# DESIGN AND IMPLEMENTATION OF SERVERLESS ARCHITECTURE FOR I2B2 ON AWS CLOUD AND SNOWFLAKE DATA WAREHOUSE

---

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

---

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

---

By

Md Saber Hossain

Dr. Abu Saleh Mohammad Mosa, Thesis Supervisor

May 2023

The undersigned appointed by the Dean of the Graduate School, have examined the thesis entitled:

# DESIGN AND IMPLEMENTATION OF SERVERLESS ARCHITECTURE FOR I2B2 ON AWS CLOUD AND SNOWFLAKE DATA WAREHOUSE

presented by Md Saber Hossain,

a candidate for the degree of Master of Science and hereby certify that, in their opinion, it is worthy of acceptance.

---

Dr. Abu Saleh Mohammad Mosa

---

Dr. Lemuel Russell Waitman

---

Tamara McMahon

---

# DEDICATION

I dedicate this thesis to my parents, whose unwavering support and encouragement have been the driving force behind my academic journey.

To the mentors who have guided me, the friends who have inspired me, cheered on me in every aspect of my life, the family who has always been there for me.

Colleagues who collaborated with me, believed in me, and for helping me grow as a scholar and a person.

Finally, I dedicate this thesis to myself, for the hard work, determination, and perseverance that I have demonstrated in pursuing my academic goals.

# ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Abu Saleh Mohammad Mosa for giving me the opportunity to work on this project and for his invaluable guidance and continuous support throughout my research journey. His expertise and encouragement have been instrumental in the completion of this thesis.

I would like to thank Vasanthi Mandhadi for her guidance and support throughout this project. Her knowledge and expertise in this field have been crucial in shaping my understanding of the objectives.

I am grateful to Md Kamruzzaman Rana and Yaswitha Jampani for their valuable contributions to this project. Their assistance has been indispensable in the successful completion of this thesis.

I would also like to thank all my colleagues at the NextGen Biomedical Informatics Center for their support and encouragement throughout my research journey. Their insights and feedback have been invaluable for this work.

Finally, I am grateful to my committee members, Dr. Lemuel Russell Waitman and Tamara McMahon, for their guidance and support throughout my thesis. Their feedback and constructive criticism have been instrumental in organizing my research work.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ORGANIZATION OF THE THESIS

**Chapter 1** presents the introduction of this thesis which includes the background material, problem statement, overall goal of this these. **Chapter 2** provides overview of i2b2 components and simplifying its installation process using docker containers. **Chapter 3** demonstrates the serverless deployment process of i2b2 in the AWS cloud. **Chapter 4** describes the creation of i2b2 data model wrapper on top of PCORnet CDM into i2b2 database. **Chapter 5** presents the snowflake JDBC driver integration and customization for i2b2 core services. **Chapter 6** discusses the results, accomplishments, limitations, and the future work for this study.

# ABSTRACT

Informatics for Integrating Biology and the Beside (i2b2) is an open-source medical tool for cohort discovery that allows researchers to explore and query clinical data. The i2b2 platform is designed to adopt any patient-centric data models and used at over 400 healthcare institutions worldwide for querying patient data. The platform consists of a web-client, core servers and database. Despite having installation guidelines, the complex architecture of the system with numerous dependencies and configuration parameters makes it difficult to install a functional i2b2 platform. On the other hand, maintaining the scalability, security, availability of the application is also challenging and requires lot of resources. Our aim was to deploy the i2b2 for University of Missouri (UM) System in the cloud as well as reduce the complexity and effort of the installation and maintenance process. Our solution encapsulated the complete installation process of each component using docker and deployed the container in the AWS Virtual Private Cloud (VPC) using several AWS PaaS (Platform as a Service), IaaS (Infrastructure as a Service) services. We deployed the application as a service in the AWS FARGATE, an on-demand, serverless, auto scalable compute engine. We also enhanced the functionality of i2b2 services and developed Snowflake JDBC driver support for i2b2 backend services. It enabled i2b2 services to query directly from Snowflake analytical database. In addition, we also created i2b2-data-installer package to load PCORnet CDM and ACT ontology data into i2b2 database. The i2b2 platform in University of Missouri holds 1.26B facts of 2.2M patients of UM Cerner Millennium data.

# CHAPTER 1 : INTRODUCTION

## 1.1 DATA ANALYTICS PLATFORMS IN HEALTH RESEARCH

The ability to collect and analyze large amounts of health data is important for translational research. This data can be used to identify new patterns and trends, develop new treatments and interventions, and improve the overall quality of care. However, limited access to real-world data due to the patient's privacy protection and security of health information laws[1]. These laws were put in place to protect patient privacy and ensure the security of health information, but they have also created barriers to data access for researchers.

Moreover, the lack of infrastructure for trustworthy self-service query access in health research exacerbates these challenges[2]. Researchers often require a significant amount of technical expertise to navigate the complex systems and processes that are necessary to access and analyze health data. This can lead to delays in data access and increased costs associated with data management, which can hinder the progress of translational research.

Informatics for Integrating Biology and the Bedside (i2b2) platform was developed to help address these issues. i2b2 is an open-source tool and provides a user-friendly interface that allows researchers to access and analyze data in a secure and efficient manner, without the need for extensive technical expertise or assistance[3].

## 1.2 THE I2B2 PLATFORM

The Informatics for Integrating Biology and the Bedside (i2b2) is an initiative that was sponsored by the NIH Roadmap National Centers for Biomedical Computing. This initiative is focused on advancing biomedical computing and informatics research by facilitating the integration of clinical and research data[3]. The i2b2 platform provides a framework for researchers to extract and store data from electronic medical records and other sources, enabling more efficient and effective analysis of data for research purposes. Through its partnership with the NIH Roadmap National Centers for Biomedical Computing, i2b2 is helping to drive innovation in biomedical research and improve patient outcomes.[3]

The i2b2 platform is designed to provide clinical investigators with the necessary software tools to manage and collect clinical research data in a cohesive manner in the genomics age. The i2b2 hive is a set of software modules, known as "cells," that utilize a common messaging protocol for interaction through web services and XML messages. Each cell can be developed independently by researchers to meet specific analytical objectives and can then be integrated into the hive to enhance the functionality available in the i2b2 hive[3]. The i2b2 hive organized data in a way that ensures data ownership and

privacy are maintained, even when shared across multiple groups or entities. This organizational structure helps to ensure that sensitive information is not disclosed to unauthorized parties while still allowing for collaborative research efforts between different groups. By preserving the privacy of the data, the i2b2 hive enables researchers to access and analyze data without compromising the confidentiality of patients or the integrity of the data.[3] The i2b2 hive is a collection of modules that interact through Web Services. Within this framework, the core modules are represented by the dark color cells (Figure 1) are essential for the operation of the i2b2 hive. Conversely, the optional modules are represented by light color cells and provide supplementary functionality to the hive (Figure 1).

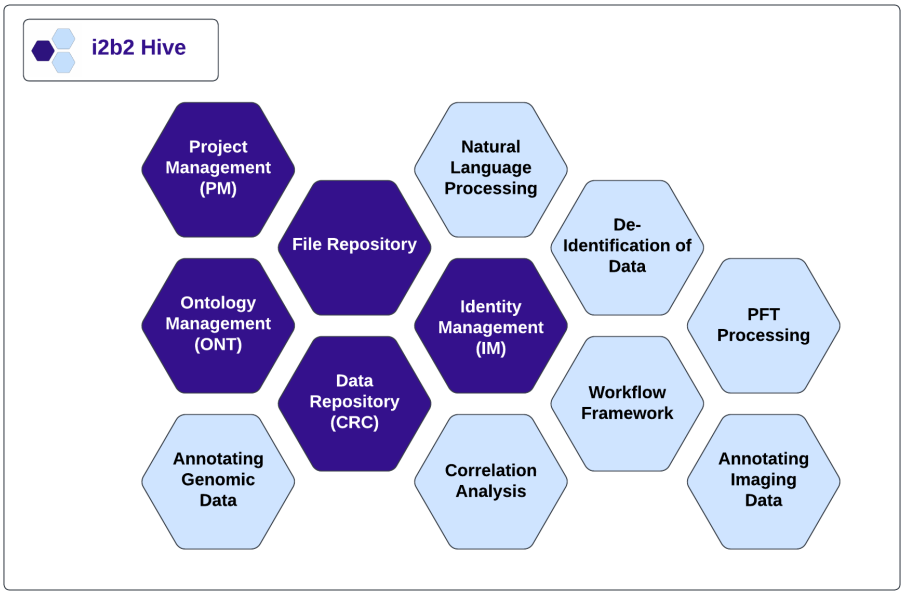


**Figure 1: An i2b2 hive**

With i2b2, researchers can create and execute their own queries on large datasets, which can significantly reduce the time and costs associated with data analysis. However,

like many open-source scientific software packages, i2b2 can be challenging to install and maintain including patching, upgrading, data modeling, and ontology mapping. These tasks present a significant obstacle to i2b2's wider adoption by institutions[4].

## 1.3 DOCKER CONTAINERIZATION

The Docker containerization process refers to the creation and deployment of software applications in isolated containers, which are separate from the host operating system and other running applications. It is an open-source platform that offers a range of tools for building, shipping, and running containerized applications. Containers are a lightweight and portable way of packaging applications and their dependencies, enabling developers to build software that can be easily deployed across different environments, including development and production[5]. Docker containers are created based on images that contain all the necessary components required to run an application, such as libraries, frameworks, and configuration files. This approach enables developers to create self-contained, isolated containers that can be easily ported across different platforms, including laptops and cloud servers, providing greater flexibility and ease of management. It is possible to simplify the distribution process by packaging server environments, dependencies and softwares required for i2b2 servers in the corresponding docker containers and reduce the effort and complexity to install a functional i2b2 platform[4].

## 1.4 AMAZON WEB SERVICES

AWS is a cloud computing platform that provides PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) services to help organizations run their applications and

manage their infrastructure[6]. It offers a wide range of cloud services. One of these services is ECS FARGATE a Platform as a Service, which allows users to run containers without managing the underlying infrastructure. The ECS FARGATE can be used with an Application Load Balancer (ALB) to distribute incoming traffic across multiple ECS tasks. To ensure security and isolation between the two services, it is recommended that the services be deployed within the same Virtual Private Cloud (VPC). VPC is an Infrastructure as a Service (IaaS) solution that provides a secure and isolated environment for running cloud-based services. The combination of ECS FARGATE, an ALB, a VPC and other services, provides a flexible and scalable platform for managing containerized applications on AWS in a secure way. There are three different servers are required to host a functional i2b2: the webserver, application server, and database server (Figure 2). Maintaining these servers on-premises can be a time-consuming and resource-intensive task. On the other hand, deploying the i2b2 in a secure network can be a challenging task as well. To overcome this, AWS cloud infrastructure is a viable option as it allows for easier maintenance and scalability.

## 1.5 SNOWFLAKE DATA ANALYTICS PLATFORM

Snowflake is a cloud-based data analytics platform that provides Software as a Service (SaaS) solutions for a complete data warehousing and analytics solution for institutions of all sizes. It is built on top of a high-performance, scalable, and secure data architecture that allows users to store and analyze data from multiple sources in real-time[7].

One of the key features of Snowflake is its ability to handle complex queries and data processing tasks through its Online Analytical Processing (OLAP) database. OLAP databases are designed to support complex analytical queries and are optimized for read-intensive workloads[8]. Snowflake's OLAP database uses a columnar storage model that allows for faster query processing and improved compression of data. Additionally, Snowflake database is fully integrated with its data warehousing and analytics capabilities, allowing efficient handling of large data volumes and complex queries. Snowflake cloud-based data warehousing and analytics platform offers a serverless computing model[7]. This means that customers do not need to provision or manage any infrastructure for their data warehouse, as Snowflake takes care of all the underlying infrastructure. Snowflake data warehouses can be scaled up or down automatically as needed, without worrying about provisioning or managing any infrastructure to support heavy workloads. On the other hand, queries running in i2b2 involve analyzing large volume of data are typically better suited for an OLAP database, which is optimized for complex analytical queries[8]. However, currently the i2b2 platform only supports OLTP database such as Oracle, Microsoft SQL Server, and PostgreSQL, which are better suited for transactional workloads.

## 1.6 OVERALL OBJECTIVES

Our primary aim is to introduce i2b2 as a self-service query tool for researchers at the University of Missouri. To achieve this, we intend to install i2b2 on top of the University of Missouri health system data and deploy the components of i2b2 in the cloud to minimize server management efforts. In this study, we seek to simplify the installation

process for i2b2 components by utilizing docker containerization process. Additionally, we plan to demonstrate a secure and scalable approach to deploy i2b2 components in the AWS cloud and the Snowflake data warehouse. We also aim to create i2b2 data-installer package that utilizes the ACT V4 ontology to transform PCORnet Common Data Model (CDM) data to i2b2 data. Finally, we aim to develop Snowflake JDBC driver support for i2b2 core services so that i2b2 services can use Snowflake analytical database as its backend database.

# CHAPTER 2 : COMPONENTS OF THE I2B2 PLATFORM

The i2b2 platform is composed of multiple cells that communicates using XML based web services. Each cell serves unique set of responsibility. The modular design facilitates the addition of new cells, and therefore new functionalities, which enable the extension of the i2b2 platform to a wide range of use, cases, and environments. Compared to many general software packages, i2b2 is a complex platform that offers a versatile implementation for handling diverse operations related to data storage, querying, and user interactions[3]. Its functionality is achieved through the integration of various web services working in together. The i2b2 is an outcome of collaborative efforts among researchers and has evolved from an earlier hospital-specific implementation. In terms of scientific software, i2b2 shares several traits with other open-source tools. It is developed by experts with substantial domain knowledge, constructed with an agile approach, challenging to test due to its broad scope of use cases, and complex to install.[4]

## 2.1 OVERVIEW OF AN I2B2 HIVE

A basic i2b2 platform comprises three core components: a frontend web-client, a backend consisting of Web services, and databases (Figure 2). The required i2b2 hive includes five cells: (1) the Data Repository (CRC) for storing and querying patient data, (2) Ontology Management (ONT) to represent concepts, (3) Project Management (PM) for hive setup, (4) File Repository (FR) for managing i2b2 files, and (5) Workplace (WORK) for managing user-specific XML objects. The i2b2 web services are hosted on the JBoss Wildfly server (Table 1), which provides a framework based on enterprise Java and Apache Axis2 and facilitates web service scalability. The JBoss Wildfly maintains connection pools of SQL data-sources for each corresponding i2b2 cell. The Apache web server (Table 1) hosts webpages that include HTML, CSS, and JavaScript for the i2b2 web client, and it functions as a proxy to direct asynchronous JavaScript (AJAX) class from the web-client to web services running on the Wildfly server.

