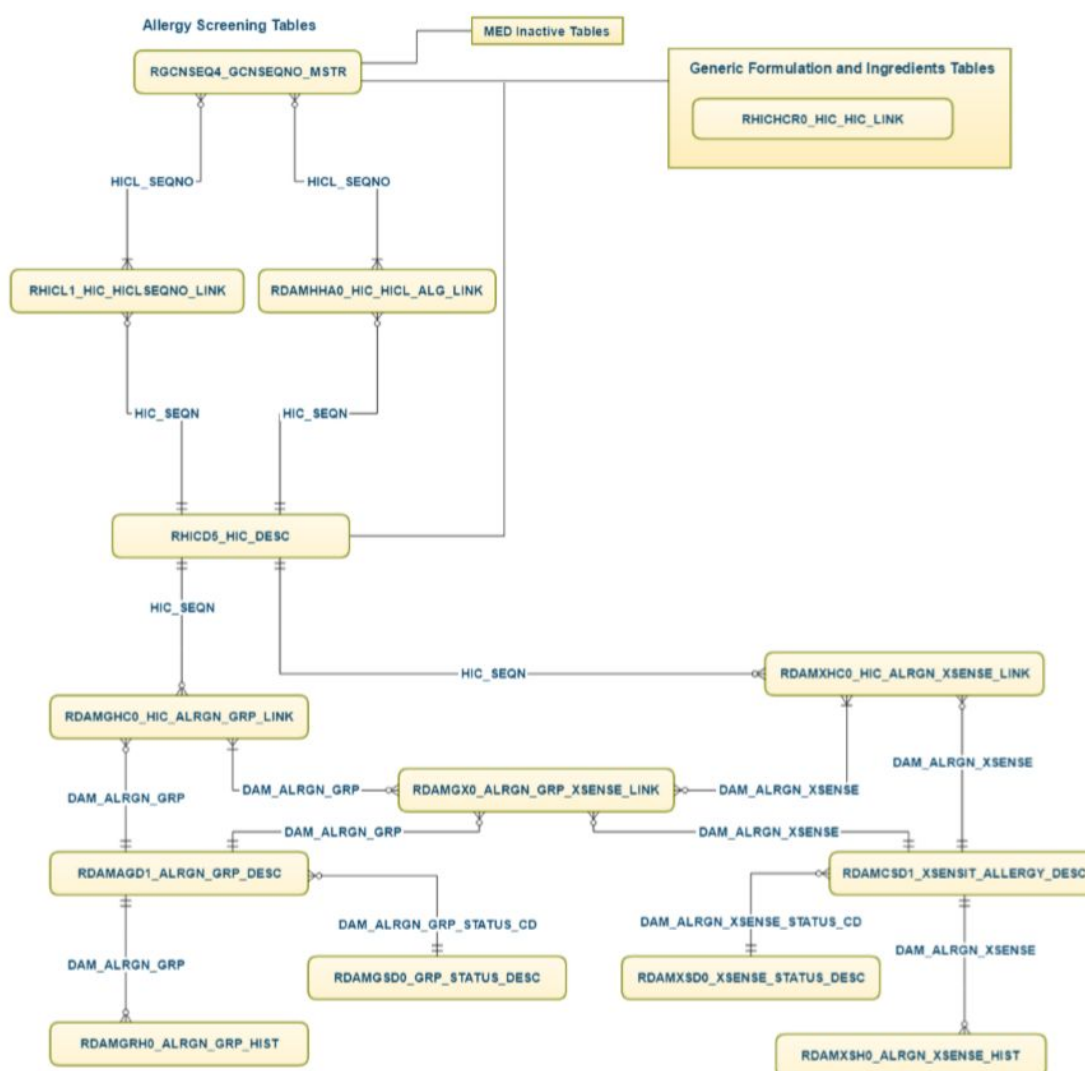


Table Name	English Name	Description
RHICHCR0	HIC_SEQN-HIC_SEQN Link Table	Links related Ingredients



