**Clinical Query 2 (CQ2)**

Version 1.0.1

September 8, 2021

Created by:

Griffin M Weber, MD, PhD

Nick Benik

Nick Brown

Ken Huling

Yun William Yu, PhD

Please direct questions to:

Griffin M Weber, MD, PhD

Associate Professor of Medicine and Biomedical Informatics

Beth Israel Deaconess Medical Center

Harvard Medical School

[weber@hms.harvard.edu](mailto:weber@hms.harvard.edu)

Table of Contents

[Introduction 2](#_Toc34172828)

[Overview 2](#_Toc34172829)

[Features 2](#_Toc34172830)

[Use Cases 2](#_Toc34172831)

[Installing CQ2 3](#_Toc34172832)

[Updating CQ2 Data 6](#_Toc34172833)

[Customizations 7](#_Toc34172834)

[Using Multiple Databases 7](#_Toc34172835)

[Custom Query Breakdowns 8](#_Toc34172836)

[Custom User Authentication and Registration 8](#_Toc34172837)

[Using Sketches in Queries 9](#_Toc34172838)

[CQ2 Architecture 10](#_Toc34172839)

[Functional Comparison of CQ2 and i2b2 12](#_Toc34172840)

[Version History 14](#_Toc34172841)

# Introduction

## Overview

Clinical Query 2 (CQ2) was initially developed at Beth Israel Deaconess Medical Center (BIDMC) and Harvard Medical School (HMS) in 2008 to serve as BIDMC’s i2b2 system. Back then, BIDMC did not support Java applications. We therefore rewrote the Java-based application layer of i2b2 as Microsoft SQL Server stored procedures. This allowed us to replace all the i2b2 Java code with a single web service proxy, which simply passes an i2b2 XML request message to a database stored procedure and returns the i2b2 XML response message that comes back from the database.

In addition to replacing the Java code, while i2b2 uses several databases (one for each cell), we merged them all into a single database, using different schemas for each cell. CQ2 is this combined database, the stored procedures, and the web service proxy. CQ2 retains the same “star schema” to store clinical data as i2b2, it uses i2b2 ontologies, and it has the same API message format as i2b2. As new versions of i2b2 are released, we update CQ2 to support (nearly all of) the new features. As a result, CQ2 works with any i2b2 client software and appears like any other i2b2 node in federated SHRINE networks.

## Features

By consolidating all the data and code into a single database, CQ2 has several advantages over i2b2: (1) It is easier to install and support since there are fewer components to manage, fewer database accounts to configure, fewer web applications to setup, etc. (2) It minimizes network traffic and improves performance since all the processing occurs within a single database. (3) Only a tiny amount of storage and resources are needed on the application/web server. (4) The code can be greatly simplified—all the CQ2 stored procedures total less than 7000 lines of code and are under 400 KB. In contrast, the i2b2 Java code is nearly 100 MB.

A unique feature of CQ2 is its query engine. Optimized indexes and the use of stored procedures make queries generally run faster than i2b2. CQ2 has two optional features to greatly improve performance (orders of magnitude faster), which you can enable or disable. The first precomputes a set of aggregate tables derived from the i2b2 fact and dimension tables. Many queries can be run on these much smaller aggregate tables rather than on the original data. The second uses a class of computational algorithms that compress the original data into small “probabilistic sketches”. The sketches can be used to obtain a very fast and accurate estimate of any query. Even with millions of patients and billions of data facts, sketch-based queries typically require less than a second to return an estimate within 10% of the actual count, and a few seconds to be within 1%. When running a query, the user can choose whether to optimize for speed or accuracy, or to continue running the query until the actual exact result is found.

## Use Cases

CQ2 has two primary use cases. The first is serving as an institution’s “enterprise” i2b2 system. Its performance optimizations enable it to scale to massive sizes (100+ million patients). The second is as an i2b2 node in federated networks. The fact that it is just a single database and one web service proxy file makes it easy to “spin up” new instances to plug into these networks. However, because CQ2 does not include the i2b2 Java layer, it does not support i2b2 Java plugins that have been developed for specialized use cases, such as image processing or genomics pipelines.