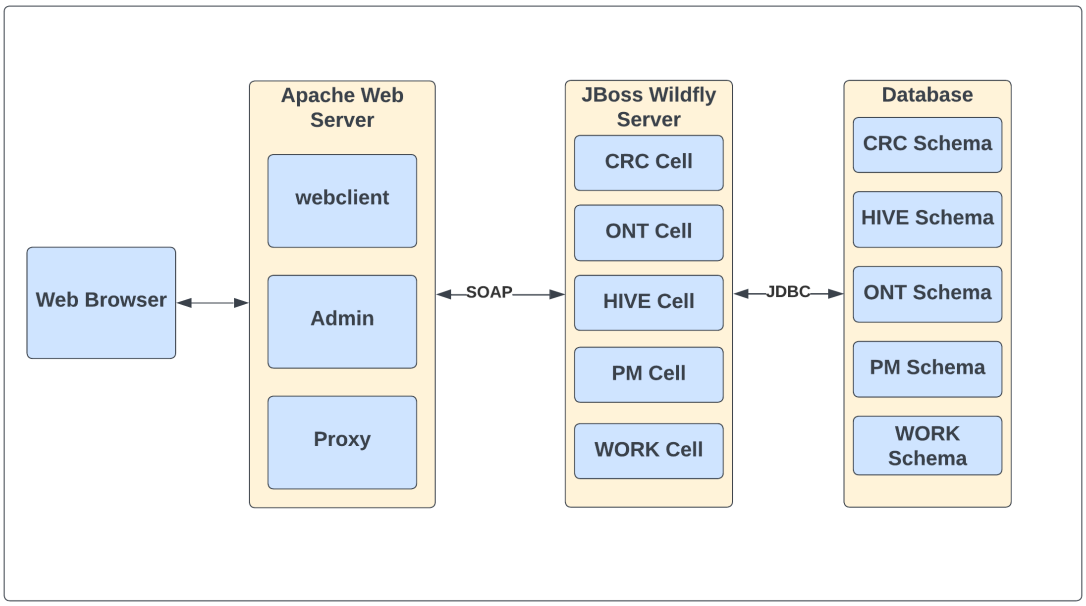


**Figure 2: High-level architecture of i2b2 platform**

The core database is relational type and can be hosted in a PostgreSQL server, Microsoft SQL Server, or Oracle SQL server (Table 1). The database server accommodates a cell-specific database schema for each hive cell, accessible solely through the corresponding cell. Password-protected database accounts ensure that only authorized users can access the cell-specific databases. Patient data and configuration information for each of the core i2b2 cells are also stored in the SQL database.

**Table 1: Summary of the high-level components of the i2b2 platform**

| High-level Component | Implementation | Responsibility |
|---|---|---|
| Web server | Apache HTTP server | Hosts HTML and JavaScript based i2b2 web-client. |
| Application Server | Web services running inside JBoss Wildfly server | Provides SOAP based API (Application Programming Interface) and serve as backend for i2b2 web-client. These services offer user management and authentication functions and convert user queries into SQL queries. |
| Database | PostgreSQL, Oracle, Microsoft SQL | Contains patient data, ontology definitions, hive, and user data in corresponding schema. |

In general, the users access the i2b2 by loading the web client on an internet browser. The i2b2 web client communicates with the backend web services by exchanging SOAP XML messages, which are routed through a proxy. The web services are designed to be stateless and retrieve data from the backend SQL database. For instance, when a user logs in, the web client sends the user's credentials to the project management cell, which then authenticates the user's credentials by verifying them in the back-end PM service. Similarly, when a user requests the count of patients with a specific diagnosis, the web client communicates with the data repository cell, which generates and executes SQL queries on the 'fact' tables (described in section 4.1) located in the backend SQL database, and then returns the count to the front-end web client.

## 2.2 INSTALLING THE I2B2 PLATFORM

The i2b2 platform follows a modular architecture where the components, referred to as cells, communicate with each other through XML-based web services. This design enables the independent implementation and installation of cells. The cells are categorized as core or optional, where core cells are necessary for a functional installation and optional cells provide additional services like Natural Language Processing[3]. The implementation of the platform is in Enterprise Java with the user interface developed in HTML-JavaScript. The platform's source code is available as open source through GitHub, and extensive documentation is available for compiling and installing i2b2 cells. Additionally, there is an online demonstration version of the software available for showcasing its functionality. Despite the availability of online documentation, tutorials, and an active user group, new users require several weeks to create a functional i2b2 installation[4]. The substantial effort required to set up a new i2b2 hive installation represents a significant challenge to wider adoption of the platform, particularly for smaller projects with limited experience in informatics.

The installation of the i2b2 platform is a challenging process that requires a moderate level of expertise in Enterprise Java and Java build tools for the compiling and deployment of the code. Another challenge is the need to adopt new installation steps to accommodate new software features and dependencies (Table 2). Additionally, due to the platform's design to be compatible with various operating systems and databases, the numerous possible configurations (Table 3) result a significant obstacle for following the exact steps to achieve a required specific configuration. These challenges, in combination, limit the utilization of the i2b2 platform by numerous institutions. Regular upgrades of the

i2b2 platform installation, including its cells, database, and operating systems, are essential for adding new functionality. However, the risk of disrupting an operational i2b2 installation often causes informatics teams to delay their upgrade efforts. A potential solution to these challenges is containerization, which has proven effective for packaging software. The use of Docker containers offers the potential to upgrade an installation by replacing deployed container images with the latest implementation[5,9].

**Table 2: Dependencies required for installing the i2b2 platform**

| i2b2 Component | Dependency | Description |
|---|---|---|
| Web server | Centos 7, httpd24, PHP, proxy, mod_ssl, shibboleth.x86_64 | Apache web server deployed in CentOS 7, an open-source operating system. The system also supports a proxy server and the mod_ssl module for secure communication. Additionally, the shibboleth.x86_64 package is available to provide support for Single Sign-On authentication. |
| Application server | JBoss Wildfly, Apache-ant, Java JDK, JDBC driver | - Alpine is a lightweight secure Linux distribution<br>- Wildfly is an open-source Java application server<br>- Apache Ant to build i2b2 cells<br>- JDK is required for compiling and running i2b2 services<br>- JDBC driver is necessary for connecting to databases |
| Database server | PostgreSQL, Oracle, and Microsoft SQL | Compatible relational database engines |

**Table 3: Configuration files required for installing the i2b2 platform**

| i2b2 component | Configuration | Description |
|---|---|---|
| Web Server | httpd.conf | The main configuration file for the Apache HTTP Server. It is used to configure various settings related to the operation of the web server, such as the ports it listens on, the server's name, and the document root directory. |
| | ajp.conf | It is used to configure the Apache JServ Protocol (AJP), that enables communication between webserver and an application server. It also defines URL prefix for proxying i2b2 web services for the web-client |
| | shibboleth2.xml | Main configuration file used by the Shibboleth Service Provider (SP). The file contains the basic configuration parameters for the Shibboleth SP, such as the entityID, the location of the metadata, the SSL certificate and key, the session timeout, and the attribute mapping rules. |
| | Shib.conf | Contains settings that configure the Shibboleth SP module. This includes defining the identity provider (IdP) entity ID, metadata location, and SSL/TLS settings etc. |
| | attribute-map.xml | A configuration file used by the Shibboleth Service Provider (SP) to map attributes received from the Identity Provider (IdP) to local attributes that can be used by the application or service |
| | httpd-shibd-foreground | It starts shibboleth and Apache server |
| Application Server | Data sources for each of the i2b2 cells | To manage connections to databases for each i2b2 cells in the Wildfly server. It contains JNDI (Java Naming and Directory Interface) name, Connection URL, Driver name, Authentication credentials, Connection pool settings |

## 2.3 THE I2B2 PLATFORM IN DOCKER CONTAINERS

Containerization is a virtualization method that enables the deployment and execution of applications in isolated environments called containers. A container is a lightweight, portable, and self-contained package that encapsulates an application along with its dependencies, configuration files, libraries, and runtime environment. Containerization provides a level of abstraction that allows an application to be run uniformly across different computing environments, such as different operating systems or cloud providers, without any modification to the application code[5]. This technology has become increasingly popular due to its ability to simplify software deployment, enhance scalability, and improve resource utilization. The most popular containerization technology is Docker, which uses container images to create and run containers. Docker uses a client-server architecture, where the Docker client communicates with the Docker daemon, which is responsible for building, running, and managing containers. Docker provides a wide range of commands that allow developers to create and manage containers, images, networks, and volumes.

We created two docker containers called 'i2b2-webclient', 'i2b2-wildfly' to encapsulate the core functionalities of the i2b2 platform, as summarized in Table 2 and Table 3. The source codes for building the container images along with the configuration files are published in the GitHub repository of University of Missouri NexGen Biomedical Informatics (https://github.com/Missouri-BMI/I2B2_SERVERLESS).

### 2.3.1 Building 'i2b2-webclient' Docker Image

The 'i2b2-webclient' Docker image is designed to provide an Apache web server environment running on a Centos 7 base image. During the build phase, the image downloads the necessary i2b2 front-end source code from the repository and deploys it in the web server. It also includes configuration files that are required for the web server to function properly, which are listed in the Web Server section of Table 2 and Table 3.

In addition to setting up the web server, the i2b2-webclient image also enables the SAML (Security Assertion Markup Language) authentication in the i2b2 and configures the Shibboleth Service Provider (SP) inside the container runtime. This enables the web server to securely communicate with other Shibboleth-enabled systems and to provide single sign-on capabilities to users. The necessary Shibboleth configuration files are also included in the image during the build phase.

By providing a pre-configured i2b2-webclient image, developers and researchers can easily deploy the i2b2 front-end in a containerized environment. This can simplify the deployment process and provide a consistent environment for running the application, which can improve reliability and reduce the risk of configuration errors.

The following steps are instructed in the ‘i2b2-webclient’ Dockerfile that builds a Docker image based on CentOS 7:

i) Sets default values for two arguments, ‘CLIENT_TYPE’ and ‘SERVER_NAME’.

ii) Updates the environment variables for ‘CLIENT_TYPE’ and ‘SERVER_NAME’.

iii) Copies the Shibboleth RPM installation file to /etc/yum.repos.d/.

iv) Updates the packages and installs Apache, mod_ssl, PHP, Shibboleth and wget using yum.

v) Downloads the webclient code from a forked repository and extracts it to /var/www/html/${CLIENT_TYPE}.

vi) Replaces the default Apache and Shibboleth configuration files with custom ones.

vii) Copies required scripts and configuration files for Apache and Shibboleth.

viii) Sets up the environment by running /usr/local/bin/environment-setup.

ix) Exposes port 80.

x) Runs the command /bin/bash -C httpd-shibd-foreground as the default command to start the container.

#### 2.3.1.1 Instructions for building and running the ‘i2b2-webclient’

The following command was executed in building the ‘i2b2-webclient’ Docker image in the local environment.

```
docker build \
-t i2b2-webclient \
--build-arg CLIENT_TYPE=webclient \
--build-arg SERVER_NAME=localhost \
--no-cache.
```

The first command builds the i2b2-webclient docker image using the two arguments listed below:

CLIENT_TYPE: [webclient/admin]

SERVER_NAME: DNS name of the hosted server

Then we ran the ‘i2b2-webclient’ in the local environment using following command:

```
docker run -p 80:80 i2b2-webclient
```

During the run phase, port mapping is provided to map the port on the host machine to the port on the container where Apache web server is listening. This allows you to access the i2b2-webclient in a web browser.

### 2.3.2 Building ‘i2b2-wildfly’ Docker Image

The "i2b2-wildfly" Docker image provides the JBoss Wildfly server, which includes the installation of the Apache Axis2 WAR file in the Wildfly folder. This

installation facilitates web services for the server. As part of the build process, the i2b2 cell source code is compiled into a WAR archive and installed in the WildFly server. Additionally, the Dockerfile configures the data sources for JDBC connections for each cell based on the configuration listed in the Application Server section of Table 2 and Table 3. The following steps are instructed in the 'i2b2-wildfly' Dockerfile that builds a Docker container based on Alpine Docker image where the Alpine Docker image provides a basic Linux distribution:

i) Set base image as 'alpine:3.14'
ii) Download required packages 'openjdk8', 'apache-ant', 'wget', 'unzip' using 'apk' package installer
iii) Download i2b2-core-server source code from the github repository
iv) Unzip downloaded source code, compile and distribute WAR achieves using 'apache-ant'
v) Deploy WAR achieves in JBoss Wildfly
vi) Expose port 8009
vii) Configure Wildfly data sources for JDBC connections in container runtime using jboss-cli.
viii) Start JBoss Wildfly server

#### 2.3.2.1 Instructions for building and running ‘i2b2-wildfly’

The following command was executed in building ‘i2b2-wildfly’ Docker image in the local environment:

```
docker build \
-t i2b2-widlfly \
--no-cache.
```

Then we executed the following command to run the ‘i2b2-wildfly’ Docker image in the local environment:

```
docker run -p 8009:8009 –env-file .env i2b2-wildfly
```

In Docker build phase, only the tag of the container image was provided. During, the run phase, port mapping was provided to map the port on the host machine to the port on the container where JBoss Wildfly server is listening.

An ‘.env’ file containing a list of environment variables listed in (Table 4) was also provided to configure the JDBC Data sources in Wildfly. The environment variables are the database credentials that were used to establish JDBC connections for each cell in the Wildfly server.

