Oracle® Health Sciences Omics Data Bank

Programmer's Guide

Release 1.0.1

E27509-02

April 2012

ORACLE®

Oracle Health Sciences Omics Data Bank Programmer's Guide, Release 1.0.1

E27509-02

Copyright © 2012 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface

This guide provides information on the Oracle Health Sciences Omics Data Bank (ODB) architecture.

Audience

This document is intended for users of Oracle Sciences Omics Data Bank. They could include Bioinformaticians, Database Administrators, Computational Biologists, Clinicians, Scientists, as well as Developers and Data Modelers.

Disclaimer Regarding Third Party Data

Oracle makes no express or implied warranty, including but not limited to warranties regarding the accuracy, completeness, merchantability, or fitness for a particular purpose, with respect to third party data loaded into this application or the results of any functions of the application using such data. It may be used for information purposes only, and no medical, clinical or other health related decisions may be based upon such results. You are solely responsible for your use of the third party data, including your right to use the data for your purposes.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Finding Information and Patches on My Oracle Support

Your source for the latest information about Oracle Health Sciences Cohort Explorer is Oracle Support's self-service Web site, My Oracle Support (formerly MetaLink).

Before you install and use an Oracle software release, always visit the My Oracle Support Web site for the latest information, including alerts, release notes, documentation, and patches.

Creating a My Oracle Support Account

You must register at My Oracle Support to obtain a user name and password account before you can enter the Web site.

To register for My Oracle Support:

1. Open a Web browser to `http://support.oracle.com`.
2. Click the **Register here** link to create a My Oracle Support account. The registration page opens.
3. Follow the instructions on the registration page.

Signing In to My Oracle Support

To sign in to My Oracle Support:

1. Open a Web browser to `http://support.oracle.com`.
2. Click **Sign In**.
3. Enter your user name and password.
4. Click **Go** to open the My Oracle Support home page.

Searching for Knowledge Articles by ID Number or Text String

The fastest way to search for product documentation, release notes, and white papers is by the article ID number.

To search by the article ID number:

1. Sign in to My Oracle Support at `http://support.oracle.com`.
2. Locate the Search box in the upper right corner of the My Oracle Support page.
3. Click the sources icon to the left of the search box, and then select Article ID from the list.
4. Enter the article ID number in the text box.
5. Click the magnifying glass icon to the right of the search box (or press the Enter key) to execute your search.

   The Knowledge page displays the results of your search. If the article is found, click the link to view the abstract, text, attachments, and related products.

In addition to searching by article ID, you can use the following My Oracle Support tools to browse and search the knowledge base:

- Product Focus — On the Knowledge page, you can drill into a product area through the Browse Knowledge menu on the left side of the page. In the Browse any Product, By Name field, type in part of the product name, and then select the product from the list. Alternatively, you can click the arrow icon to view the complete list of Oracle products and then select your product. This option lets you focus your browsing and searching on a specific product or set of products.
- Refine Search — Once you have results from a search, use the Refine Search options on the right side of the Knowledge page to narrow your search and make the results more relevant.
- Advanced Search — You can specify one or more search criteria, such as source, exact phrase, and related product, to find knowledge articles and documentation.

Finding Patches on My Oracle Support

Be sure to check My Oracle Support for the latest patches, if any, for your product. You can search for patches by patch ID or number, or by product or family.

To locate and download a patch:

1. Sign in to My Oracle Support at `http://support.oracle.com`.
2. Click the **Patches & Updates** tab.

   The Patches & Updates page opens and displays the Patch Search region. You have the following options:

   - In the Patch ID or Number is field, enter the primary bug number of the patch you want. This option is useful if you already know the patch number.
   - To find a patch by product name, release, and platform, click the Product or Family link to enter one or more search criteria.
3. Click **Search** to execute your query. The Patch Search Results page opens.
4. Click the patch ID number. The system displays details about the patch. In addition, you can view the Read Me file before downloading the patch.
5. Click **Download**. Follow the instructions on the screen to download, save, and install the patch files.

Finding Documentation on Oracle Technology Network

The Oracle Technology Network Web site contains links to all Oracle user and reference documentation. To find user documentation for Oracle products:

1. Go to the Oracle Technology Network at

   `http://www.oracle.com/technetwork/index.html` and log in.
2. Mouse over the Support tab, then click the **Documentation** hyperlink.

   Alternatively, go to Oracle Documentation page at

   `http://www.oracle.com/technology/documentation/index.html`
3. Navigate to the product you need and click the link.

   For example, scroll down to the Applications section and click Oracle Health Sciences Applications.
4. Click the link for the documentation you need.

Related Documents

For more information, see the following documents in the *Oracle Health Sciences Cohort Explorer* documentation set:

Oracle Health Sciences Cohort Explorer Documentation

The *Oracle Health Sciences Cohort Explorer Online Documentation Library* (Part E24441) documentation set includes:

- *Oracle® Health Sciences Cohort Explorer User's Guide Release 1.0* (Part E24437)
- *Oracle® Health Sciences Cohort Explorer Administrator's Guide Release 1.0* (Part E24438)
- *Oracle® Health Sciences Cohort Explorer Release Notes Release 1.0* (Part E24440)

- *Oracle® Health Sciences Cohort Explorer Secure Installation and Configuration Guide Release 1.0* (Part E24988)
- *Oracle® Health Sciences Cohort Explorer Implementation Scripts Guide Release 1.0* (Part E24989)
- *Oracle® Health Sciences Cohort Explorer Release Content Document Release 1.0* (Part E25021)

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Omics Data Model

This chapter contains the following topics:

1.1 Introduction

Oracle Health Sciences Omics Data Bank (ODB) consists of two groups of tables. One set of tables is a reference to provide the genome-features metadata required to link sample specimen results to specific portions of the genome. The second set of tables in the model is used to capture the sample specimen ('Omics') results and link each result to some object in the reference model and also link back to the patient. The patient link is accomplished by linking ODB with Cohort Data Model (part of Oracle Health Sciences Cohort Explorer 1.0).