# Installing CQ2

CQ2 has three parts: (1) a database; (2) a web service proxy, which gets installed on a web server; and, (3) the i2b2 web client, which is also installed on a web server. The CQ2 software consists of the database code and the web service proxy. You must separately download the i2b2 web client from i2b2.org. Follow the instructions below to install CQ2 and a demo ontology and dataset.

1. Create a new SQL Server database. (The database name can be anything you want.)
2. Create the database objects in the new database.
   1. Create schemas for each cell in the new database. Run each statement, one line at a time.

CREATE SCHEMA CRC AUTHORIZATION DBO;

CREATE SCHEMA HIVE AUTHORIZATION DBO;

CREATE SCHEMA ONT AUTHORIZATION DBO;

CREATE SCHEMA PM AUTHORIZATION DBO;

CREATE SCHEMA WORK AUTHORIZATION DBO;

* 1. Run Database\CreateTablesViews.sql to create all the tables and views used by CQ2. They will be placed in the CRC, HIVE, ONT, PM, and WORK schemas.
  2. Run Database\CreateFunctionsProcedures.sql to create all the functions and stored procedures used by CQ2. They will be placed in the CRC, HIVE, ONT, PM, and WORK schemas.
  3. Optionally run Database\CreateOptionalObjects.sql. This creates a table and several stored procedures in the DBO schema. These are not normally used by CQ2. They are optional objects that are helpful for debugging or making a copy of a CQ2 instance.

1. Load the lookup, project, and ontology table data.
   1. Run Database\Demodata\DefaultData.sql to load data into various lookup and parameter tables. This script creates a demo project and a few example user accounts. Edit these as needed.
   2. Run Database\Demodata\DemoOntologyPart1.sql and Database\Demodata\DemoOntologyPart2.sql to install a subset of the standard i2b2 demo ontology. These scripts might take several minutes to run. Replace these with your own ontology.
   3. Load the CRC.CONCEPT\_DIMENSION table using the following query. Again, customize this based on your ontology.

INSERT INTO CRC.CONCEPT\_DIMENSION

(CONCEPT\_PATH, CONCEPT\_CD, NAME\_CHAR, SOURCESYSTEM\_CD)

SELECT C\_FULLNAME, C\_BASECODE, C\_NAME, SOURCESYSTEM\_CD

FROM ONT.I2B2

WHERE C\_SYNONYM\_CD='N' AND C\_FACTTABLECOLUMN='concept\_cd'

AND C\_TABLENAME='concept\_dimension' AND C\_COLUMNNAME='concept\_path'

AND M\_APPLIED\_PATH='@' AND C\_BASECODE<>''

1. Load the clinical data (fact and dimension tables).
2. Execute the first of six data update stored procedures:

EXEC CRC.uspUpdateStep1CreateDataTables;

1. Run Database\Demodata\DemoDataPatientVisit.sql, Database\Demodata\DemoDataModifierProvider.sql, and Database\Demodata\DemoDataFact.sql to load a subset of the standard i2b2 demo dataset. (Normally, instead of loading data this way, you would execute the CRC.uspUpdateStep2LoadDataTables procedure after replacing its code with your custom ETL process.)
2. Run the following query to load the new concept dimension table. (This step would also be replaced by your customized CRC.uspUpdateStep2LoadDataTables procedure.)

INSERT INTO CRC.CONCEPT\_DIMENSION\_NEW

SELECT \* FROM CRC.CONCEPT\_DIMENSION

1. Execute the remaining data update stored procedures:

EXEC CRC.uspUpdateStep3IndexDataTables;

EXEC CRC.uspUpdateStep4CreateCQ2Tables;

EXEC CRC.uspUpdateStep5SwapTables;

EXEC CRC.uspUpdateStep6DropOldTables;