**Table 4: List of environment variables to configure JDBC Wildfly Data Sources**

| Environment Variable | Type | Data Source Name | i2b2 cell |
|---|---|---|---|
| DB_CRC_URL | JDBC URL | QueryToolDS | CRC |
| DB_CRC_USER | JDBC username | | |
| DB_CRC_USER_PASS | JDBC password | | |
| DB_HIVE_URL | JDBC URL | CRCBootStrapDS, OntologyBootStrapDS, WorkplaceBootStrapDS | HIVE |
| DB_HIVE_USER | JDBC username | | |
| DB_HIVE_USER_PASS | JDBC password | | |
| DB_ONT_URL | JDBC URL | OntologyDS | ONT |
| DB_ONT_USER | JDBC username | | |
| DB_ONT_USER_PASS | JDBC password | | |
| DB_PM_URL | JDBC URL | PMBootStrapDS | PM |
| DB_PM_USER | JDBC username | | |
| DB_PM_USER_PASS | JDBC password | | |
| DB_WD_URL | JDBC URL | WorkplaceDS | WD |
| DB_WD_USER | JDBC username | | |
| DB_WD_USER_PASS | JDBC password | | |

## 2.4 SUMMARY

We were able to successfully deploy the 'i2b2-webclient' and 'i2b2-wildfly' docker containers in the docker host running inside the local machine. The i2b2 web-client was accessible via localhost URL. In order to test the full functionality of i2b2 web application in local environment, we have used the i2b2-pg image, a PostgreSQL server populated with i2b2 synthetic dataset of 200 patients provided by the i2b2 tranSMART foundation (https://github.com/i2b2/i2b2-docker/blob/master/pg/docker-compose.yml). We replaced the i2b2-web, i2b2-wildfly image with our created container images and deploy the i2b2 platform using command 'docker-compose up –build' in the docker CLI (Command-Line

Interface). Finally, we tested the system if a user could successfully login using i2b2 web client running in a local machine and build a query to execute.

# CHAPTER 3 : DEPLOYING I2B2 IN THE AWS CLOUD

In this study, the 'i2b2-webclient' and 'i2b2-wildfly' have been containerized using Docker and subsequently deployed on the Amazon Web Services (AWS) cloud computing platform. The utilization of Docker containerization technology allows for the efficient encapsulation of these i2b2 platform components, while AWS provides a scalable and reliable infrastructure for hosting these containers. This deployment approach offers several benefits, including enhanced flexibility and portability of the i2b2 platform, as well as improved resource utilization and ease of management. It also automates the deployment, scaling, and availability of i2b2 components, which reduces the operational burden for IT teams.

Serverless computing is a novel approach to cloud computing in which cloud providers assume responsibility for the underlying infrastructure, including servers and operating systems, relieving developers of the burden of managing and maintaining such infrastructure. The need for serverless computing has arisen due to the desire for more efficient and cost-effective ways of deploying and scaling applications without the need to manage server infrastructure. We have utilized the Amazon Elastic Container Service (ECS) with FARGATE launch type, coupled with various AWS services (Table 5), to deploy the i2b2 platform in the cloud.

**Table 5: List of AWS resources used in the deployment process**

| AWS Resources | Description |
|---|---|
| VPC (Virtual Private Cloud) | Enables resources to be launched into a virtual network, providing complete control over the network environment. This includes the ability to select IP address ranges, create subnets, configure route tables, and network gateways. The use of VPC enables the implementation of multiple layers of security, such as security groups and network access control lists, to manage and control access to running containers. |
| ECR (Elastic Container Registry) | A fully managed container registry service that facilitates secure storage, management, and deployment of Docker container images. ECR integrates with other AWS services, such as Elastic Container Service (ECS), to provide a reliable and streamlined approach to deploying containerized applications in the cloud. |
| ECS (Elastic Container Service) | A cloud computing service that simplifies the deployment, management, and scaling of containerized applications in the cloud. It enables users to launch and manage containers on a cluster of Amazon Elastic Compute Cloud (EC2) instances or using the serverless compute engine AWS FARGATE. ECS allows users to define task and service definitions for their containers, specify resource requirements, and automate deployment and scaling. |
| FARGATE | A serverless compute engine that eliminates the need to manage underlying infrastructure when running containers. It is a fully managed service that offers flexible and granular resource allocation, with the ability to define CPU and memory requirements for containers. FARGATE integrates with other AWS services, such as Elastic Load Balancing and Amazon Virtual Private Cloud, to provide a secure and scalable platform for running containerized applications in the cloud. The service simplifies the process of running containers, allowing users to focus on application development and deployment. |
| Application Load Balancer | It enables the distribution of incoming traffic across multiple targets, such as Amazon Elastic Compute Cloud (EC2) instances or containers running on AWS FARGATE. ALB supports Layer 7 routing and provides advanced features, such as SSL/TLS termination, content-based routing, and sticky sessions. ALB supports container-based workloads and is compatible with popular container orchestration platforms, such as AWS ECS |
| Web Application Firewall | It protects web applications from common web attacks and exploits. With a set of customizable rules, WAF filters and blocks unwanted traffic based on various criteria, such as IP addresses, HTTP headers, and query strings. The service integrates with other AWS services like Elastic Load Balancing, to provide a scalable and distributed layer of security. Advanced features like rate limiting, geo-blocking, and bot management provide additional protection against emerging threats and attacks. WAF improves the security posture of web applications running on AWS by providing an effective and configurable layer of protection against known and unknown web threats. |

## 3.1 NETWORK INFRASTRUCTURE FOR I2B2 IN THE AWS CLOUD

The network infrastructure for i2b2 in the AWS cloud (Figure 3) creating a VPC with two public and two private subnets in different availability zones. The separation of subnets in multiple availability zones helps to ensure high availability and fault tolerance. To route network traffic, an internet gateway, and a Network Address Translator (NAT) gateway were used. The routing table associated with the public subnets was configured to route network traffic to the internet gateway, while the routing table associate with the private subnets was set up to route traffic to the NAT gateway. This configuration enables instances in the private subnets to access the internet through the public NAT gateway while maintaining a secure network architecture.

A security group was created to enhance the security of the i2b2 application. The security group of the i2b2 application allows inbound HTTP traffic on port 80 from all IPV4 addresses and all outbound IPV4 traffic. An application load balancer was also created with an SSL certificate and a target group. The application load balancer is responsible for routing incoming traffic from the clients to the registered target in the two availablility zones. The application load balancer listens on port HTTPS:443 and forward network traffic to port 80 via i2b2 application security group. The application load balancer also monitors the health of its registered target and scales the workload accordinly, ensuring that the i2b2 application is always available and responsive. In addition to the network infrastructure, a Web application Firewall (WAF) is also attached to the application load balancer (ALB) to enhance the security of the i2b2 platform. The WAF

provides a set of customization rules to filter and block unwanted traffic based on various criteria, such as IP addresses, HTTP headers and query strings. The ALB acts as a single entry point for the i2b2 webclient. The WAF is integrated with the ALB to inspect incoming traffic and filter out any malicious requests or attacks. The ALB forwards the filtered traffic to the container instances, which provides an added layer of security for the i2b2 application. This network configuration enables the i2b2 application to be protected against common web exploits and attacks while providing secure and reliable access to the authorized users.

To run the ‘i2b2-webclient’ and ‘i2b2-wildfly’ as an AWS FARGATE service, a task definition for the i2b2 application was created with network mode set to awsvpc, operating system family to Linux, and required compatibilities to FARGATE. The task definition was then configured by providing the repository URL of the ‘i2b2-webclient’ and ‘i2b2-wildfly’ docker images stored in the Elastic Container Registry (ECR), port mappings and required environment variables (Table 4) for the container images. In ECS, port mapping allows mapping the container's internal port to a port on the host instance or the load balancer. This mapping enables network traffic to reach the container and the application running inside it through a specified port. Without port mapping, the container's port would be unknown to the outside world, making it inaccessible and unusable. Finally, an AWS FARGATE service for the i2b2 application was created using the task definition and VPC configuraiton that had been established for the i2b2 service.

We decided to use Snowflake as i2b2 back-end database. Snowflake is a cloud-based data warehousing platform that offers a range of features designed to support modern data analytics workloads, such as autmatics scaling, support most common standarized

version of SQL, and JDBC driver compatibility. Currently, the i2b2 platform supportes Oracle, PostgreSQL, and the Microsoft SQL Server as database. However, we modified the source code of i2b2-core-servers to support Snowflake as i2b2 data source. The development process will be described later in this study. The i2b2 ECS services running inside containers communicate with snowflake database via JDBC connection.

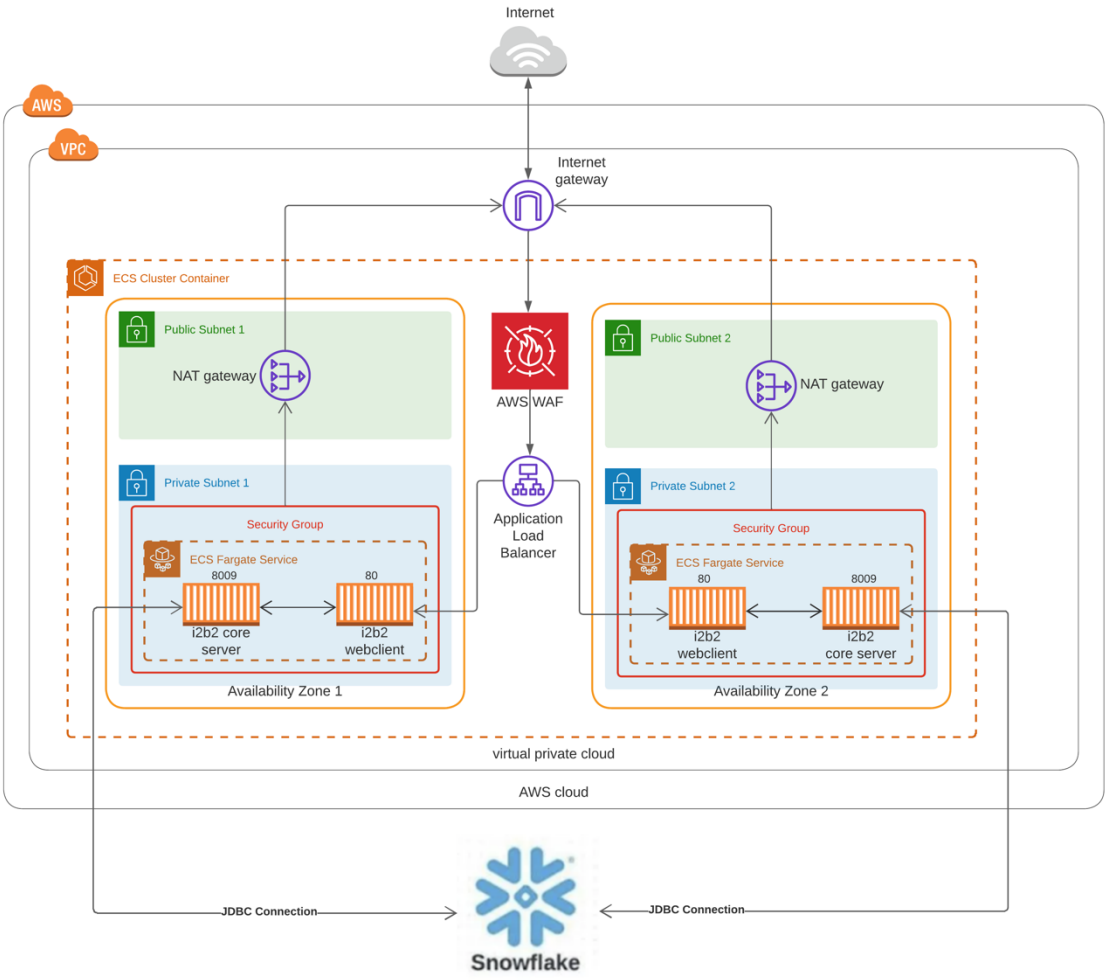


**Figure 3: Overall Network Infrastructure for i2b2 service in the AWS Cloud**

## 3.2 THE I2B2 WEBCLIENT AND WILDFLY SERVER IN THE ECS SERVICE

To deploy i2b2 in the AWS ECS FARGATE environment, we had to create two Docker containers for the 'i2b2-webclient' and 'i2b2-wildfly'. Initially, we tested these containers on our local machine to ensure their functionality. However, our main objective was to deploy these containers in the AWS ECS FARGATE environment. This environment provides a serverless infrastructure, which means that the FARGATE environment manages the infrastructure for us. We do not have to manage the underlying servers, operating systems, or maintenance tasks associated with them. This approach provides us with a highly scalable and cost-effective way of running i2b2 in the cloud. By using AWS FARGATE, we can automatically scale our application as demand fluctuates and pay only for the computing resources we use. This reduces the overhead cost of managing our own infrastructure and allows us to focus on the development and deployment of the i2b2 application. Running containers in ECS (Figure 4) involves several steps. First, we created two private repositories in the AWS ECR and upload the docker images in the ECR. Then, we created the task definition in the ECS using the uploaded container images. Finally, we created the i2b2 service in AWS ECS.