This Omics data can be used by tools such as BI server that require a star schema to build dynamic SQL correctly. However, it is important to note that in this release the scope of the product is the data model only, without any front-end user interfaces. The additional components included with the model is a set of scripts that can be used to load reference genomic data from specific sources as well as several types of result data from a limited set of formats.

The model is intended to handle very large amounts of data (in the order of terabytes and more). To gauge the scale of the data, consider that the human genome is around 3 billion bases in each strand and the number of genes that produce proteins is around 23,000. Each gene has many different variants and attributes that will be described in detail. Since this holds true for each protein, a lot of supporting reference data is loaded for each gene/ protein/ pathway. The data cannot merely be reloaded to maintain an organized table as is the case with other tables that use ETL processes to load data. One approach is to use Index Organized tables since data can be added to the reference model as more reference data is discovered for each gene.

1.1.1 Reference Data

The reference data is loaded from 4 distinct sources:

1. The genomic information with corresponding gene data is loaded from EMBL files which are stored online in the Ensembl database (`http://www.ensembl.org`). Ensembl is a joint project of the European Bioinformatics Institute and the Wellcome Trust Sanger Institute. This online database maintains references to

other online database projects (dbSNP, NCBI, Cosmic, and so on) and provides references to each of these database. The model loads this cross reference information to, including known variation data, allow queries to use specific database references if needed.

2. The second source is the online SwissProt database (`http://www.ebi.ac.uk/uniprot/`) from which the model will obtain protein information. This database project is also a consortium of various groups including the European Bioinformatics Institute.
3. The third source is the HUGO Gene Nomenclature Committee (`http://www.genenames.org/`). This source provides reference seed information required for identifying human gene locations annotation only. The HUGO gene names are needed to find various cross references as well as the correct chromosome number for each gene. The HUGO Gene Nomenclature Committee is the authoritative group for all gene names.
4. The fourth reference source is Pathway Commons `http://www.pathwaycommons.org/pc/` which is used as the source for published pathways and proteins/genes participating in each pathway. The coverage of pathway as a reference is minimal.

In general, the above files use similar features to represent the data. The EMBL format is a flat file representation which provides an easy mechanism to parse and store data in a separate database structure. Both Ensembl and SwissProt have native schemas that can be downloaded. However, these schemas are 3NF structures that require a lot of work to coerce into a star schema model. The ODB model does not copy the source schema structures in any way. In addition, there are a lot of extra objects in the native schemas that are not necessary for the type of queries needed for the ODB requirements and these objects are omitted in ODB schema.

Most of the online databases let you download complete references, or specific references for sections of the genome. Since some customers may only need some genes or some proteins, and some may not need any protein information at all, the model will allow for any combination of specific data to be loaded as well as updated. Ensembl and SwissProt databases are maintained in an additive manner, so that new data is added on top of the existing data. This lets the ODB reference data to be expanded as required by the customer. The HUGO and Pathwaycommons databases are reloaded each time a new version of gene names or pathways is uploaded.

1.1.2 Result Data

In ODB, the data model handles two main types of genetic results: gene expression and sequencing. Gene Expression experiments capture information on how effectively certain genes respond to various conditions. On another hand, for sequencing results, while there are many different types of sequencing techniques, the net effect is to record all of the variants detected for each organism being tested, copy number variation and other related features.

The model is designed to facilitate gene expression results to be queried with sequencing results in the same SQL statement.

1.2 Logical Data Model

ODB contains two sets of tables:

1. Reference data tables
2. Result data tables

Each set of tables comes with a set of loading scripts to load data into these tables. The reference loaders write to the reference tables, while the result loaders write to result tables. However, there is one exception, the W_EHA_VARIANT table, where the result loaders enable you to report on any novel variants by writing to this table with any new variants found. A dedicated procedure invoked by the result loader reports on any novel variants.

Figure 1-1 shows how the reference tables link to create the ODB reference tables section. Only table names are shown in the figure.

Figure 1–1   Reference Data Logical Model (Table Names Only)