RGCNSEQ4	Clinical Formulation ID Table	Provides attributes of a Clinical Formulation ID (GCN_SEQNO) drug formulation
RHICL1	HICL_SEQNO-HIC Relation Table	Links individual ingredients to an ingredient list
RDAMHHA0	Drug Allergy Screening HICL_SEQNO HIC Relation Table	Links individual ingredients to an ingredient list for the purpose of allergy screening.
RHICD5	Hierarchical Ingredient Code Description Table	Relates the HIC_SEQN to its text description and provides attributes of that relationship
RDAMGHC0	DAM Ingredient Allergen Group Link Table	Links an ingredient to one or more Specific Allergen Group Codes
RDAMGX0	DAM Specific Allergen Group Code Status Code Description Table	Links the Specific Allergen Group Code to the Cross-Sensitive Allergen Group Code
RDAMAGD1	DAM Specific Allergen Group Code Description Table	Provides the Specific Allergen Group Code's text description, Possibly Inactive Indicator, and Status code
RDAMGRH0	DAM Specific Allergen Group Code History Table	Tracks the replacement history of the Specific Allergen Group Code
RDAMGSD0	DAM Specific Allergen Group Code Status Code Description Table	Provides the description of the Allergen Group Status Code
RDAMXHC0	DAM Ingredient Cross-Sensitivity Link Table	Links an ingredient to one or more Cross-Sensitive Allergen Group Codes
RDAMCSD1	DAM Cross-Sensitive Allergen Group Code Description Table	Provides the Cross-Sensitive Allergen Group Code's text description, Possibly Inactive Indicator, and Status code
RDAMXSD0	DAM Cross-Sensitive Allergen Group Code Status Code Description Table	Provides the description of the Cross-Sensitive Allergen
RDAMXSH0	DAM Cross-Sensitive Allergen Group Code History Table	Tracks the replacement history of the Cross-Sensitive Allergen Group Code

## Drug-Food Interaction Module ERD

This ERD describes how to get descriptions of a drug-to-food interaction from two tables: clinical formulation drugs and MED routed medication. The Drug-Food Interaction Master Table (RDFIMMA0) is where the drug-to-food interaction is obtained. Drug-Food Interaction Monograph Text Table takes the FDCDE from the master table and provides a description of the particular drug to food interaction. Drug-Food Code Relation Table takes the GCN\_SEQNO from Clinical Formulation ID Table, which is a table of clinical formulation drugs and converts it to FDCDE to obtain all the drug-food interactions. The drug food interaction module routed medication table takes the ROUTED\_MED\_ID from the MED Routed Medication Table, which is a table of MED routed medication and converts it to FDCDE to obtain all the drug-food interactions.

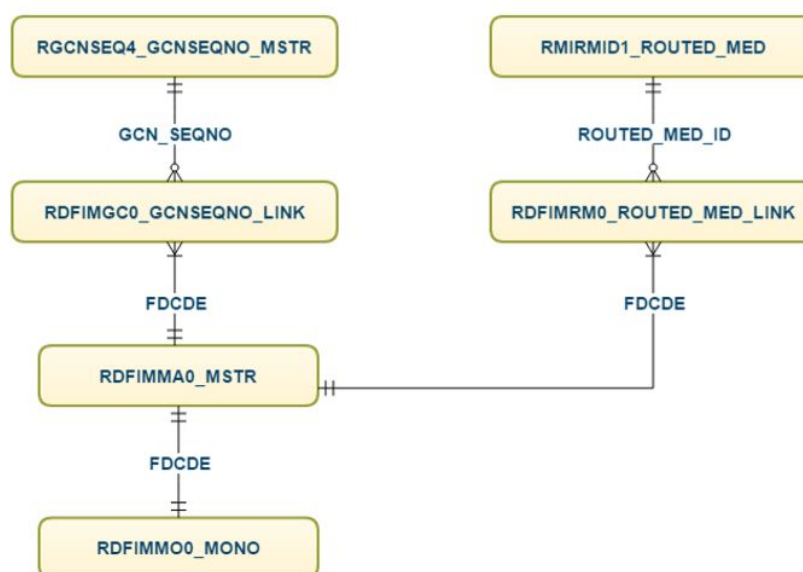


Table Name	English Name	Description
RGCNSEQ4	Clinical Formulation ID Table	Provides attributes of a Clinical Formulation ID
RDFIMGC0	GCN_SEQNO - Drug-Food Code Relation Table	Links clinical formulations to food interactions
RDFIMMA0	Drug-Food Interaction Master Table	Provides attributes of the drug-food interaction
RDFIMMO0	Drug-Food Interaction Monograph Text Table	Provides the text for the drug-food interaction professional monograph
RMIRMID1	MED Routed Medication Table	All the different routes to take a drug

RDFIMRM0	Drug food interaction module Routed Medication Table	Links a routed medication to a drug-food interaction
----------	---	--