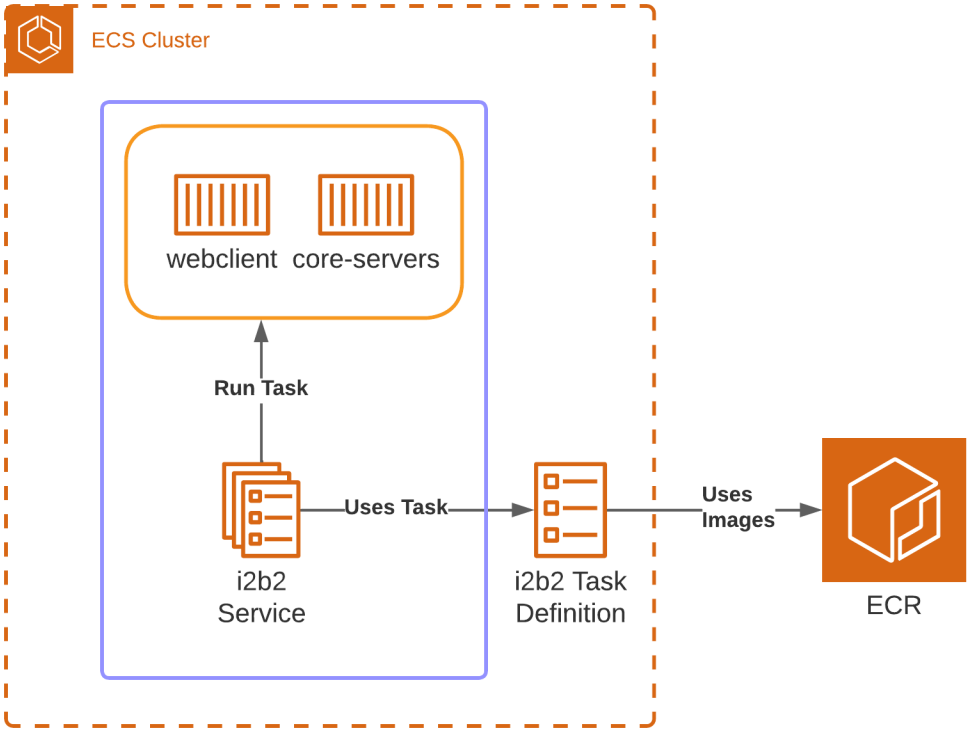


**Figure 4: Running i2b2 containers as ECS service**

### 3.2.1 Uploading Docker Images in ECR

Our research project involved uploading 'i2b2-wildfly' and 'i2b2-webclient' Docker images to the AWS Elastic Container Registry (ECR). The ECR is a fully managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker images. To begin the process of uploading Docker images to ECR, we first had to ensure that we had the appropriate credentials and permissions to access the ECR. This involved setting up an IAM (Identity and Access Management) user with the necessary permissions, generating an access key and secret access key, and configuring the AWS CLI to use these credentials. To log in to the AWS CLI with an AWS SSO role-based profile

we used the "aws sso login --profile <profile_name>" command. This command guides us through a process that involves opening a web browser to authenticate our access and then retrieving temporary AWS credentials for use in subsequent CLI commands.

```
aws sso login –profile <profile_name>
```

Then, we logged into the AWS ECR repository using docker command line interface command with the aws login credential.

```
aws ecr get-login-password –profile <profile_name> \
| docker login –username AWS –password-stdin \
<aws_account>.dkr.ecr.<region>.amazonaws.com
```

Next, we created two private repositories in the ECR for ‘i2b2-webclient’ and ‘i2b2-wildfly’ and retrieved the image URIs for the containers using the commands listed below. We used those image URIs as the tag of docker images that we wanted to upload with the ECR repository URI. This repository URI is unique to each ECR repository and is used to identify the location of the repository. The following command was used to create the ‘i2b2-webclient’ docker image repository in the AWS ECR.

```
aws ecr create-repository \
--repository-name i2b2-webclient \
--image-scanning-configuration scanOnpush=true \
--profile <profile_name>
```

The following command was used to create the ‘i2b2-wildfly’ docker image repository in the AWS ECR.

```
aws ecr create-repository \
--repository-name i2b2-wildfly \
--image-scanning-configuration scanOnpush=true \
--profile <profile_name>
```

Then we built and tagged the ‘i2b2-webclient’ Docker image with the ECR image URI using the command shown below:

```
docker build \
-t <image_tag> \
--build-arg CLIENT_TYPE=webclient \
--build-arg SERVER=<SERVER_NAME> \
--no-cache .
```

We also built and tagged the ‘i2b2-wildfly’ Docker image with the ECR image repository URI using following command:

```
docker-build \
-t <image_tag> \
--no-cache .
```

Once we built and tagged the Docker images, we used the Docker CLI to authenticate our Docker client with the ECR registry. This involved running a command that generated a Docker login token, described earlier in this chapter. Finally, we pushed

the images using docker push command ‘docker push <image_tag>’ to upload the images in the ECR.

### 3.2.2 Creating the i2b2 Service in the ECS

In this step, we created ECS task definition for i2b2 service. ECS task definition define the containers that will be launched in the ECS FARGATE environment. We set the application environment to FAGATE, which means the containers will run on AWS FARGATE. We also specified the operation system as LINUX and configured the task with 4vCPUs and 16GB of memory. This configuration ensures that the containers have enough resources to run efficiently. We have attached the ‘i2b2-webclient’ and ‘i2b2-wildfly’ containers to the task definition with their associated port mappings and environment variables. The ‘i2b2-wildfly’ container is responsible for running the i2b2 application server, while the i2b2-webclient container serves as i2b2 web server. Finally, we created a cluster in the ECS and launch i2b2 service using the task definition.

## 3.3 SUMMARY

We were able to successfully deploy the i2b2 docker container in the AWS ECS. We observed the events log from the i2b2 ECS service in the AWS console to make sure if it has reached a steady state. The i2b2 web client is accessible via DNS URL (https://i2b2.nextgenbmi.umsystem.edu/webclient/) and we were able to login using University of Missouri Shibboleth authentication (Single Sign-On). We wanted to provide highly secured environment to the i2b2 container instances from the web attacks. Deploying the i2b2 ECS service in the private subnet and access via application load

balancer and security groups ensures the security of the i2b2 application. A securing analysis was conducted by the University of Missouri Security Analyst team and found no Critical or high-risk issues in the report generated with the security analysis tool (Table 6). The security team was able to address a total of 57 issues, including 4 of medium severity and 45 of lower severity. Although the addressed issues adhered to best practices for server-side applications, they were not considered harmful from a malicious attacker's perspective.

**Table 6: Overview of report generated by Security Scan**

| **Issue Type** | **Issues Count** |
|---|---|
| Medium severity issues | 4 |
| Low severity issues | 45 |
| Informational severity issues | 8 |
| Total security issues included in the report | 57 |
| Total security issues discovered in the scan | 57 |

# CHAPTER 4 : PREPARING DATA FOR I2B2 DATABASE

## 4.1 THE I2B2 DATAMART AND ONTOLOGY

The Data Repository Cell (referred to as CRC) was created with the aim of holding clinical data from a variety of sources, including clinical trials, medical record systems, and laboratory systems. This data is stored in three tables, namely patient, visit, and observation tables. Apart from these, there are three lookup tables, such as concept, provider, and code tables, and two mapping tables named 'patient_mapping' and 'visit_mapping'. [10]

The i2b2 data mart is a data storage system that is designed based on the star schema structure (Figure 5), which was originally proposed by Ralph Kimball. This schema consists of a central fact table that is surrounded by one or more Dimension tables. The essential aspect of constructing a star schema is to identify what constitutes a fact.[10]

In the context of health data, a logical fact represents an observation made on a patient. It is worth noting that an observation might not signify the beginning or date of the condition or event that is being described. Instead, it is simply a record or notation of something that has occurred. For example, if diabetes is recorded as a fact in the database at a particular time, it does not necessarily mean that the patient developed diabetes at that

moment; it only indicates that a diagnosis was documented at that time, and there could be multiple diagnoses of diabetes for the same patient over time.

The fact table holds basic information about the observation, including the patient and provider numbers, a concept code for the observed concept, a start and end date, and other parameters explained in this document. In i2b2, the fact table is known as OBSERVATION_FACT, which encompasses all patient observations such as diagnoses, procedures, medications, and laboratory test results. For large medical institutions with millions of patients, the number of rows of observations in this table could reach billions.

Dimension tables contain further descriptive and analytical details about the attributes in the fact table. A dimension table may include information about how specific data is organized, such as a hierarchy that can be utilized to categorize or summarize data. In the i2b2 data mart, there are five dimension tables: PATIENT_DIMENSION, CONCEPT_DIMENSION, VISIT_DIMENSION, PROVIDER_DIMENSION, and MODIFIER_DIMENSION, which provide supplementary information about the fields in the fact table.

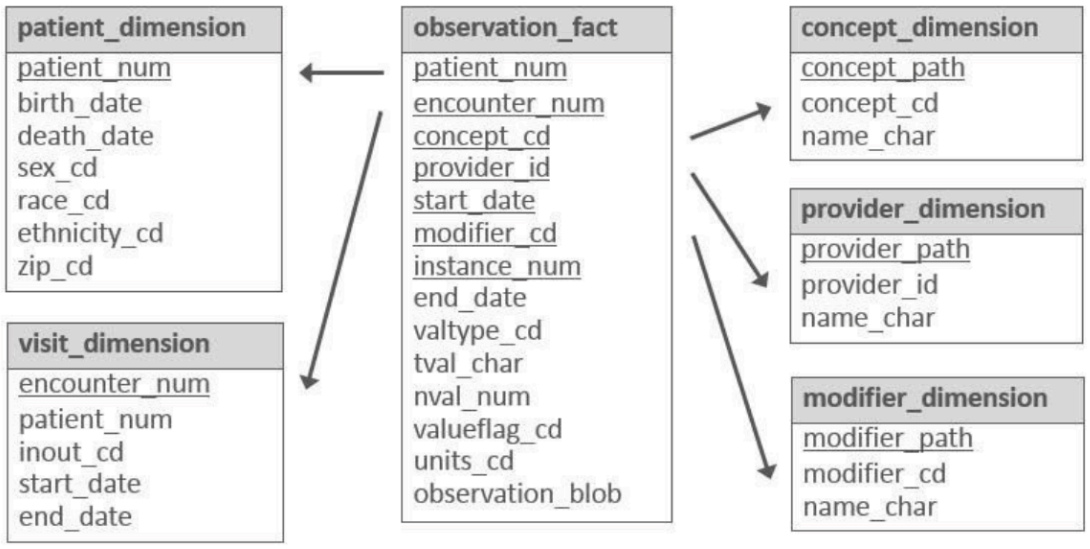


**Figure 5: Main tables in the i2b2 star schema**[10]

The i2b2 data is typically stored in a relational database, such as Oracle or SQL Server, using a star schema format. This schema comprises a fact table and several dimension tables. The fact table contains quantitative or factual information, whereas the dimension tables provide additional descriptors that further characterize the facts. These facts are identified using concept codes, and the hierarchical arrangement of these codes, along with their descriptive terms and other relevant information, is collectively referred to as the i2b2 ontology or metadata.

i2b2 ontology data can be organized into a single table that includes all possible data types or categories, or into multiple tables, with each table representing a specific data type. Examples of data types include diagnoses, procedures, demographics, lab tests, encounters, providers, health history, transfusion data, microbiology data, and genetics data. All metadata tables must follow the same basic structure. This structure plays a crucial role in visualizing concepts in the i2b2 workbench and querying the data.

The ONT cell, which is a core component of i2b2 Hive, manages vocabulary definitions and provides semantic meaning to data by containing concepts and information about relationships between them. Vocabularies in the ONT cell are arranged in a hierarchy that represents the relationship between terms, with the top levels being the "parents" or "roots" and the lower levels being their "children". Categories are defined as a set of data that share a common rule or rules for querying against the Clinical Research Chart (CRC) and are usually displayed visually as a table of terms. The ONT cell can incorporate vocabularies from different sources, which are identified by unique prefixes, and each distinct vocabulary and their associated codes are jointly called a scheme.

## 4.2 PCORNET COMMON DATA MODEL

During the Obama administration in 2010, the Patient Protection and Affordable Care Act (ACA) was enacted to enhance healthcare outcomes and reduce costs. The focus of this act was on improving insurance coverage for lower-income groups. To provide patient-centered care, the non-profit organization Patient-Centered Outcomes Research Institute (PCORI) was established through ACA, which aimed to achieve health care outcomes at personal population levels.[11] The United States Congress funded PCORI through the Patient-Centered Outcomes Research Trust Fund (PCORTF). Since health, disease patterns, and outcomes depend on multiple factors, it is necessary to consider all possible situations before estimating treatment and treatment outcomes. Comparative effectiveness research (CER) addresses these issues by comparing current healthcare mediations and analyzing how to provide healthcare for different individuals. To enhance

healthcare professionals' decision-making, PCORI opted for this method to achieve better healthcare outcomes.

In today's world of participatory medicine, involving patients in their healthcare is crucial to improve healthcare systems. In 2013, PCORI established the National Patient-Centered Clinical Research Network (PCORnet) to have an integrated view. PCORnet was built on the vision provided by many other networks[12]. To gain knowledge of community views and interests, PCORI involved 8 Patient-Powered Research Networks (PPRNs) and Clinical Research Networks (CRNs), which perform extensive studies on existing Electronic Health Record (EHR) and other data sources. Moreover, to maintain the integrity among networks, it is always essential to have policies and plans. This network of networks, PCORnet, involved two of the networks that excelled in health plans.

### 4.2.1 PCORnet: A Network of Networks

PCORnet provides a well-established ecosystem for research platforms, which is flexible in expanding connections and sharing highly valued research data to benefit public health. This national-level network of networks facilitates a platform where Clinical Research Networks (CRNs) can better answer researchers' questions. The process of questioning and answering is depicted in Figure 6: PCORnet Research Data Request Process. By using this system, PCORnet can positively influence public health issues and improve people's lives. Patient-Powered Research Networks (PPRNs) within the network offer better study designs and ideas from patients' perspectives. These studies primarily focus on the patients' questions about their health conditions and healthcare needs. Collaboratively, PCORnet provides better support for researchers and healthcare

professionals by involving patients in achieving better patient-centered care. In summary, PCORnet's network of networks offers a powerful platform for collaborative research, data sharing, and patient involvement in healthcare.

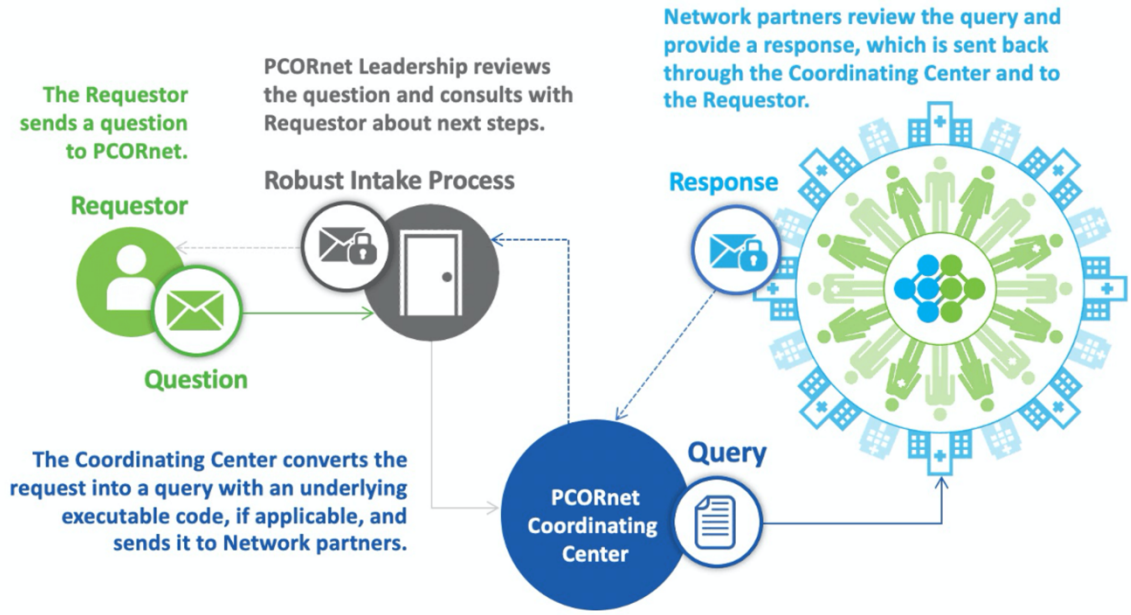


**Figure 6: PCORnet Research Data Request Process**[12]

### 4.2.2 Common Data Model (CDM)

The objective of PCORnet was to enhance patient-centered research in the Clinical Research Networks (CRNs) and other contributors by introducing a standard data model. EHRs utilized by institutions within or outside the organization have different data models, resulting in the challenge of integrating data from multiple sources. To tackle this issue, a universally compatible and system integrated data model was required, and PCORnet's Common Data Model (CDM)[13] offered a widely used solution that contains the most valuable clinical information. The PCORnet CDM v6.0 consists of 23 tables that represent various aspects of a patient's clinical and demographic information. These tables are

connected using primary and foreign keys to maintain the integrity of the data, and HL7 data standards are applied to deal with missing values. The CDM also incorporates actual dates and a "Patient Identifier" to represent the unique population of each patient, allowing for easy navigation and aggregation of data for analysis. The entities involved in the model are clearly illustrated in the Figure 7.