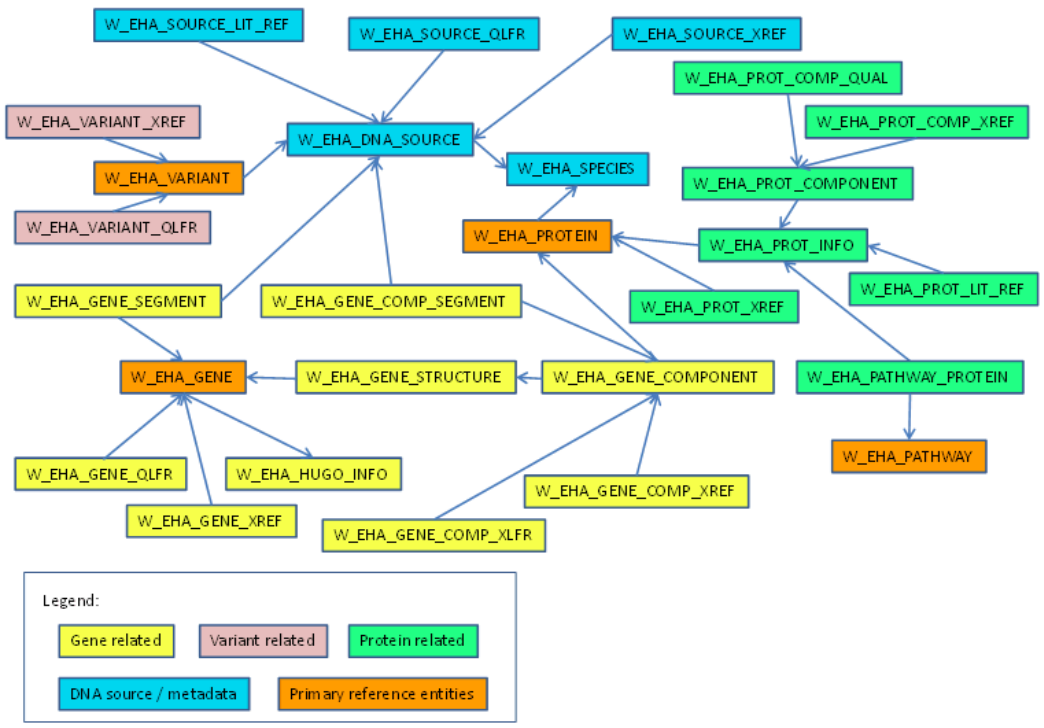
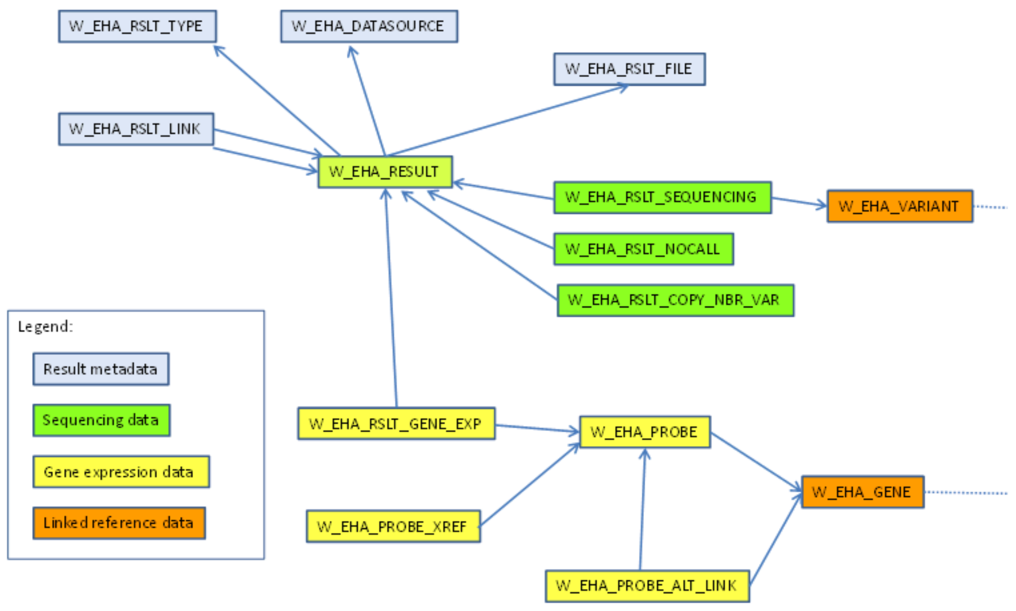


Figure 1-2 shows how the result tables link to create the ODB result tables section. Only table names are shown in the figure.

Figure 1–2 Result Data Logical Model (Table Names Only)



1.3 Reference Data Tables

The reference data starts with the SPECIES Dimension table and the DNA_SOURCE table.

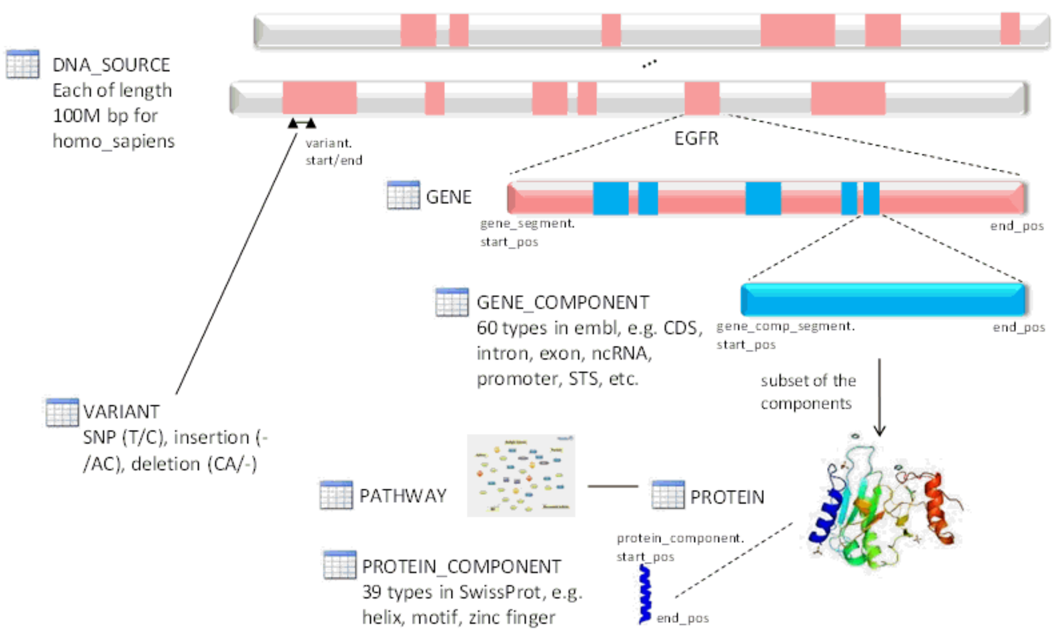
W_EHA_SPECIES:

This species table stores information about each genome in the database. The current model will allow for any number of species genomes to be loaded. This requires you to specify species in queries if there are similar genes between the organisms being tested.