1. Run the following queries to load the patient and encounter mapping tables.

INSERT INTO CRC.PATIENT\_MAPPING

SELECT CAST(PATIENT\_NUM AS VARCHAR(50)), 'HIVE',

PATIENT\_NUM, 'A', 'demo',

NULL, NULL, NULL, NULL, NULL, NULL

FROM CRC.PATIENT\_DIMENSION

INSERT INTO CRC.ENCOUNTER\_MAPPING

SELECT CAST(ENCOUNTER\_NUM AS VARCHAR(50)), 'HIVE',

'demo', ENCOUNTER\_NUM,

CAST(PATIENT\_NUM AS VARCHAR(50)), 'HIVE', 'A',

NULL, NULL, NULL, NULL, NULL, NULL

FROM CRC.VISIT\_DIMENSION

1. Create a database account that the web services will use. Database\CreateAccount.sql contains SQL queries to create an account with the minimum permissions to run CQ2. This script will likely need customization (as described in script comments) for production CQ2 instances.
2. Setup the CQ2 web service. CQ2 contains both ASP classic and PHP versions of the web service.  
   ASP Classic:
   1. Create a directory on a Windows IIS web server for the CQ2 web service.
   2. Make sure ASP Classic is enabled.
   3. Copy the file WebService\CQ2ServiceProxy.asp into the directory. We suggest renaming it default.asp.
   4. Edit the connection string near the top of the file to point to your CQ2 database.
   5. Update the URLs in the database table PM.PM\_CELL\_DATA to match where you placed the file. Note that the URLs should be in the form “path/?/service/”. For example, “http://localhost/i2b2/services/?/QueryToolService/”. Do not change the service names (the part after the question mark). Only edit the path (the part before the question mark) to match the location of the file. The same file (path) can be used for all cells. Note that if you renamed the file “default.asp”, the URL can be in the form “path/?/service” rather than “path/default.asp?/service”.

PHP:

1. Create a directory on a Windows IIS / Apache web server for the CQ2 web service.
2. Install PHP on your web server. CQ2 is tested with PHP 8.x.x.
3. Copy the file WebService\CQ2ServiceProxy.php into the directory.
4. Edit the connection string near the top of the file to point to your CQ2 database.
5. Update the URLs in the database table PM.PM\_CELL\_DATA to match where you placed the file. Note that the URLs should be in the form “path/service/”. For example, “http://localhost/i2b2/services/CQ2ServiceProxy.php/QueryToolService/”. Do not change the service names. Only edit the path to match the location of the file. The same file (path) can be used for all cells
6. Setup the i2b2 web client.
   1. Create a directory on a Windows IIS web server for the i2b2 web client.
   2. Download the i2b2 web client source code from <https://www.i2b2.org/software>. Version 1.7.12 or newer is required for CQ2.
   3. Unzip and copy the i2b2 web client code into the new directory.
   4. Edit the i2b2\_config\_data.js file so that the urlCellPM value is the URL of your PM web service. The URL should be in the form “path/?/PMService/getServices” for the ASP classic proxy or “path/PMService/getServices” for the php proxy, where the path is the URL of your CQ2 service proxy.
   5. Setup the i2b2 web client proxy file. You can either use the index.php proxy that comes with the i2b2 web client (recommended) or the WebClientProxy.asp file that comes with CQ2.
      * To use index.php, first make sure that PHP is enabled for the web client directory. Then, edit the index.php file so that $pmURL points to the PM web service.
      * To use the CQ2 WebClientProxy.asp file, copy it to the web client directory. Make sure ASP Classic is enabled. Edit the i2b2\_config\_data.js file, changing the urlProxy value to “WebClientProxy.asp”.

# Updating CQ2 Data

Updating data in CQ2 generally uses these steps:

1. A set of new data tables are created. The suffix “\_NEW” is added to the names of the new tables. For example, CRC.PATIENT\_DIMENSION\_NEW.
2. Updated data are loaded into these new tables.
3. Indexes are added to the new tables.
4. CQ2 generates a set of new derived aggregate tables. The names of these tables have the prefix “CQ2\_”.
5. The original tables are “swapped” for the new tables. This is done by first adding the suffix “\_OLD” to the original data tables and derived tables. For example, CRC.PATIENT\_DIMENSION is renamed CRC.PATIENT\_DIMENSION\_OLD. Then, the “\_NEW” is removed from the new tables.
6. The old tables are dropped.