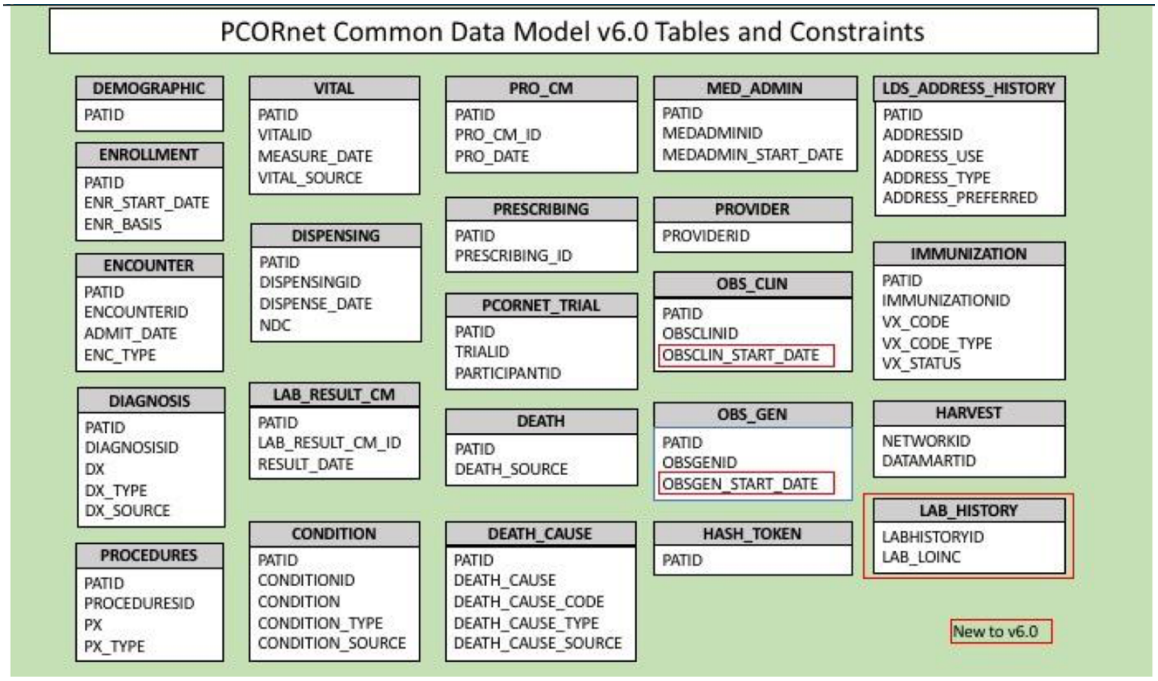


**Figure 7: PCORNET COMMON DATA MODEL TABLES v6.0** [13]

4.2.2.1 Demographic and Provider

The "Demographic" table in the PCORnet CDM adopted the US census format to ensure compatibility and consistency in recording patient information. This table primarily captured non-sensitive patient information. However, for sensitive information such as a patient's full name and Social Security Number (SSN), separate tables like "Private Demographic" were utilized. Access and release of information from these tables require

a valid Institutional Review Board (IRB) approval and a Health Insurance Portability and Accountability Act (HIPAA) waiver to ensure patient privacy and confidentiality.

In the "Provider" table, information regarding physicians was recorded, including their name, specialty, and NPI national provider identifier. This table serves to ensure the accuracy and completeness of physician information for patient-centered research purposes.

#### 4.2.2.2 Encounter and Enrollment

The PCORnet CDM includes an encounter table that captures data on patient visits to different physicians on a given day under a unique encounter identifier. However, a patient is assigned a different identifier for each visit on a different day, potentially resulting in multiple encounter identifiers for a single patient. The table also captures information on the type of encounter, admission, discharge, facility, payer, and relevant dates. It is important to exercise caution while linking this table to others since a patient may have multiple rows based on the number of diagnoses or procedures given on the same day.

The enrollment table contains information on the patient's insurance and payer details. This table maintains a single record for each patient, except for when there is a break in coverage. Even if the coverage is disrupted for a single day, a new record is assigned for that individual.

#### 4.2.2.3 Diagnosis and Procedure

In the diagnosis table, patient diagnosis information was organized and represented using a separate column to distinguish the type of diagnosis code used, such as ICD-9,

ICD-10, ICD-11, or SNOMED CT. The data in this table also included the source of the diagnosis, which was differentiated into admitting, interim, final, or discharge. This differentiation was important since the data mostly came from definitive or discharge diagnoses. The table also included columns for primary diagnosis and diagnosis origin to differentiate the different kinds of diagnoses.

In the procedures table, various procedure codes such as ICD-9 CM, ICD-10 PCS, ICD-11 PCS, LOINC, CPT/HCPCS, NDC, and revenue, and 'other' for internally used ontologies were used. Dates related to procedures were also captured, and entities like patient identifier, encounter identifier, provider identifier, principal procedures, and procedure source were included. By having this information recorded separately, the data can be analyzed more accurately, making it a valuable resource for research.

#### 4.2.2.4 Vitals, Lab Results, and Condition

The PCORnet model includes data related to patient-reported conditions, vitals, and lab results. Vitals, such as height, weight, temperature, and blood pressure, were recorded in the Vitals table. The source of the data, including patient self-reporting and personal devices, was captured alongside tobacco usage information. Multiple rows were created for multiple readings taken in a single day. In the Lab Results table, LOINC codes represented the results, while local codes were used for other labs with a name. The specimen type and details, resulting units, modifiers, and procedure type used were also recorded, along with all relevant dates. Additionally, the Conditions table contained non-coded data, including medical history, to provide researchers with insight into patients' existing health conditions and the primary reason for their encounter. The diagnosis codes

used in the diagnosis table were also utilized in this table, with a separate column indicating the condition's status as active, inactive, or resolved.

#### 4.2.2.5 Death and Death Cause

The PCORnet data model offers valuable insights on mortality through two distinct tables. The first table contains information on the sources of the data, including SSN, state or national death indices, tumor data, and local definitions. The second table, the death cause table, includes details on the cause of death, such as ICD-9 and ICD-10 codes and the type of cause. By leveraging these tables, researchers can gain a better understanding of mortality trends and factors impacting mortality in the studied population.

#### 4.2.2.6 Medications

The PCORnet data model has various tables to capture information about prescribed drugs, including the Med_admin, Prescribing, and Dispensing tables. All three tables contain data on dosage units, routes, and sources. However, there are some differences between them. The dispensing table, for instance, captures data from hospital pharmacies or mail orders, while the prescribing table is derived from providers' orders and contains inpatient and outpatient order information. The prescribing table is more detailed, providing information on the quantity, refills, and other relevant aspects. The drugs are coded based on Rx_norm. Additionally, the Med_admin table captures data on patients' medications in inpatient, outpatient, or in-home healthcare settings.

### 4.2.3 University of Missouri's (MU) CDM

One of the affiliated networks under many PCORnet CDRNs is the Greater Plains Collaborative (GPC). This network comprises 13 healthcare institutions, including the University of Missouri (MU)[12]. By utilizing the existing PCORnet Common Data Model (CDM) and MU's Electronic Health Record (EHR) data, a common data model was implemented. MU's CDM consists of 15 tables and includes supplementary private tables. Unlike the PCORnet CDM, MU's CDM does not use NDC and Revenue codes for procedure coding. The data for this CDM was extracted from a variety of sources, including the IDX billing, Cerner Millennium, Outpatient Pharmacy, EFCC cancer registry in NAACCR format, Social Security Death Master File, and Trial databases from the university's REDcap.

The CDM created by MU can be requested and used by researchers to gain insights and develop strategies for better healthcare outcomes. The data can be utilized for various research purposes, including population health research, clinical trial research, and translational research. Researchers can also use the data to develop machine learning algorithms and predictive models to improve patient outcomes. The implementation of the common data model provides a standardized approach to data analysis across the network, allowing researchers to access a vast pool of data while ensuring consistency and accuracy in the analysis process.

**Table 7 Data Lake Statistics**

| Table Name | Records | Patients |
|---|---|---|
| Demographic | 2,153,187 | 2,153,187 |
| Encounter | 25,473,447 | 1,223,013 |
| Diagnosis | 68,075,935 | 1,136,359 |
| Procedures | 46,191,968 | 1,023,647 |
| Death | 356,468 | 304,185 |
| Address_history | 1,732,642 | 1,581,578 |
| Obs_clin | 466,719,942 | 966,335 |
| Lab_result_cm | 166,754,971 | 590,513 |
| Prescribing | 87,385,007 | 912,696 |
| Provider | 30,270 | - |
| Med_admin | 39,470,457 | 523,859 |

Table 7 provides a detailed breakdown of the data counts for MU's CDM. The CDM comprises 2,153,187 distinct patient records, with 1,136,359 records having a diagnosis and 1,023,647 records having received treatment procedures. A total of 304,185 records indicated that the patients had passed away, and 1,732,642 records had address histories available. The CDM captured data from 30,270 providers who administered medications to 523,859 patients. Encounters were reported for only 1,223,013 patients, but the total number of encounters captured was 25,473,447, indicating that many patients had multiple visits. The total number of diagnoses and procedures recorded was 68,075,935 and 46,191,968, respectively, indicating that many patients suffered from multiple diseases and had undergone various treatments.

## 4.3 I2B2 ACT ONTOLOGY

The ACT Network ontology is a vital component of the ACT Network, a national research infrastructure that connects researchers with a wealth of clinical and translational research data.[14] The ontology provides a standardized vocabulary for defining clinical concepts and enables researchers to query the data in a consistent and meaningful way. The ACT ontology includes over 30,000 concepts and over 1.4 million relationships between them, covering a wide range of domains, including diagnoses, procedures, medications, and patient characteristics.

One of the primary benefits of the ACT Network ontology is its ability to facilitate semantic interoperability between different EHR systems. By providing a standardized vocabulary, researchers can combine and compare data from multiple sources, irrespective of their EHR systems or coding systems within the same EHR system. This interoperability significantly enhances the pool of data available to researchers, empowering them to conduct more comprehensive research studies.

The ACT Network ontology is continuously evolving and expanding to keep up with the latest advancements in medical research and clinical terminologies. With regular updates reflecting feedback from users and changes in medical terminology, the ontology stays up to date with the latest developments in the healthcare community. The ACT Network has developed an i2b2 ACT ontology package that allows researchers to query on patient data using ACT ontology within the i2b2 platform. We have mapped the UM's PCORnet CDM data with the i2b2 data model using ACT ontology. In Table 8 Overview of ACT Ontology Terms, we present a comprehensive view of the ACT ontology terms

available in the i2b2, along with their corresponding coding systems such as ICD10CM, LOINC, and RXNorm and count of distinct concepts associated with each table.

**Table 8 Overview of ACT Ontology Terms**

| Ontology Tables | Coding Systems | Concepts Count |
|---|---|---|
| ACT Diagnoses ICD-10-CM | ICD10CM | 96,366 |
| ACT Diagnoses ICD-9-CM | ICD9CM | 17,753 |
| ACT Diagnoses ICD10-ICD9 | ICD9CM, ICD10CM | 128,152 |
| ACT Procedures ICD-10-PCS | ICD10PCS | 244,334 |
| ACT Procedures ICD-9-Proc | ICD9PROC | 4,676 |
| ACT Procedures HCPCS | HCPCS | 9,186 |
| ACT Procedures CPT-4 | CPT4 | 16,564 |
| ACT Laboratory Tests | LOINC | 547 |
| ACT Laboratory Tests (Provisional) | LOINC | 142,860 |
| ACT Medications Alphabetical | NDC, RXNORM | 1,059,167 |
| ACT Medications VA Classes | NDC, RXNORM | 1,244,825 |
| ACT COVID-19 | LOINC, CPT4, HCPCS, SNOMED, NDC, ICD10PCS, RXNORM, ICD9CM | 66,451 |
| ACT Vital Signs | LOINC | 85 |
| ACT Social Determinants of Health | LOINC | 22 |

## 4.4 MAPPING ACT ONTOLOGY WITH PCORNET CDM DATA

The i2b2 uses a hierarchical organization for its ontology, comprising multiple levels. The topmost levels represent main categories e.g., Clinical Observations, Demographics, Diagnosis, and Procedures, which are further divided into more specific concepts. For instance, the 'Diagnosis' ontology encompasses concepts like Infectious Diseases, Cardiovascular Diseases, and Neoplasms etc.

A unique identifier, called a concept code, is assigned to every concept in the i2b2 ontology. These codes represent the concept in the i2b2 database and are utilized in data retrieval queries. Additionally, each concept has associated metadata that describes its characteristics, including its permissible values, data type, and relationships with other concepts in the ontology.

The i2b2 ontology provides an adaptable and robust framework for representing and querying clinical data in a standardized and interoperable manner. It is built to accommodate the addition of new concepts as necessary to support different types of clinical data. This is done using ontology mapping, a process that links new concepts to existing ones based on their semantic similarity.