W_EHA_DNA_SOURCE:

The DNA_SOURCE table will store multiple records for different reference DNA strands for each species. Each cell in the species will have a copy of this reference DNA. There are buffers of DNA considered to be the reference for each organism. These reference strands are then used to map detected variations for each organism tested. The W_EHA_DNA_SOURCE table has the foreign key top SPECIES and also has a CLOB field to store the reference strand information. The DNA has a specific character notation to keep track of each DNA base. There are additional characters used for sections that have not be sequenced ("N") and there are other characters used to represent other possible DNA bases. The records in this table are used as the parent records to map genes and gene components. If Ensembl releases patches that show how some genes are re-defined new DNA_SOURCE records will be created and then link the other records as needed. This table also stores the chromosome location which is described later.

Figure 1–3 Genome Division by the Data Model



W_EHA_GENE:

Each chromosome of the DNA strand has many different genes, which have a starting position and ending position. The entire size of the gene does not create the protein directly, but there are recognized sections of the DNA that scientists agree should be considered as part of the gene. The W_EHA_GENE table has fields for how the Ensembl database refers to the gene, as well as the recognized gene name. The recognized gene name is maintained by HUGO Gene Nomenclature Committee (http://www.genenames.org/). This reference information is loaded into the model to provide accurate chromosome information for each gene since the patch DNA sequences loaded do not list chromosomes.

W_EHA_GENE_SEGMENT:

This table is required to map the different segments of a gene to each buffer. Some genes are sequenced in multiple buffers and require this joining table to track each segment. This table gives the location in the buffer as well as a sequence number to keep track of the order of each segment that compose the gene. There is also a COMPLEMENT field that is used to indicate if a gene is transcribed in reverse order to create the protein.

_XREF, _QUALIFIER:

The XREF table associated with many of the different tables is used to list all of the cross reference information stored in the Ensembl database. There is a finite list of databases used, and each database has a specific format for the reference ID. You can use these reference ID values for queries. The _QUALIFIER tables associated with the different tables is used to list the other attributes. Each object can have an unlimited number of attributes such as "/note" that provides information to annotate the object. The database model will store this annotation data in case it is needed for reference.

W_EHA_GENE_STRUCTURE:
The process of gene transcription is accomplished by many different interim molecules that originate from sections of the gene. For each protein created, there is a distinct set of sections which are used. Each of these groups is identified in the EMBL file having the same TRANSCRIPT_ID qualifier. A GENE_STRUCTURE record is created to link to the protein and to be used as a parent record for all of the gene components. A given gene can have multiple proteins that are created (sometimes using the same sections) and each will have a different structure. Also, earlier research may have incorrect gene structures and the information is kept for historical reasons.

W_EHA_GENE_COMPONENT:
This table is used to store the various gene components. The EMBL file has many different objects listed (mRNA, CDS, STS, tRNA, misc RNA) and they all have a specific meaning. Views will be used to group the various types of objects in case there are queries to find genetic results that intersect with various gene regions. More user friendly names will be given (such as, mRNA = MESSENGER_RNA, CDS = CODING_REGION, STS = STRUCTURAL_SEGMENTS, and so on). You will be able to run various queries searching for mutations that occur in any of these regions, including the entire gene region.

W_EHA_GENE_COMP_SEGMENT:
This table is used to link each component to the DNA_SOURCE. Many of the gene components have joined sections. Sometimes the joined sections are detected in different source buffers as well and the foreign key to DNA_SOURCE is required for each part of the gene components. There is a sequence number to keep track of the order of each section used in the gene component.

W_EHA_PROTEIN:
Most of the known genes produce different types of protein molecules. The EMBL file lists the amino acids that comprise each protein molecule and uses an identifier for each protein molecule. The SwissProt files contains much more information about the protein molecule. There are more descriptive names for each protein which are not stored in the EMBL file (such as, insulin). The SwissProt file will also provide links to cross references as well as literature references. Each protein molecule can have many different components which are linked to this parent PROTEIN record.

W_EHA_PROT_COMPONENT:
This table is used to store all the protein components that are stored in the SwissProt files. You can import all or as many of the SwissProt files needed. This data may also be important for queries or reference. This can be important to show changes that may occur when variants are detected in the gene regions used to generate the amino acids of the protein.

W_EHA_VARIANT:
The VARIANT table is used to record the known reference sequence, REFERENCE_SEQ, corresponding to one or more variants that differ from the reference DNA_SOURCE. Most of these variants are well documented and compiled from other research. When results are uploaded, sometimes novel variants are detected and there are no known references for this variant. These results will generate new VARIANT records which may be of interest to researchers. There is a STATUS field which will be used to indicate NOVEL or KNOWN variants. Since this table will be queried frequently, it will be quite large and will require partitioning. The VARIANT table has a foreign key to the DNA_SOURCE record, not the GENE record. The reason being

that some genes may overlap, and there may also be several structures that are affected by a variant. This table will be used to create result foreign keys as described later.