\*\*\* You will need to customize these steps in several ways:

All steps can be run by executing the procedure CRC.uspUpdateAllSteps, which in turn calls six other procedures corresponding to the six steps (CRC.uspUpdateStepN). However, in practice, we suggest implementing a validation step that checks the new tables before swapping them with the old tables. This could be as simple as manually comparing the old and new row counts before running the last two steps. Also, we often delay dropping the old tables for a while to confirm that users aren’t having any problems after a data update. This allows us to revert back if needed.

You must customize the procedure CRC.uspUpdateStep2LoadDataTables for your environment. This procedure copies data into the new tables. Instead of embedding that code in this procedure, you could alternatively use SSIS or some other ETL process.

You might need to customize the other update procedures too. For example, by default they only update the fact table and the patient, visit, and concept dimension tables. If your provider or modifier data change frequently, you will need to add these to the workflow.

To generate the derived aggregate tables, CQ2 needs the ontology and the concept\_dimension data. It gets these by calling the views CRC.vwCQ2\_Ontology and CRC.vwCQ2\_ConceptDimension. By default, the CRC.vwCQ2\_Ontology view points to a table named ONT.i2b2. If your ontology is loaded into a different table or multiple tables, then you need to modify this view so that it selects the UNION ALL of these tables. Both views, by default, point to the “live” tables. If you are updating your ontology by placing the data in staging tables (e.g., ONT.i2b2\_NEW), then you should modify these views to point to those staging tables. In other words, the views must point to the ontology that corresponds to the data in the “\_NEW” tables before you execute CRC.uspUpdateStep4CreateCQ2Tables.

Instead of reloading the entire fact and dimension tables with each update, it is possible to just add new records to those tables. However, you still have to rebuild the derived aggregate tables. Create a set of “\_NEW” views that point to your fact and dimension tables so that the stored procedure CRC.uspUpdateStep4CreateCQ2Tables still works. For example, create a view named “CRC.OBSERVATION\_FACT\_NEW” that selects all rows from “CRC.OBSERVATION\_FACT”.

# Customizations

## Using Multiple Databases

Database views can provide a lot of flexibility on how you setup CQ2. Below are some examples:

By default, CQ2 uses a single database configuration.

|  |
| --- |
| **CQ2** |
| All tables and procedures |

A two database configuration places the patient data and derived aggregate tables in a separate database (CQ2\_Data). Views replace tables in the main CQ2 database. During data updates, the entire CQ2\_Data database can be swapped with a new database (rather than swapping tables).

|  |  |
| --- | --- |
| **CQ2** | **CQ2\_Data** |
| Replace the fact table and the patient and visit dimension tables with views that have the same name, but point to the actual tables containing the data in a separate database CQ2\_Data. | Place the following three tables in this database:   * CRC.PATIENT\_DIMENSION * CRC.VISIT\_DIMENSION * CRC.OBSERVATION\_FACT |
| Replace the CRC.CQ2\_\* tables with views with the same names that point to the corresponding tables in CQ2\_Data. | Place all derived CRC.CQ2\_\* tables in this database. |
|  | Place these two views in this database, pointing back to the CQ2 database:   * CRC.vwCQ2\_Ontology * CRC.vwCQ2\_ConceptDimension |
|  | Place the stored procedures CRC.uspUpdateStep\* in this database and execute them from here. |

A three database configuration additionally places patient and encounter sets in a separate database. This prevents a user running many large queries from “blowing up” the CQ2 database with all the storage needed to save those sets. The procedure CRC.uspRunQueryInstance must be modified so that patient and encounter sets are written to the CQ2\_Sets database. Example code for this is commented out in the procedure. The example code also illustrates limiting the total set site.

|  |  |  |
| --- | --- | --- |
| **CQ2** | **CQ2\_Data** | **CQ2\_Sets** |
| Create views for the PATIENT\_SET and PATIENT\_ENC collection tables. Modify the procedure CRC.uspRunQueryInstance to write to the tables in CQ2\_Sets. |  | Place the following two tables in this database:   * CRC.QT\_PATIENT\_SET\_COLLECTION * CRC.QT\_PATIENT\_ENC\_COLLECTION |
| Views that point to the data and CQ2 tables in CQ2\_Data | Data and CQ2 derived tables |  |

A four database configuration can split the CQ2\_Data database into one database with the fact and dimension tables and a second database with the derived tables. The former can be your existing i2b2 CRC database, if you have previously installed i2b2.