### 4.4.1 The i2b2 Data Installer Package

The i2b2 Data Repository is a part of the i2b2 software suite and the source code is stored in a GitHub repository (https://github.com/i2b2/i2b2-data). The i2b2 Data Repository contains a collection of SQL scripts those create all the required tables and schemas (Table 9) for i2b2 database. The installer also provides instructions for executing the SQL scripts using JDBC driver.

**Table 9: Overview of i2b2 database schemas**

| i2b2 Schema | Description of Tables |
|---|---|
| I2B2DATA | It consists of various tables that organize and store clinical data in a standardized format. It comprises a collection of dimension tables, fact tables as well as additional tables required for CRC cell. |
| I2B2PM | It stores all the tables required for functioning Project Management (PM) cell.<br>It contains information about cells, projects, users, and users' access to the project. It also stores user login history in different tables. |
| I2B2METADATA | It stores i2b2 ontology tables and other tables required for accessing the ontology. |
| I2B2HIVE | It is a part of Project Management (PM) Cell. The i2b2 hive schema consists of tables that represent server configuration parameters and contain information about other cells. |
| I2B2WORKDATA | It manages users' workplaces, information about queries saved in the workplaces in different tables. |

### 4.4.2 ETL Pipeline for Mapping PCORnet CDM Data with ACT Ontology

The i2b2 data installer package supports 3 different SQL engines (Oracle, Microsoft SQL Server, PostgreSQL) as i2b2 back-end database. However, we modified the i2b2 core services to enable Snowflake as i2b2 database as described in Chapter 5. In this process, we modified the i2b2 data installer and added additional supports to use Snowflake as i2b2 database. We have forked the i2b2 data installer repository (https://github.com/Missouri-BMI/i2b2-data) and added new SQL scripts under snowflake folder structure that create all the required i2b2 tables and schemas in the snowflake database. We have used the 'ACT' version of the installer to create required tables and schema for ACT ontology along with the other tables and schemas required for each cell. We have added new SQL scripts that are compatible with Snowflake while retaining the

same table definitions. The modification added another SQL engine (Snowflake) compatibility to i2b2 data installer package. After creating all the required schemas and tables in the Snowflake database using the forked i2b2 data installer, we created ETL (extract, transform, load) pipeline in the GitHub repository (https://github.com/Missouri-BMI/I2B2_ON_P_CDM) to extract data from PCORnet CDM database, transform the tables to i2b2 fact tables and dimension tables and finally load the transformed data into i2b2 database in fact and dimension table format. We have docker containerized the ETL process and populated the i2b2 fact tables and dimension tables by running the docker container. The docker container also modify the 'C_DIMCODE' columns of each ACT ontology tables and replace the snowflake unsupported SQL commands with supported SQL commands. We developed python scripts that read 'C_DIMCODE' column of each ontology table and identify the unsupported code using regular expression then finally replace the unsupported SQL commands with supported SQL commands. Table 10 shows the list of i2b2 ACT ontology hierarchies and the fact tables linked to them, whose are generated from the associated deidentified PCORNET CDM tables.

**Table 10: Mapping PCORnet CDM data with i2b2 fact and dimension tables**

| ACT Ontology Hierarchy | PCORnet Table | i2b2 Table |
|---|---|---|
| ACT Laboratory Tests (Provisional) | DEID_LAB_RESULT_CM | lab_fact |
| ACT Diagnoses ICD-10-CM | DEID_DIAGNOSIS | diagnosis_fact |
| ACT Procedures HCPCS | DEID_PROCEDURES | procedure_fact |
| ACT Procedures ICD-10-PCS | DEID_PROCEDURES | procedure_fact |
| ACT Social Determinants of Health | DEID_VITAL<br>DEID_ENCOUNTER | vital_fact<br>visit_dimension |
| ACT Medications VA Classes | DEID_PRESCRIBING | prescribing_fact |
| ACT Procedures CPT-4 | DEID_PROCEDURES | procedure_fact |
| ACT Visit Details | DEID_DEMOGRAPHIC | demographic_fact |
| ACT Demographics | DEID_DEMOGRAPHIC | demographic_fact |
| ACT Procedures ICD-9-Proc | DEID_PROCEDURES | procedure_fact |
| ACT COVID-19 | DEID_DIAGNOSIS<br>DEID_PROCEDURES<br>DEID_LAB_RESULT_CM<br>DEID_PRESCRIBING | diagnosis_fact<br>procedure_fact<br>lab_fact<br>prescribing_fact |
| ACT Laboratory Tests | DEID_LAB_RESULT_CM | lab_fact |
| ACT Diagnoses ICD10-ICD9 | DEID_DIAGNOSIS | diagnosis_fact |
| ACT Diagnoses ICD-9-CM | DEID_DIAGNOSIS | diagnosis_fact |
| ACT Vital Signs | DEID_VITAL | vital_fact |
| ACT Medications Alphabetical | DEID_PRESCRIBING | prescribing_fact |

As part of our research project, we performed the ETL process to integrate data from PCORnet CDM, a standardized data model used in clinical research. To achieve this, we extracted relevant PCORnet CDM tables (Table 10) using SQL queries and then created intermediate tables from the extracted data (Figure 8). These intermediate tables contain columns required for i2b2 fact and dimension tables that we intended to construct. Then we created views which has exact table definition of i2b2 fact and dimension tables from the intermediate tables. We have created '.sql' scripts to perform the ETL process for each table in PCORnet CDM (Table 10). These scripts contained SQL commands that were used to extract the relevant data from each table, transform it into the required format, and load

the data into corresponding intermediate tables. The Figure 8 shows the ETL process of Diagnosis table where, we have used DX_TYPE (’10’, ‘9’) and DX (‘K40.30’) column values to generate the ‘concept_cd’ column (‘ICD10CM:K40.30’) in i2b2 fact table.

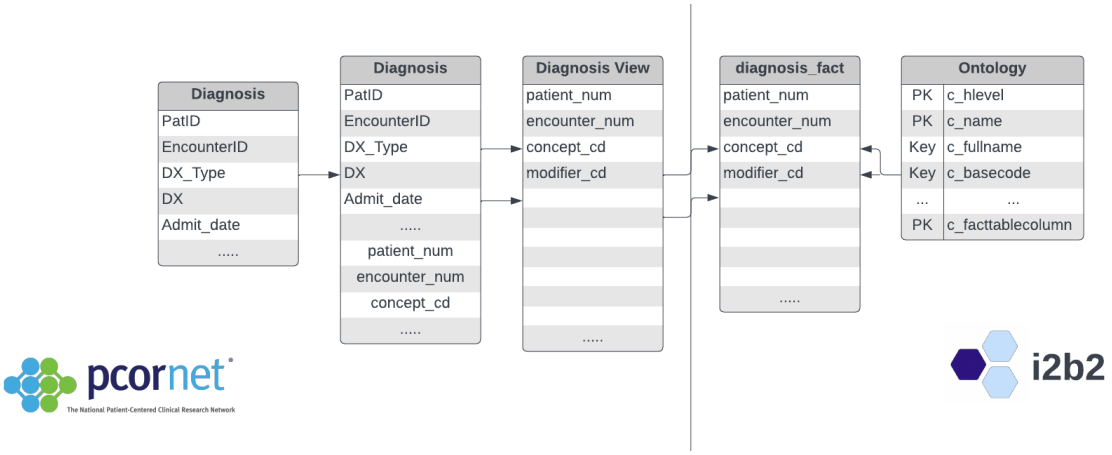


**Figure 8: Example of Mapping i2b2 fact table with PCORnet CDM table**

We have also mapped the result columns in the PCORnet CDM lab results table with the i2b2 fact table columns (Figure 9).

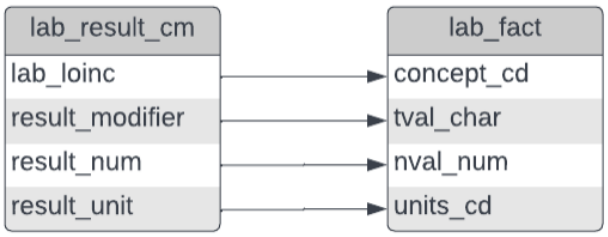


**Figure 9: Example of Mapping lab results**

Finally, we have populated the i2b2 multi-fact and dimension tables with data from the views, using SQL insert statements. By performing the ETL process, we were able to

integrate data from PCORnet CDM into our i2b2 database, facilitating further analysis and interpretation of the data.

### 4.4.3 Multi-Fact View Approach

The mapping process was designed to integrate the legacy i2b2 data model, which represents patient data through both fact and dimension tables. The legacy i2b2 data model is composed of a central fact table called 'observation_fact' that is accompanied by numerous dimension tables. However, the multi-fact table feature in i2b2 enables queries to be performed across more than one fact table. In PCORnet Common Data Model, there are multiple fact tables distinguished by domains like Labs, Diagnosis, Procedures, Medications, Vitals (Figure 10).

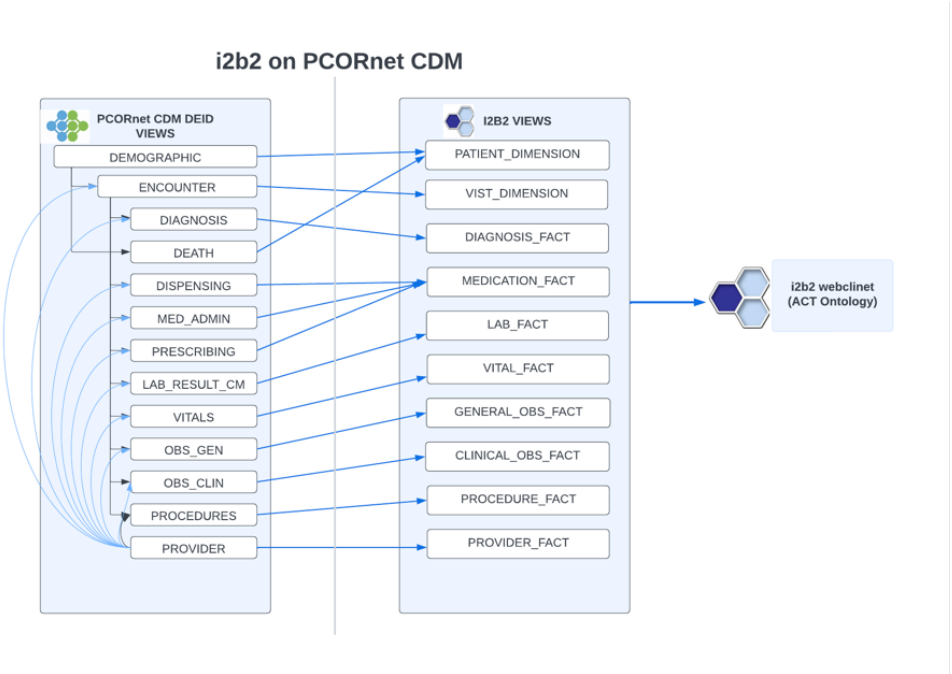


**Figure 10: Mapping i2b2 Fact and Dimension Views with PCORnet CDM Views**

Since i2b2 fact tables and dimension tables doesn't changes without the ETL data refresh, we built i2b2 transformation scripts that create views for i2b2 fact and dimension tables on PCORnet CDM tables so that fact and dimension views in i2b2 automatically changes when the source tables in PCORnet CDM changes. Multiple fact and dimension views are created for one or more domains in the PCORnet CDM data. The transformation scripts also update the i2b2 meta data tables using the standard terminology in the fact views and ACT V4 ontology that is available i2b2 version 1.7.13. These i2b2 transformation scripts can be used by all the PCORnet sites to stand up an i2b2 instance on their existing PCORnet CDM data.

## 4.5 SUMMARY

We were able to successfully perform the data installation operation to populate the i2b2 database. The operation performs 3 different tasks: 1) generates the fact and dimension tables, 2) execute patient count generator scripts for each i2b2 ontology hierarchy and 3) identifies the gaps in mapping PCORnet CDM with ACT ontology and generates report in tabular format and store in the i2b2 database under 'Report' schema. The Figure 11 represents the i2b2 web client is displaying the i2b2 ACT ontology hierarchies (Table 10) where numbers, highlighted in the red colors are the patient counts generated by the patient count generator scripts.

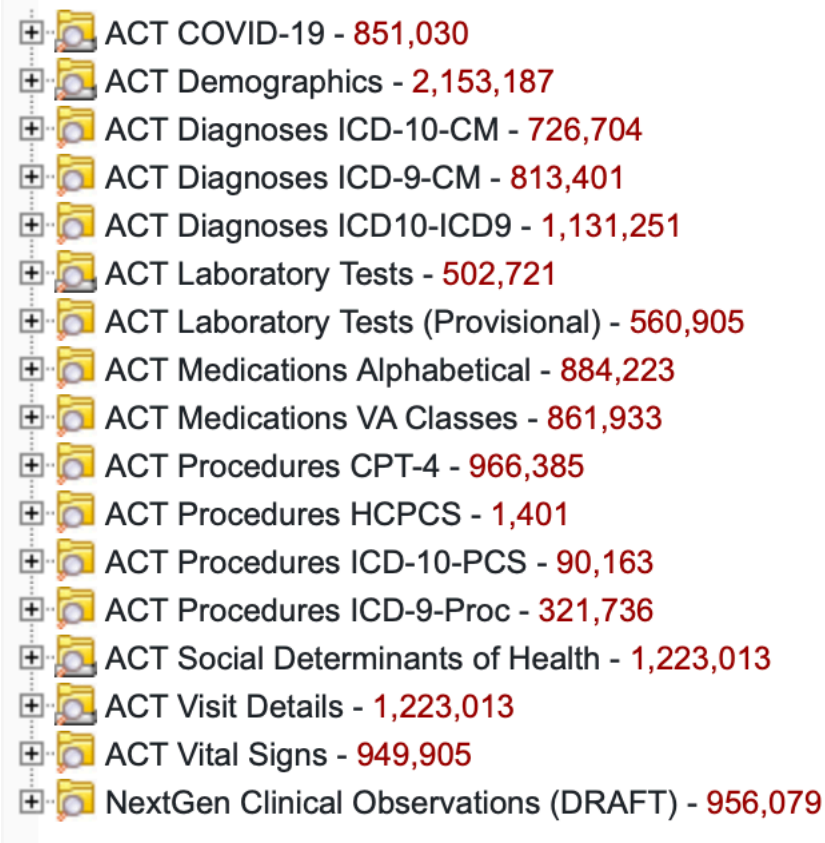


**Figure 11: ACT ontology in i2b2 web client user interface**

We have created a i2b2 project named 'NextGen Data Lake De-Identified' in the i2b2 hive and populated the I2B2DATA, I2B2METADATA schema (Table 9) tables with the University of Missouri PCORnet CDM tables, converted into i2b2 fact and dimension tables. Table 11 represents the count of total rows of data and distinct patient counts for each i2b2 fact and dimension table stored in the i2b2 database. It holds 1.26 billions of observation facts of 2.1 millions patients, extracted from MU's PCORnet CDM.