W_EHA_HUGO_INFO:

This table is very important to store reference seed information needed for identifying gene locations. The EMBL files will report each gene with a LOCUS_TAG which uses the registered name with the HUGO Gene Nomenclature Committee (`http://www.genenames.org/`). The entire reference data from this group will be loaded as seed data in this table. This is important because of the way the EMBL files store patch sequences. The patch sequences (which are corrections to the human genome project) list the chromosome using the accession number of the DNA used for detection. The W_EHA_HUGO_INFO table is required to look up the HUGO gene names to find various cross references as well as the correct chromosome number for each gene.

W_EHA_PATHWAY:

Pathway is used to describe a series of interactions in a cell. Numerous biological pathways exist, including genetic, metabolic, signaling, and so on. This table is used to store publicly available pathways. Each pathway's participants are defined in PATHWAY_PROTEIN table which has a foreign key to PATHWAY table.

W_EHA_PATHWAY_PROTEIN:

This table is used to keep track which gene or proteins belong to a particular pathway. It has a foreign key to PATHWAY table which associates each gene or protein with one or more pathways.

1.4 Result Data Tables

The model currently supports the following two categories of results:

- Sequencing
- Gene Expression

Types of sequencing results include simple variants, copy number variation and no-call. These results are directed into four major tables:

- W_EHA_RSLT_SEQUENCING
- W_EHA_RSLT_GENE_EXP
- W_EHA_RSLT_COPY_NBR_VAR
- W_EHA_RSLT_NOCALL

W_EHA_RSLT_SEQUENCING:

This table contains sequencing results, more specifically variant information. It has a foreign key into the W_EHA_VARIANT table. The records in this table are linked to the record in the variant reference table. Information such as insertions, deletions, or substitutions would be recorded in this table along with any quality metrics on this information.

Note: A record in the W_EHA_RSLT_SEQUENCING table may come from any three result file types, namely VCF, MAF, or Complete Genomics (CGI) masterVar file.

W_EHA_RSLT_SEQUENCING_X:

This table is an additional table to store less frequently used sequencing metadata from the input file. It is always used as a helper table along with the main RSLT_ SEQUENCING table.

W_EHA_RSLT_NOCALL:

This table contains results coming from Complete Genomics sequencing files. Only CGI format records no-call results, that is instances when there is incomplete information to make a call regarding variant information on an allele.

W_EHA_RSLT_NOCALL_X

This table is an additional table to store less frequently used sequencing metadata from the input file. It is always used as a helper table along with the main RSLT_ NOCALL table.

W_EHA_RSLT_COPY_NBR_VAR:

This table contains copy number variation results coming from the Complete Genomics platform along with any relevant quality or count metrics. Currently no loading scripts are provided that support automatic parsing and loading of this data into the copy number variation result table. However, you can generate custom sequel to populate this table in order to query the results.

W_EHA_RSLT_CNV_X

This table is an additional table to store less frequently used sequencing metadata from the input file. It is always used as a helper table along with the main RSLT_ COPY_NBR_VAR table.

W_EHA_PROBE:

This table is intended to hold probe information for gene expression results, and each probe is designed to represent a particular gene. Since probe design varies by vendors, there may be multiple probes that correspond to same gene. In the rare instance where more than one gene matches a probe, the model has W_EHA_PROBE_ALT_LINK that would need to be manually populated. W_EHA_PROBE must be populated by the expression loader prior to loading any results corresponding to gene expression. In addition, any reference information pertaining to probes can be recorded in the W_ EHA_PROBE_XREF table.

W_EHA_RSLT_GENE_EXP:

This table is loaded from gene expression results and allows for storing gene intensity measurements, as well as quality metrics such as p-value and call information. A record can be inserted into this table only if the specified probe already exists in the W_EHA_PROBE table in order to establish a foreign key relationship.

W_EHA_RSLT_TYPE:

This table is a pre-seeded table with the types of results currently supported by the model. The result types are listed in Appendix A, "Additional Result Tables". Each record in the W_EHA_RSLT% tables supports a single specific result type.

W_EHA_STUDY

This table allows each result to be linked to a study if so specified by the end user during loading. The study table is intended to be a shadow copy of a table in the clinical data model, namely Cohort Explorer Data Model study table, thus it only holds study name and description. This table should be populated by the end user before loading any results pertaining to a given study. The presence of this table allows partitioning the results into groups based on a study the results have been collected for. This partitioning scheme leads to significant performance improvements.

W_EHA_CHROMOSOME

This table holds all the chromosomes names and is pre-seeded with names for all Human chromosomes. Refer to Appendix A, "Additional Result Tables" for pre-seeded data information. A result record in W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_NOCALL, W_EHA_RSLT_COPY_NBR_VAR may be linked to a particular chromosome. Similar to W_EHA_STUDY, this table is used to partition results for improved query performance.

W_EHA_RSLT_SPECIMEN

W_EHA_RSLT_SPECIMEN table is linked to all four result data tables. Every record in any of the result tables must have a foreign key that links to a particular specimen in this table. W_EHA_RSLT_SPECIMEN table in turn, links to W_EHA_DATASOURCE table which holds information about the database a given specimen comes from. This information, along with additional fields in SPECIMEN table such as SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER, can be sued to uniquely identify and pull more metadata about a given specimen from other source systems. This information is not stored in the ODB.