Many factors might go into your decision on how to setup CQ2. As a general rule of thumb, we recommend a single database if you have less than 1 million patients and 1 billion facts, and multiple databases if you have more than 10 million patients or more than 1 billion facts.

## Custom Query Breakdowns

CQ2 uses the CRC.QT\_BREAKDOWN\_PATH table to define how different types of query breakdowns should be performed. However, you can also hard-code the SQL for breakdowns in the procedure CRC.uspRunQueryInstanceBreakdown. There is example code in that procedure showing how to do this. The advantage is that you can run breakdowns that are not based on the ontology and/or optimize the breakdown queries for performance.

## Custom User Authentication and Registration

By default, CQ2 authenticates users by checking their USER\_ID and encrypted PASSWORD in the PM.PM\_USER\_DATA table. You can alternatively write your own custom authentication code (e.g., using Active Directory) by editing the service proxy file. There is a comment in the code in that file showing where to do this. Also, to use custom authentication, create a record for the user in the PM.PM\_USER\_DATA table, but leave the password blank (the empty string ''). To help with this, after you authenticate a user, pass the user\_id to the procedure PM.uspRegister. This will check if the user already has an account. If not, it will create the account and add the user to project. Modify this procedure as needed.

# Using Sketches in Queries

CQ2 uses “probabilistic sketches” to improve the performance of query, sometimes thousands of times faster than i2b2 for very large databases. The tradeoff is that the query result is an estimate, rather than the exact result. Though, typically the error is small. The benefits of sketches only start with about 100,000 patients and might not be noticeable even with 1 million patients if you have a fast database server. For very large databases, with 10+ million patients or 1+ billion facts, it might not be possible to run exact queries in a reasonable amount of time, and sketches are the only practical option.

CQ2 currently supports three types of query methods:

* “Accurate estimate” uses a large sketch, which has an error of about 1% (95% of queries will return a value within 1% of the true result) and usually takes a few seconds to run.
* “Fast estimate” uses a small sketch, which has an error of about 10%, but usually runs in less than a second.
* “No sampling” runs a normal i2b2 query, which returns the exact result, but is the slowest.

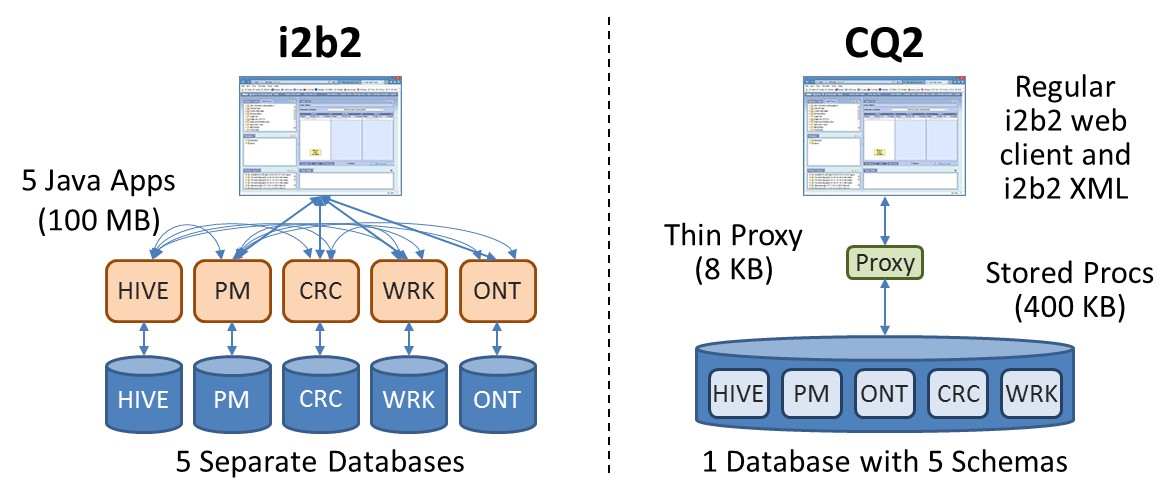
When users click to run a query in the i2b2 web client, they will see a drop-down menu to select the query method. This is a “hidden feature” in the web client that is not visible in standard i2b2. The web client shows the menu if there is a CRC parameter named QUERY\_OPTIONS\_XML in the PM.PM\_CELL\_PARAMS table. The value of this parameter is XML that defines which query methods should be visible to the user, and which order they should be listed. The method listed first is the default. We recommend using “Accurate estimate” as the default for large databases, and either removing this parameter or making “No sampling” the default for small databases.