**Table 11: Summary of i2b2 fact and dimension tables**

| Table Name | Row Count | Patient Count |
|---|---|---|
| Demographic_fact | 6,763,746 | 2,153,187 |
| Diagnosis_fact | 65,135,834 | 1,136,359 |
| Lab_fact | 154,233,375 | 586,435 |
| Obsclin_fact | 368,194,029 | 956,079 |
| Patient_dimension | 2,153,187 | 2,153,187 |
| Prescribing_fact | 57,884,178 | 899,365 |
| Procedure_fact | 46,141,683 | 1,023,647 |
| Provider_dimension | 30,270 | 30,270 |
| Visit_dimension | 25,473,447 | 1,223,013 |
| Visit_fact | 75,506,671 | 1,223,013 |
| Vital_fact | 460,716,400 | 966,334 |

# CHAPTER 5 : SNOWFLAKE AS I2B2 BACKEND DATABASE

The i2b2 Core Server (https://github.com/i2b2/i2b2-core-server) is written in Java and utilizes several open-source libraries and frameworks, including Hibernate, Spring Framework, and Apache Axis2 etc. It also supports multiple databases, including PostgresSQL, Oracle, and Microsoft SQL Server using JDBC drivers. As the MU's Data Lake as described in the section 4.2.3, is hosted in the Snowflake Data warehouse, we goal was to integrate the i2b2 databases with the same warehouse. This would help in minimizing the extra costs associated with storage, as well as the effort required for maintaining and administering another database server. We intended to use a cloud database provider for the i2b2 database and chose Snowflake data warehouse because of its extensive cloud features and support for JDBC driver. It also shown promising performance in running i2b2 queries in Snowflake on larger population datasets. Snowflake implements JDBC type 4 driver interface that supports core JDBC functionality. The driver can be used with most client tools/applications that support JDBC for connecting to a database server. We have forked the i2b2 core server repository (https://github.com/Missouri-BMI/i2b2-core-server/tree/feature/snowflake). and modified the source code to implement JDBC as an additional data source for i2b2 web services running inside Wildfly.

## 5.1 SNOWFLAKE DATA WAREHOUSE

Snowflake is a cloud-based data warehousing platform that allows storing, processing, and analyzing large amounts of data (Table 12). It provides fast query processing by using multiple virtual warehouses, automatic query optimization, cluster tuning, micro-partitioning, automatic clustering, and re-clustering of tables, and quickly scales up and down (both vertically and horizontally) without disruption. Snowflake also eliminates the time and effort of managing data warehouse infrastructure. Snowflake offers advanced features such as SQL support for complex queries and joins, compatibility with JSON, semi-structured and structured data, and supports various data ingestion options including batch, streaming, and real-time data ingestion. Additionally, it provides end-to-end encryption, granular access controls, data masking, and supports various compliance certifications like HIPAA, making it a highly secure and versatile option as an application backend database.

Snowflake also provides support for JDBC (Java Database Connectivity), which allows Java applications to connect to Snowflake as a data source. The JDBC driver provided by Snowflake offers a high degree of compatibility with standard Java database APIs, making it easy to integrate with existing Java applications.

**Table 12: Key Features of Snowflake Data Warehouse**

| Feature | Description |
|---|---|
| Security, Governance, and Data protection | Snowflake provides a multi-layered security model, encryption, access controls, RBAC, MFA, auditing, and logging capabilities to ensure the confidentiality, integrity, and availability of data. It complies with security and data protection regulations and standards such as SOC 2, HIPAA, GDPR, and PCI. Snowflake also offers advanced data governance features, such as data retention policies, data lineage tracking, and data masking, to help organizations comply with regulatory requirements and internal policies. |
| Standard and Extended SQL Support | Snowflake provides full SQL support, including standard SQL as well as extensions to support more complex queries and data types. It also supports advanced SQL features such as stored procedures, user-defined functions, and recursive queries. It offers a wide range of data types, including structured, semi-structured, and unstructured data. |
| Scalability | Snowflake data warehouse is designed to be highly scalable and elastic. It separates compute and storage, which allows organizations to scale each independently, and uses instant elasticity to quickly add or remove compute resources as needed. This makes Snowflake a flexible and cost-effective data warehousing platform that can handle various workloads. |
| Near-Zero Administration | Snowflake is a cloud-based data warehousing platform that aims to simplify data management and maintenance by minimizing the need for physical hardware, software installation, and configuration. Snowflake automates most administrative tasks, such as performance tuning, software upgrades, and data protection. With Snowflake's tools for monitoring query performance and optimizing queries, organizations can improve the efficiency of their analytics workloads |
| Optimized Storage | Snowflake uses its internal optimized, compressed, columnar storage to store and retrieve data in a highly efficient manner. In columnar storage, data is organized and stored by column rather than by row, which allows for more efficient processing of analytical queries. In addition, columnar storage is highly compressible, which means that it takes up less space on disk and in memory than traditional row-based storage. This results in reduced storage costs and faster data retrieval times. |

## 5.2 SNOWFLAKE JDBC DRIVER SUPPORT FOR I2B2 CORE CELLS

We have made significant changes to the i2b2 core server GitHub forked repository (https://github.com/Missouri-BMI/i2b2-core-server/tree/feature/snowflake) by implementing JDBC integration for Snowflake, a popular cloud-based data warehousing and analytics platform. With this integration, the i2b2 core services can now easily access and manipulate data stored in Snowflake, using the Java Database Connectivity (JDBC) driver. We also implemented i2b2-data loader in Snowflake for PCORnet common data model using the ACT ontology, which is described in the chapter 4.4.

The i2b2 core services are a collection of Java-based applications that are designed to provide researchers and analysts with a powerful and flexible platform for managing and analyzing data. At the heart of the i2b2 core services is the Data Access Object (DAO) layer, which is responsible for encapsulating the logic required to access and manipulate data from the database. The i2b2 core services use the Java Native SQL (JDBC) library, which provides a standard API for connecting to and executing SQL statements against databases. In addition to JDBC, the i2b2 core services also utilize the JDBC Spring Framework in their DAO layer. The Spring Framework provides a set of tools and utilities for building robust and efficient data access layers, which greatly simplify the development and maintenance of the i2b2 core services. Together, these frameworks enable the i2b2 core services to communicate with databases and perform a wide range of database operations, including reading and writing data.

The Data Access Layer (DAO) of the i2b2 services provide support for three different SQL engines: Oracle, Microsoft SQL Server, and PostgreSQL. This allows users

to choose the database system that best suits their needs and preferences. However, it is important to note that not all JDBC drivers, including the Snowflake JDBC driver, support all types of query statements. For example, certain data types, built-in function may not be supported by all drivers, and the creation of sequences and temporary tables can be different for different JDBC driver types. Data Access Layer classes in the i2b2 services handles exceptions that may occur with the queries not supported by the JDBC driver in conditional logic blocks written in the class implementation. The conditional logic blocks include changes in data types for query parameters and the ResultSet objects for different JDBC data sources. As part of our effort to expand the capabilities of i2b2 service, we have introduced new conditional logic blocks in every DAO class implementation where necessary to ensure that the JDBC Snowflake driver is supported. This ensures that the system remains robust and functional, even with the integration of a new data source.

### 5.2.1 Snowflake Data Source in Wildfly

Snowflake provides a JDBC type 4 driver that supports JDBC functionality. The JDBC driver must be installed in a 64-bit environment and requires Java 1.8 (or higher). Our docker containerization process describe in the section 2.3 ensures the environment compatible for the Snowflake JDBC driver. The docker implementation of 'i2b2-wildfly' execute a shell script to install the JDBC driver module and configure jdbc connection-pool for each service in the container runtime using jboss-cli (jboss command-line interface). We have used snowflake jdbc driver version 3.13.9 and added the driver as a module in the Wildfly using following jboss-cli command:

```
module add --name=net.snowflake --
resources=/opt/jboss/wildfly/customization/snowflake-jdbc-3.13.9.jar
```

Then we have added snowflake driver as a data source for the Wildfly server by following command:

```
/subsystem=datasources/jdbc-driver=snowflake:add(driver-
name="snowflake",driver-module-name="net.snowflake",driver-class-
name=net.snowflake.client.jdbc.SnowflakeDriver)
```

Finally, we have added connection-pool for each i2b2 services using their jndi-name (Java Naming and Directory Interface). The below command was executed to incorporate a snowflake connection-pool into the i2b2 Data Repository (CRC) service:

```
data-source add
--jndi-name=java:/CRCBootStrapDS
--name=CRCBootStrapDS
--connection-url=${DS_HIVE_DB}
--driver-name=snowflake
--user-name=${DS_HIVE_USER}
--password=${DS_HIVE_PASS}
--max-pool-size=200
```

We pass the following arguments as the function parameter listed in the Table 13. The configuration of environment variables of the data source was covered in chapter 2.3.1.

**Table 13: Arguments used in configuring the Wildfly Data source**

| Function Argument | Description |
|---|---|
| jndi-name | The name of a resource in the Java Naming and Directory Interface (JNDI) naming service. The i2b2 jndi resource are: CRCBootStrapDS, QueryToolDS, OntologyBootStrapDS, OntologyDS, PMBootStrapDS, WorkplaceBootStrapDS, WorkplaceDemoDS |
| name | The name of a resource or subsystem. It is used with the jboss-cli commands to perform operations on a specific resource or subsystem |
| connection-url | Connection URL for the database |
| driver-name | Unique identifier of the database |
| user-name | Username credential for the database account |
| password | Password credential for the database account |
| max-pool-size | Maximum database connection can contain the data source |

### 5.2.2 Modifying Data Access Layer classes in the i2b2 services:

The i2b2 core services abstracts the creation of JDBC driver connection in its 'DataSourceLookupDAOFactory'. The DAO objects retrieve the JDBC driver identifier and construct queries and query result objects in its own class definition. We have added 'SNOWFLAKE' as a new data sources in the 'DataSourceLookupDAOFactory' so that, other DAO objects can identify Snowflake as another data source for the system. Then, we went through all the i2b2 services and modified the source code of i2b2-core-server in our forked repository. Figure 12 represents the steps were taken while modify the i2b2-core-server source code. i) The first step was to review all the i2b2 services and identify their corresponding SQL codes in the Data Access Object (DAO) class implementation. ii) The next step involved running these SQL codes in Snowflake, a cloud-based data warehousing platform, to test their compatibility and ensure that they execute without any errors. iii) If

any errors were encountered during the testing phase, a conditional logic block implementation was added in the DAO class specifically for Snowflake. The errors identified were due to differences in the implementation of creating sequence, temporary tables, unsupported data types, and different date functions used in Snowflake compared to the original implementation. The conditional logic block implementation addressed these differences in the Snowflake implementation, enabling the SQL codes to run without errors in the Snowflake environment. iv) Finally, the updated DAO class implementation was thoroughly tested to ensure that all i2b2 services were working correctly with the Snowflake database. The snowflake driver support implementation of i2b2-core-server is available in the public forked repository (https://github.com/Missouri-BMI/i2b2-core-server/tree/feature/snowflake), and it can be deployed in the cloud or locally as described in the chapter 2.3.2.

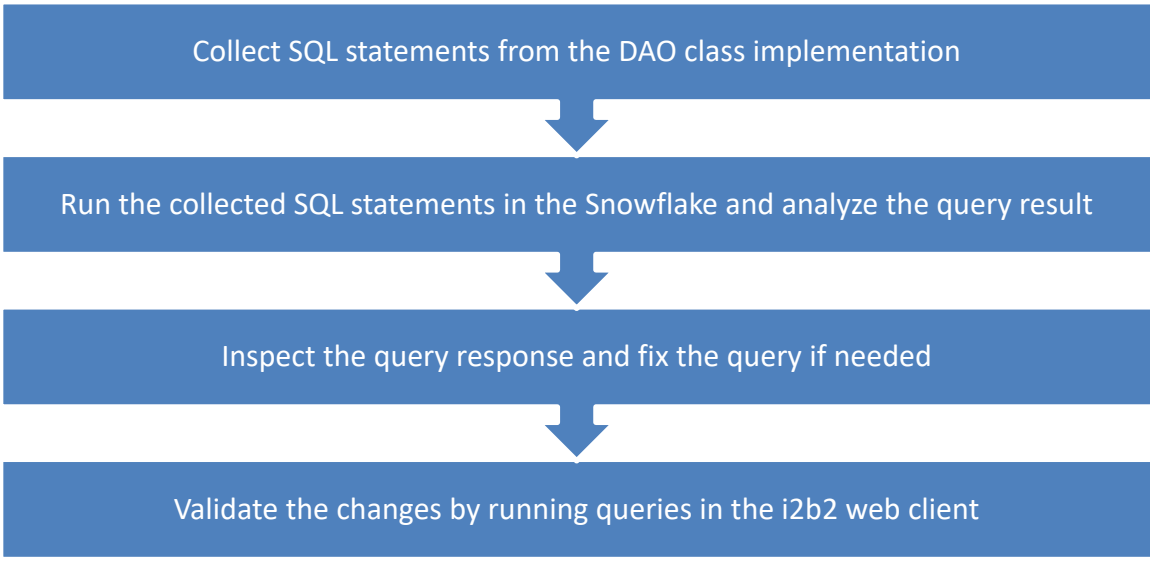


**Figure 12: Step taken in the modification of i2b2-core-server source code**

## 5.3 RESULTS

In this chapter, our goal was to leverage Snowflake as the backend database for i2b2, as it offered superior analytical query performance and test its query execution performance as i2b2 back-end database. We were able to successfully run queries in the i2b2 web client which uses snowflake as i2b2 back-end database. We have executed various types of queries (Table 14) in the i2b2 web client for different types of warehouses, query complexity and closely observed the results described in the Figure 13. We utilized the various query features available within the i2b2 query tool to construct 9 distinct types of queries (Table 14). These features include the basic query functions such as adding terms in the separate groups, adding date constraints and value constraint to filter out the observations. We can also include groups in the exclusion criteria by using exclude functionality. In addition to these features, we also included i2b2's query timing features to construct more complex queries. The i2b2 query timing features allowed us to treat groups of terms within the same financial encounter or treat independently from other groups. We also used the result types mentioned in the (Table 14) to generated result for the i2b2 queries. The i2b2 query tool provides several result types including distinct patient count, patient and encounter set for future analysis, show graphical representation of patient count distribution in various groups e.g. age, gender, race etc.