W_EHA_DATASOURCE:

This table stores information regarding specimen sources. Each genomic result must have a specimen record connected to it coming from another schema with patient results. Specimen identifier is the link between the clinical results and genomic results. This table needs to be populated by the user prior to running any result loaders. If ODB is intended to be used with Cohort Explorer data model, this table should be seeded with one specimen datasource, namely Cohort Data Model.

W_EHA_RSLT_FILE:

This table contains information about the input file you provide to populate the results. It stores information about the file storage type and the path to the file. The table is designed to store either external files (regular storage denoted by 'E'), or SecureFiles (secure storage denoted by 'S'). Currently, only loading of regular files is supported by the loaders, although you can manually link to any type of storage including Secure Files. Specific file descriptions as to their native formats and so on is stored in the W_EHA_RSLT_FILE_TYPE table and referenced via foreign keys.

W_EHA_RSLT_FILE_TYPE:

This table is pre-seeded with file types supported by the loaders into the model. The list of valid pre-seeded values is specified in the appendix. W_EHA_RSLT_FILE has a foreign key into this table.

2 Prerequisites for Loading Data

This chapter contains the following topics:

2.1 Setting up a Directory Object

Configure an Oracle directory object to be used for loading data files. Note that all of the loaders described here now use external tables. The ODB schema user must have the "CREATE ANY DIRECTORY" privilege as well as the "CREATE ANY TABLE" privilege. The name used for the directory object is used as a parameter to all of the loaders. The Oracle database OS account must have permissions to access the directory specified in the Oracle directory object. An example of this command is:

```
>create directory ODB_LOAD as '/home/oracle';
```

Note: The directory used must reflect the path requirements of the operating system that the database server is installed on. Windows database servers will have different naming convention than Linux servers. Also the directory used must be mounted on the host OS of the database server before the database server is started.

After a directory is created, the user creating the directory object needs to grant READ and WRITE privileges on the directory to other users as follows:

```
GRANT READ ON DIRECTORY ODB_LOAD_dir TO odb;
```

Extra Step on Exadata vs non-Exadata:

On Exadata, all the loaded result files are stored in staging tables because direct path loading is required to get the best compression. There is now a new script, `load_exadata_results.sql`, which is to be run by a DBA when the loaders are not loading result files. This script uses parallel DML for the session to enable direct path loading. For each staging table, the script locks the staging and the corresponding result table exclusively before moving the data. This script is only required on Exadata and must be executed by the ODB schema user. On Exadata, the loaders use a synonym which points to the staging tables and on non-Exadata the synonym points to the real result table.

2.2 Setting up Oracle Wallet

Oracle Wallet must be set up with the credentials used to connect to the schema where gdm is installed. Perform the following steps to set up Oracle Wallet:

1. Add the following code to tnsnames.ora under $ORACLE_HOME\NETWORK\ADMIN

```
DB001_Wallet =

  (DESCRIPTION =

    (ADDRESS_LIST =

      (ADDRESS = (PROTOCOL = TCP)(HOST = 10.178.187.186)(PORT
= 1521))

     )

   (CONNECT_DATA =

    (SERVICE_NAME = db001)

   )

  )
```

> **Note:** Set the SERVICE_NAME and HOST values above to point to your database installation.

2. Oracle wallet can be created on the client or the middle tier system. Open a command prompt terminal and execute the following:

```
>cd d:

>d:

>mkdir wallets

>cd wallets

>mkstore -wrl D:\wallets -create -nologo

Enter password: <type a 8 alphanumeric-character password>

Enter password again: <retype above password>

>dir

Volume in drive D is Data

Volume Serial Number is C###

Directory of D:\wallets

11/24/2011  09:24 PM    <DIR>          .

11/24/2011  09:24 PM    <DIR>          ..

11/24/2011  09:13 PM             3,965 cwallet.sso

11/24/2011  09:13 PM             3,888 ewallet.p12
```

Note: The last command should show two files created by running mkstore -create: cwallet.sso and ewallet.p12

3. Add your database credentials to your wallet.

   ```
   >mkstore -wrl D:\wallets -createCredential DB001_Wallet gdm

   Your secret/Password is missing in the command line

   Enter your secret/Password: <enter password for gdm user>

   Re-enter your secret/Password:<re-enter password>

   Enter wallet password:<enter the 8 digit password given while
   creating wallet>
   ```

   Note: For every user credential added to the wallet, you must create a new dataset name in tnsnames.ora. The system assumes username as gdm.

4. Configure SQLNET to look for wallet. Add the following lines of code to 'sqlnet.ora' under $ORACLE_HOME\NETWORK\ADMIN:

   ```
   WALLET_LOCATION = (SOURCE=(METHOD=FILE)(METHOD_
   DATA=(DIRECTORY=D:\wallets)))

   SQLNET.WALLET_OVERRIDE = TRUE
   ```

5. Test connectivity via sqlplus. On any command prompt terminal enter the following:

   ```
   >sqlplus /@DB001_Wallet
   ```

   You will get the following result:

   ```
   SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 25
   15:54:35 2011

   Copyright (c) 1982, 2010, Oracle.  All rights reserved.

   Connected to:

   Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 -
   64bit Production

   With the Partitioning, OLAP, Data Mining and Real Application
   Testing options
   ```

2.3 Integration with Oracle Health Sciences Cohort Explorer Data Model or Another External Data Model

Integration with other data models is done through specimen record. Each genomic data result file must be accompanied with SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER information, as well as SPECIMEN_DATASOURCE. These entities should match the record in specimen datasource schema. OHSCE Data Model is the default datasource for specimen in the current release. SPECIMEN_WID in W_EHA_RSLT_SPECIMEN table should match SPECIMEN_WID in the W_EHA_SPECIMEN table in OHSCE Data Model.