When a query is run using sketches, the web client will display both the estimate result as well as the error (95% confidence interval). It will also indicate the query method used. Note that confidence intervals are calculated based on the specific query. For example, although “Accurate estimate” can have a 1% error, for some queries the estimated error can be much larger. For other queries, the CQ2 query engine might determine that “No sampling” would actually be faster than using sketches. In this case, it will return the exact result with no confidence interval.

If the user requests a patient or encounter set to be generated, and a sketch-based query method is selected, then only the sampled patients will be saved. A “No sampling” query must be run to save the full list of matching patients or encounters.

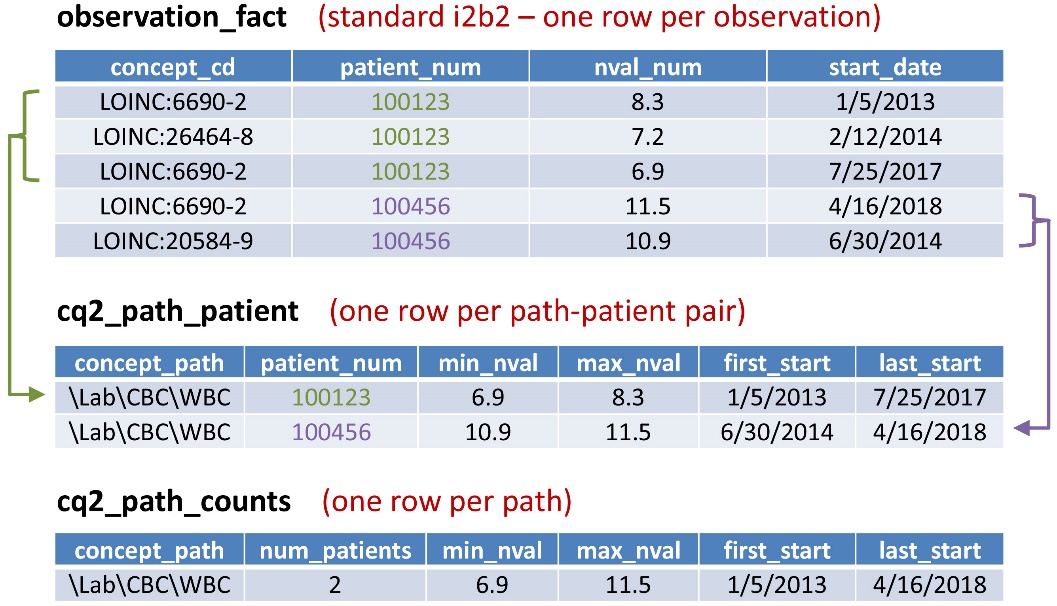
# CQ2 Architecture

CQ2 combines the five main i2b2 databases into a single database with schemas (Figure 1). The functionality of the Java application layer of i2b2 has been ported to stored procedures in the CQ2 database. A thin web service proxy enables the regular i2b2 web client to communicate with CQ2 using standard i2b2 XML messages.