## Table 14: Overview of i2b2 queries executed in the comparison

| Query Features | | Very Simple | Simple | Medium Complex | Complex | Highly Complex | Very Complex | Temporal |
|---|---|---|---|---|---|---|---|---|
| **Query Features (Basic)** | Number of groups | 1 | 2 | 3 | 3 | 3 | 4 | 2 |
| | Number of terms | 1 | 2 | 3 | 3 | 8 | 12 | 2 |
| | Number of date constraints | 0 | 0 | 0 | 1 | 4 | 4 | 0 |
| | Number of Exclusions | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | Number of value constraints | 0 | 0 | 0 | 1 | 2 | 5 | 0 |
| **Query Features (Temporal)** | Number of groups are independent of financial encounters | 1 | 2 | 3 | 1 | 1 | 1 | 0 |
| | Number of groups are in the same financial encounters | 0 | 0 | 0 | 2 | 2 | 3 | 2 |
| **Result Features** | Patient Count | √ | √ | √ | √ | √ | √ | √ |
| | Patient set | √ | √ | √ | √ | √ | √ | √ |
| | Encounter set | √ | √ | √ | √ | √ | √ | √ |
| | Gender patient breakdown | √ | √ | √ | √ | √ | √ | √ |
| | Vital status patient breakdown | √ | √ | √ | √ | √ | √ | √ |
| | Race patient breakdown | √ | √ | √ | √ | √ | √ | √ |
| | Age patient breakdown | √ | √ | √ | √ | √ | √ | √ |
| | Top 20 diagnosis breakdown | √ | √ | √ | √ | √ | √ | √ |
| | Top 20 medications breakdown | √ | √ | √ | √ | √ | √ | √ |

The reason behind executing a range of queries under different warehouses was to evaluate the performance of Snowflake in handling varying levels of workload and determine which warehouse configuration best suited the needs of the i2b2 application. When running i2b2 queries in Snowflake, the size of the warehouse being used can impact the performance of the queries. Specifically, if the query being executed involves a higher degree of complexity, such as including multiple fact and dimension tables, a larger warehouse shows better results. In contrast, when running less complex queries, switching from a larger warehouse to a smaller warehouse may not have a significant impact on performance. It is important to note that the size of the warehouse is not the only factor that affects query performance. Other factors, such as the number of nodes, the amount of data being processed, and the complexity of the query itself, can also play a role in determining the optimal warehouse size for a given query.

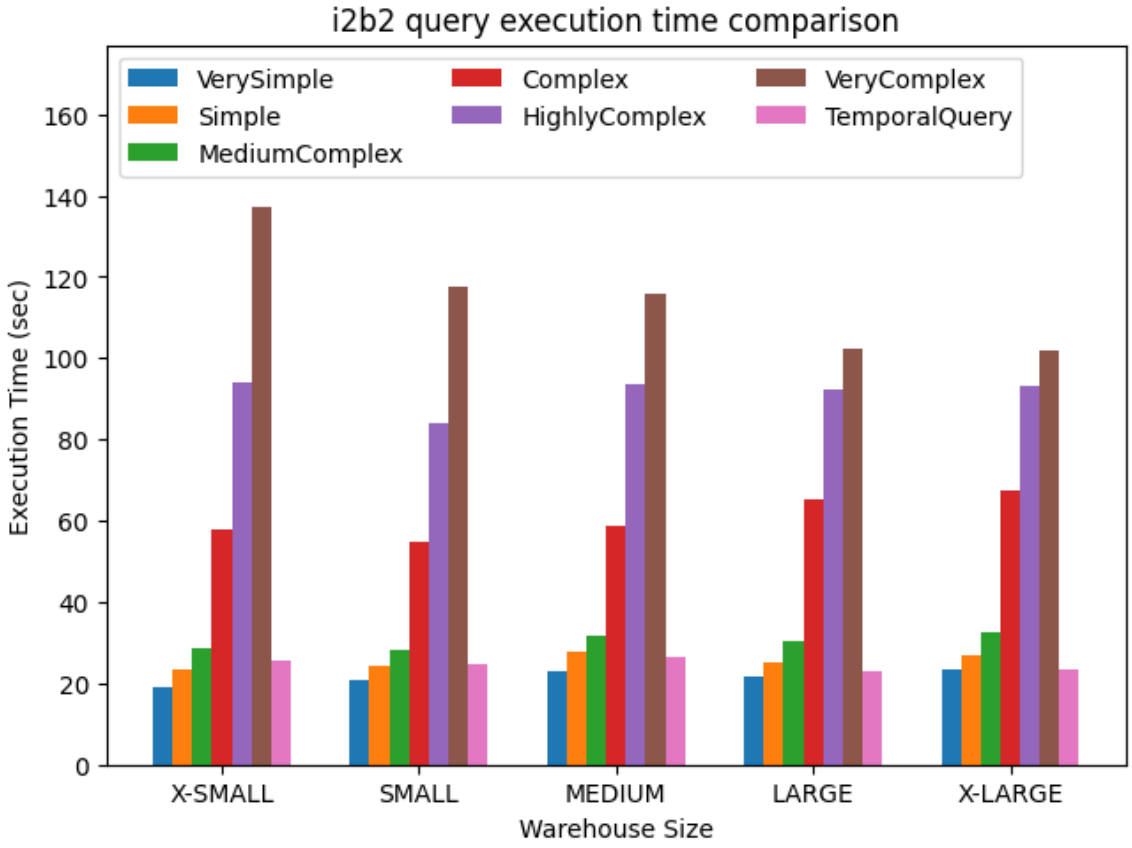


**Figure 13: i2b2 query execution time comparison against different warehouses**

# CHAPTER 6 : DISCUSSION

The aim of our research project was to accomplish several objectives. Firstly, we sought to simplify the process of installing i2b2. To achieve this, we containerized the i2b2 web client and web services in their corresponding docker containers. We created an i2b2 docker containerization package that can automatically download, compile, and configure the i2b2 platform, enabling the easy deployment of a functioning i2b2 instance in the cloud or local environment.

Next, we focused on deploying the i2b2 components in the AWS cloud, with the goal of implementing a serverless architecture solution that would allow us to host i2b2 in a secure environment with minimal effort required for maintenance. We also aimed to use Snowflake as the i2b2 database in the cloud to reduce administrative tasks involved in maintaining the server, as the MU data lake is also in Snowflake. To achieve this, we modified the source codes of i2b2 core-server and data installer repository and added Snowflake as a new data source for the system. Our solution enhances the functionality of the i2b2-core-servers and can be used in existing server configurations.

Finally, we populated the fact and dimension tables of i2b2 Snowflake database using the ETL operation we developed for mapping PCORnet data with the ACT ontology. Our ETL scripts package offers automatic installation of i2b2 data from PCORnet CDM source systems with minimal effort, by providing configuration parameters for source and target databases.

Our installation packages serve as a useful reference for quickly and easily installing i2b2, and they can be used by infrastructure teams to efficiently instantiate new instances of the i2b2 platform at different sites. One of the primary challenges of installing i2b2 is the manual collation of dependencies. Our installer packages include all the required dependencies, such as server configuration parameters, dependency packages, and database credentials for different cells.

The second challenge of installing i2b2 is configuring the platform components to connect properly. We simplified this configuration process, enabling developers to easily modify the configuration parameters to suit their environment. We also focused on ensuring that i2b2 runs in a secure environment in the cloud. To achieve this, we followed recommended practices for deploying i2b2 applications in the cloud. We deployed containers in the private subnets inside the VPC, disabled public IP association, established an application load balancer with WAF, and enforced secure HTTPS connections. We also enabled SSO login mechanism for i2b2 web client, allowing users to log in to i2b2 using their institution's login credentials.

## 6.1 SNOWFLAKE QUERY EXECUTION PERFORMANCE AGAINST LARGER DATASET

We have developed Snowflake JDBC driver support for i2b2 web services to utilize the computing power warehouses of snowflake data warehouse and its query execution performance against analytical type queries. The Snowflake i2b2 database initially loaded with MU's PCORnet CDM data. It holds 1.26B facts of 2M patients in the different fact and dimension tables. However, we wanted test the system with larger population database and evaluate its performance. We have created a project in the same

i2b2 platform and loaded i2b2 fact and dimension tables from 13 Greater Plains Collaborative (GPC) sites, stored in PCORnet CDM and tested the system by executing same query listed below in the different i2b2 projects having MU and GPC data.

| Male AND >=65 years old AND Has disease of the digestive system (K00-K95) |
|---|

We ran the query on both i2b2 project and found it took 44.0 sec to retrieve the patient count from MU data having 65 millions facts of 1.1 millions patients diagnosis records. Where it took 50.0 sec to retrieve the patient count from GPC data having 1.37 billions facts of 23 millions patient diagnosis records.

## 6.2 LIMITATIONS AND FUTURE WORK

The docker containerization process of i2b2 simplifies the installation and upgrade of i2b2 platform in a greater flexible way. Our demonstration showcased i2b2 deployment in AWS ECS FARGATE, but it is worth noting that other cloud service providers like Azure and Google Cloud Provider also provide similar capabilities. Although we chose AWS based on the requirement specific needs, also for its popularity and user-friendliness, while other cloud providers could be suitable as well. We enhanced the functionality of i2b2 services by developing Snowflake JDBC driver support i2b2 core services, which enables the i2b2 services to query directly from Oracle, Microsoft SQL Server, PostgreSQL, and Snowflake. However, our data installation package and ETL pipeline currently only support Snowflake as i2b2's backend database. In future work, we plan to

expand our data installation package to support other databases and cloud providers, making i2b2 more adaptable and accessible to a wider range of research settings.

## 6.3 CONCLUSION

In this thesis, we addressed the need for a self-service query tool for researchers and aimed to make the i2b2 platform accessible to the MU researchers. The i2b2 platform is a powerful tool for clinical researcher, but it can be difficult to install and uses. Our approach simplified the installation process by providing pre-built docker containers that can be easily deployed on a local machine or in the cloud. We also demonstrated a secure and scalable way to deploy i2b2 in the AWS ECS FARGATE serverless compute engine.

Overall, our contributions have significantly reduced the effort to installing and maintaining the i2b2 platform. By simplifying the installation process, we also made it easier for i2b2 developers to load data into i2b2 database. We provided a data installation package that uses ACT ontology to convert PCORnet CDM data into i2b2 fact and dimension tables. We also developed Snowflake JDBC driver support for i2b2 services. By providing snowflake driver support, we have made it possible for i2b2 services to query directly from Snowflake analytical database.

# BIBLIOGRAPHY


1. Shen JJ, Samson LF, Washington EL, Johnson P, Edwards C, Malone A. Barriers of HIPAA Regulation to Implementation of Health Services Research. *J Med Syst*. 2006;30(1):65-69. doi:10.1007/s10916-006-7406-z

2. Cimino JJ, Ayres EJ, Beri A, Freedman R, Oberholtzer E, Rath S. Developing a self-service query interface for re-using de-identified electronic health record data. *Stud Health Technol Inform*. 2013;192:632-636.

3. Murphy SN, Mendis M, Hackett K, et al. *Architecture of the Open-Source Clinical Research Chart from Informatics for Integrating Biology and the Bedside*. http://www.bisti.nih.gov/ncbc/

4. Wagholikar KB, Mendis M, Dessai P, et al. Automating Installation of the Integrating Biology and the Bedside (i2b2) Platform. *Biomed Inform Insights*. 2018;10:117822261877774. doi:10.1177/1178222618777749

5. Hung LH, Kristiyanto D, Lee SB, Yeung KY. GUIdock: Using Docker Containers with a Common Graphics User Interface to Address the Reproducibility of Research. *PLoS One*. 2016;11(4):e0152686. doi:10.1371/journal.pone.0152686

6. Amazon. Amazon Web Services. Accessed April 23, 2023. https://aws.amazon.com

7. Dageville B, Cruanes T, Zukowski M, et al. The Snowflake Elastic Data Warehouse. In: *Proceedings of the 2016 International Conference on Management of Data*. ACM; 2016:215-226. doi:10.1145/2882903.2903741

8. Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*. 1997;26(1):65-74. doi:10.1145/248603.248616

9. Hosny A, Vera-Licona P, Laubenbacher R, Favre T. AlgoRun: a Docker-based packaging system for platform-agnostic implemented algorithms. *Bioinformatics*. 2016;32(15):2396-2398. doi:10.1093/bioinformatics/btw120

10. i2b2. i2b2 Server-Side Architecture. Accessed April 23, 2023. https://community.i2b2.org/wiki/display/ServersideArchitectureHome/OVERVIEW

11. PCORI. Accessed April 23, 2023. https://www.pcori.org

12. The National Patient-Centered Clinical Research Network. PCORnet Network. Accessed April 23, 2023. https:/pcornet.org/network

13. PCORnet Common Data Model. Accessed April 23, 2023. https://pcornet.org/data/

14. ACT Network. Accessed April 23, 2023. https://dbmi-pitt.github.io/ACT-Network/ontology.html

# VITA

Md. Saber Hossain is a graduate research assistant in the Center for Biomedical Informatics at the University of Missouri, supervised by Dr. Abu Saleh Mohammad Mosa. He completed his Bachelor of Science in Computer Science and Engineering from Chittagong University of Engineering and Technology and pursuing his master's in Health Informatics. He has worked as a mobile application developer over five years in various startup and corporate companies. During this time, he led software engineers to develop and maintain more than 10 iOS applications. He is passionate about problem-solving with data structures and algorithms. He has a strong knowledge of web and mobile application development and experienced in using cloud services. As part of his graduate research assistantship work, his contribution simplified the installation and maintenance of the i2b2 platform. He demonstrated a way to deploy the i2b2 platform on AWS and Snowflake cloud data analytics platform in a secure and scalable way. His serverless deployment of i2b2 components reduced the effort required to install and maintain servers. He is interested in building his career as a software engineer.