Every result record imported into the ODB schema is linked to a SPECIMEN record. The SPECIMEN data can come from any external schema either on the same instance (co-located) or on an external instance. The loaders which load various files call a specific stored procedure to validate that the SPECIMEN exists. The assumption is that any database used to provide the SPECIMEN information will have a single numeric field to identify the correct SPECIMEN. This numeric value is stored in each result record (without direct FK definition). The stored procedure used is in a package ODB_UTIL which is not wrapped to allow additional external databases to be supported. Currently this stored procedure is implemented to support validating SPECIMEN records stored in the Cohort Data Model. You need to expand this stored procedure to support other schemas that are intended to provide specimen data.

Following is the structure of ODB_UTIL.GET_SPECIMEN_WID stored procedure:

```
function get_specimen_wid (

    i_datasource_id in number,

    i_specimen_number in varchar2,

    i_specimen_vendor in varchar2

    ) return number;
```

This strored procedure has three parameters:

- DATASOURCE_ID: W_EHA_DATASOURCE table is used to configure each external database to provide SPECIMEN data. There is one default record for CDM, which has to be configured at the time of installation to specify the SCHEMA and optionally database link name. The stored procedure uses this information to use dynamic SQL to validate the specimen information.
- SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER: These two VARCHAR2 fields used to identify a unique specimen. s

The stored procedure looks up the W_EHA_DATASOURCE record and compares the name field. Currently, there is a check for a name of CDM and then code for looking specimen data in CDM. If additional database schemas need to be used to provide specimen information, you must first add a record first to W_EHA_DATASOURCE with a unique name. The stored procedure has to specifically handle that datasource name and add the code to validate the passed specimen number and specimen vendor number. Note that most of the data files support a specimen number, and the loaders currently have a specimen vendor number passed as a parameter.

2.3.1 Specimen and Vendor Number Requirement

To load results into any of the result tables, each specimen referred to in the input result files: VCF, MAF, CGI masterVar, gene expression must be present in the OHSCE data model. If a file with multiple specimen is being loaded, e.g. VCF file, and one of the specimen is not found in the Cohort Explorer datamart schema, then NONE of the records for the entire file will be loader. Note that this User needs to consider the compatibility of the available ENSEMBL version in ODB with other variation reference and results data before loading them to ODB.

2.4 Reference Version Compatibility

You must consider the compatibility of the available ENSEMBL version in ODB with other variation reference and results data before loading them to ODB.

Following are the list of data files to be considered for version compatibility:

1. GVF data file should belong to the same version of ENSEMBL that exists in ODB. For example, if ODB is loaded with ENSEMBL 66 version, then GVF file to be loaded should also belong to ENSEMBL 66.
2. Variation data files, which include CGI masterVar, VCF and MAF should be based on the same reference genome that was used by the ENSEMBL version. For example, if loaded ENSEMBL 66 version is using GRCh 19 reference genome, then the results to be loaded should also be mapped based on GRCh 19 version.
3. Copy Number Variation result data should also be checked for reference genome version compatibility with ENSEMBL version as specified in point 2 above.

2.5 Handling Newline Characters in Input Files

All reference and result input text files have an End-Of-Line character convention, that should be followed by the operating system on which the database server is loaded. For a windows database server, text files in a Linux or Unix environment must be processed by the tool unix2dos to convert file to the DOS format.

3 Loaders for Reference Data

This chapter contains the following topics:

3.1 Ensembl and SwissProt Loaders (Java)

3.1.1 Installing the Loader

Following are the prerequisites to installing the loader for reference data:

1. You must have an Oracle database instance with ODB installed in a schema, for which the name and password are known.

   Note: As the Reference part of the ODB model changes, the Reference Data Loader has to adapt to these changes. Therefore, both should be updated together.

2. Java Runtime 1.6 or higher must be installed and should be the default on the machine (can be verified using the "java -version" command from the command prompt).

Perform the following steps to install the loader:

1. Copy the Reference Loader folder into a directory of your choice. The program should be run from the directory it is installed in.

3.1.2 Files to Load

Following is a list of files to be loaded:

1. The Ensembl multi-gene EMBL files can be downloaded from:

   `ftp://ftp.ensembl.org/pub/release-66/embl/homo_sapiens`

The link above may not necessarily reflect the most recent version of multi-gene file available. Oracle recommends that you use the latest release available from Ensembl.

The whole Human genome (including various patches) is contained in four zipped .DAT files, and each one has to be loaded.

Each file has many contigs in it, each with a sequence of approximately 100,000 base pairs, multiple genes and other features pertaining to this sequence.

2. The SwissProt file (a single file) can be downloaded here:

   `http://www.ebi.ac.uk/uniprot/database/download.html`

3. Get the (UniProtKB/SwissProt) Flat File.

3.1.3 Loading the Data

Before you begin:

1. As the Loader runs, it will log some information in the GDM.LOG file. The file is always appended, so it will be growing. So, occassionally you may want to delete it and start from scratch the next time you run the Loader. Some of the information logged can be very useful for investigative purposes. Oracle recommends that you check the log when you have a problem.
2. The order of loading EMBL and SwissProt files is unimportant. The scenario outlined below is just an example. However, the GVF files (which are now loaded using a separate loader application) can only be loaded after all the Ensembl EMBL files have been loaded.