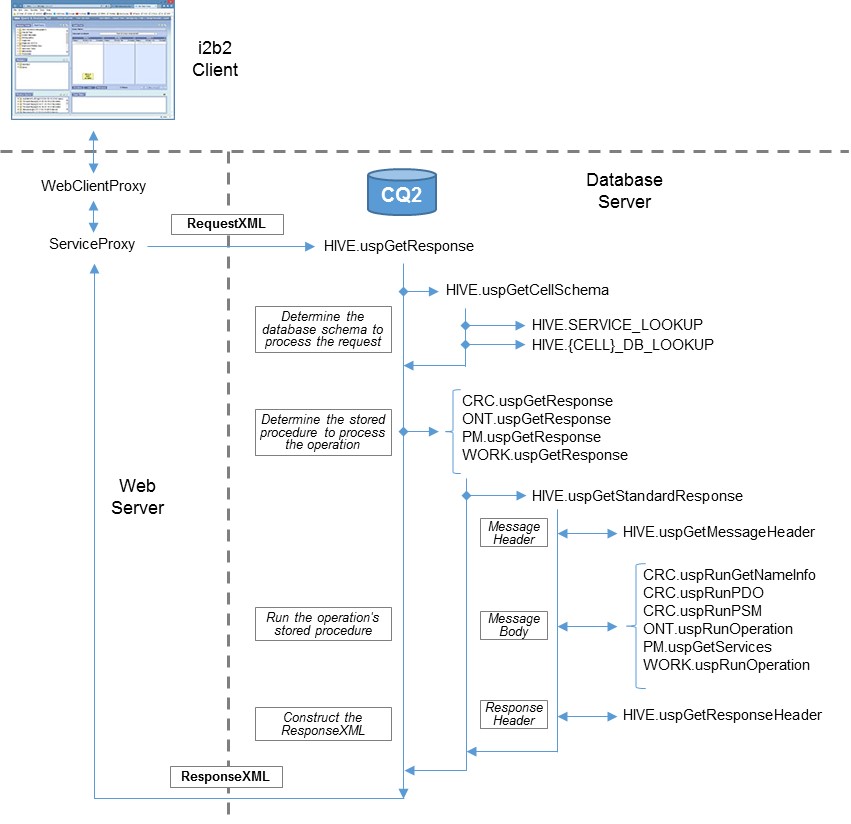
**Figure 1. Comparing the software architecture of i2b2 and CQ2.**

CQ2 precomputes aggregate “rollup” tables to improve the performance of its query engine (Figure 2). Most queries can use these rollup tables instead of the full i2b2 observation\_fact table. The example below shows the observation\_fact data for two patients with multiple white blood cell (WBC) count test results. The cq2\_path\_patient table combines these into a single WBC row per patient. The cq2\_path\_counts table further reduces this to a single row for WBC that indicates that two patients had the test, with values ranging from 6.9 to 11.5 between 1/5/2013 and 4/16/2018.



**Figure 2. An example of aggregate rollup tables in CQ2.**

The i2b2 web client sends a RequestXML message to a web client proxy on the web server (Figure 3). This sends the message to a service proxy (which is usually on the same web server), which in turn sends it to the HIVE.uspGetResponse database stored procedure. This calls HIVE.uspGetCellSchema, which parses the request to determine the project and service, and then uses the HIVE.SERVICE\_LOOKUP table to determine which HIVE.{CELL}\_DB\_LOOKUP table contains the name of the database schema that corresponds to the project and service. HIVE.uspGetResponse then calls a procedure named uspGetResponse in this schema, which determines the procedure to use to run the requested operation. That information is sent to HIVE.uspGetStandardResponse, which calls the procedure to run the operation, and then constructs the ResponseXML message. HIVE.uspGetResponse saves the RequestXML, ResponseXML, and additional details (e.g., request time, processing time, user’s IP address, etc.) to the HIVE.MessageLog table. It then returns the ResponseXML to the service proxy, which sends it back to the web client proxy, and finally back to the web client.



**Figure 3. Messaging flow in CQ2.**

# Functional Comparison of CQ2 and i2b2

i2b2 XML messages. CQ2 uses the same XML message format as i2b2, making it compatible with any “cells” that have been developed to work with an i2b2 “hive”, including SHRINE.

PM Cell. CQ2 supports multiple projects, users with different roles in projects, encrypted passwords and password tokens. CQ2 does not support any of the Admin services, which enable users to view and edit the PM tables through the web client. We currently have no plans to add this. We expect that users of CQ2 would prefer to manage these tables through database tools and ETL workflows.

Ontology Cell. CQ2 supports multiple ontology tables, managed through the table\_access table. However, the stored procedure that generates the derived aggregate data tables requires a single view that combines all ontology tables. We might modify this procedure in the future to read directly from the ontology tables, eliminating the need for the view.

Workplace Cell. The Workplace cell in CQ2 functions the same as i2b2.

CRC Cell (PDO). We believe CQ2 functions the same as i2b2. It requires the user to have a role that allows viewing patient level data. It uses the PATIENT\_MAPPING and ENCOUNTER\_MAPPING tables as designed in i2b2. It supports the different ways of specifying which data to return. However, due to limitations in the i2b2 documentation and few examples, it is difficult to know if the code has been ported correctly to CQ2.

CRC Cell (getNameInfo). Like i2b2, CQ2 enables users to paginate through previous queries and search for queries by name. However, we have found little documentation on this service. As a result, we are unable to determine how some of the search options in i2b2 are supposed to work.

CRC Cell (PSM). This is the aggregate query service.

* CQ2 supports loading, renaming, and deleting previous queries.
* CQ2 obfuscates query results depending on the user’s role(s). However, unlike i2b2, CQ2 does not lockout accounts if the user runs queries that return the same result multiple times. We plan to add this feature in the future.
* CQ2 supports all query types, with the following exceptions:
  + CQ2 currently does not have Full-Text search setup to query the OBSERVATION\_BLOB field. This is used in i2b2 to search notes and other types of data. We will probably add this feature in the future. (We have not yet had a use case for it.)
  + CQ2 supports nested queries (a previous query used as a “concept” in a new query) and temporal queries. However, it does not recognize the temporal constraints of nested previous queries. In other words, if a previous temporal query is used in a new query, then just the “population in which events occur” of the previous query, without the temporal constraints, are used to determine the list of matching patients. We might fix this in the future, but it is unclear how often users actually need nested temporal queries.
  + In CQ2, a patient set or an encounter set can be included in a query. However, individual patients or encounters cannot be used. We might add this feature if there is need.
  + i2b2 supports server-side Java based CRC plugins to handle special types of queries. Because CQ2 does not have a Java application layer, these plugins cannot be used. The equivalent in CQ2 would be an extra stored procedure, but we have not tried this.
* In CQ2, a query with only “Exclude” groups and no “Include” groups returns zero patients. This is different than i2b2, which assumes the user means to start with all patients. We plan to change the logic in CQ2 to match i2b2 in the future.
* CQ2 handles ontology-based query breakdowns the same as i2b2. However, custom breakdown SQL should be placed in the CRC.uspRunQueryInstanceBreakdown procedure, rather than the CRC.QT\_BREAKDOWN\_PATH table. We plan to add support for SQL in that table in the future.
* CQ2 does not support multiple fact tables in the same way as i2b2, which can reference specific fact tables from the ontology. Instead, we typically create a view named OBSERVATION\_FACT, which combines multiple fact tables. We might add this feature in the future.
* CQ2 does not queue queries like i2b2. We have a design on how to implement a similar feature in CQ2. However, queries in CQ2 are generally fast enough that they can be processed as soon as they are received. Thus, we have not yet had a need to implement query queues.
* CQ2 has an optimized query engine that leverages pre-computed aggregate tables and probabilistic sketches. This functionality does not exist in i2b2.

# Version History

Pre-Release Timeline

* 2007 – i2b2 web client created as an alternative to the i2b2 Workbench
* 2008 – Initial prototype of CQ2 as an alternative to the i2b2 application layer
* 2012 – First production instance of CQ2 at BIDMC
* 2019 – Addition of sketches to improve query performance

Version 1.0.0 (March 1, 2020)

* Released under a free academic license