Perform the following steps to load files:

1. Since the SwissProt file is a single file, it can be loaded in under an hour (Human proteins only) or in a day (all species).To load the SwissProt file, run SwissProt.bat. When swissProt.bat is run you can optionally pass the Species List file. The purpose of the Species List file is to allow only loading protein information for the organism(s) you want. The format of the file is simple: type in the species primary (Latin) name, one species per line. A file for just the human genome is now included in the distribution - it is named Species.dat and contained in the main Loader Directory. If there is no -protFile option with the name of a species list file, ALL proteins for ALL species will be loaded (which will take much longer).

   If Oracle Wallet is set up swissProt.bat will use the credentials stored in the Wallet to connect to the schema else it will prompt for a password. If Oracle Wallet is set up the user will have to pass the following parameters to run swissProt.bat:

   a. Username — when oracle Wallet is set up enter ""
   b. Url — instance alias for which the Wallet credential was created - if you start SqlPlus as 'sqlplus /@DB001_Wallet, then this value here must be DB001Wallet.
   c. Schema name
   d. Path for Oracle Home
   e. Path to Wallet
   f. Path and name of species list. This is an optional parameter. If you do not have this list enter "

g. Complete path and name of the data file

Following is an example of how the swissProt.bat is to be run if Oracle Wallet is set up and Species list is not present:

```
C:\>swissProt.bat "" DB001Wallet trc_gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets ""
SwissProt.dat
```

Following is an example of how the swissProt.bat is to be run if Oracle Wallet is set up and Species list is present:

```
C:\>swissProt.bat "" DB001Wallet trc_gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets
Species.dat SwissProt.dat
```

If Oracle Wallet is not set up, you will have to pass the following parameters when swissProt.bat is run:

a. Username to connect to schema

b. Url — Full DB URL (host:port:instance). For example, Localhost:1613:devdb1

c. Schema name

d. Path for Oracle Home — when Oracle Wallet is not set up enter ""

e. Path to Wallet — when Oracle Wallet is not set up enter ""

f. Path and name of species list. This is an optional parameter. If you do not have this list enter ""

g. Complete path and name of the data file

Following is an example of how the swissProt.bat is to be run if Oracle Wallet is not set up and Species list is not present:

```
C:\>swissProt.bat trc_gdm localhost:1613:devdb1 trc_gdm "" ""
"" SwissProt.dat
```

This is an example of how the swissProt.bat is to be run if Oracle Wallet is not set up and Species list is present:

```
C:\>swissProt.bat trc_gdm localhost:1613:devdb1 trc_gdm "" ""
Species.dat SwissProt.dat
```

2. The Ensembl EMBL file(s) can be loaded next. Multiple EMBL files must be loaded one at a time, in any order, but without duplication (each file can only run once - otherwise, at present, some information is duplicated).

   If Oracle Wallet is set up, Embl.bat will use the credentials stored in the Wallet to connect to the schema else it will prompt you for a password. If Oracle Wallet is set up the user will have to pass the following parameters when Embl.bat is run:

   a. Username — when Oracle Wallet is set up, enter ""

   b. Url — instance alias for which the Wallet credential was created - if you start SqlPlus as `'sqlplus /@DB001_Wallet`, then this value here must be DB001Wallet.

   c. Schema name

   d. Path for Oracle Home

   e. Path to Wallet

   f. Complete path and name of the data file

Following is an example of how embl.bat is to be run if Oracle Wallet is set up:

```
C:\>embl.bat "" DB001Wallet trc_gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets
embl.dat
```

If Oracle Wallet is not set, you will have to pass the following parameters when Embl.bat is run:

a. Username to connect to schema
b. Url — Full DB URL (host:port:instance). For example, Localhost:1613:devdb1
c. Schema name
d. Path for Oracle Home — when Oracle Wallet is not set up enter ""
e. Path to Wallet — when Oracle Wallet is not set up enter ""
f. Complete path and name of the data file

This is an example of how the embl.bat is to be run if Oracle Wallet is not set up:

```
C:\>embl.bat trc_gdm localhost:1613:devdb1 trc_gdm "" ""
embl.dat
```

3.2 HUGO Loader (PLSQL)

The Hugo Loader is responsible for populating curated gene nomenclature records, taken from an online resource maintained by HUGO Gene Nomenclature Committee (HGNC), into ODB's reference database. The input data for the loader comprises the complete HGNC dataset which can be retrieved from their Statistics and Downloads webpage here: `http://www.genenames.org/cgi-bin/hgnc_stats.pl`. The data downloaded will be a large file with tabular text in tab-separated values given with column headers.

A batch file for Windows, and an alternative shell script for Linux-bash, has been provided for loading data. The content below shows the step-by step process of this load procedure.

Installing and Running the Loader

The loader can found bundled in the latest ODB build in the compressed file Hugo_Loader.zip. This folder consists of four files:

- loaddata_hugo.ctl
- hugo_loader.bat
- hugo_loader.sh

To run the loader, perform the following steps:

1. Copy the above files into a folder on your system along with the downloaded input file from HUGO.
2. Open a command prompt terminal and change the directory to where the above files reside.
3. The hugo_loader.bat file uses the credentials stored in Oracle Wallet to connect to the schema that has the ODB. The following parameters should be passed when the hugo_loader.bat is run:

   a. The name and complete path of the Hugo data file.
   b. The schema name

4. The format of the command for the batch file is as follows:

   ```
   C:\> hugo_loader.bat <HUGO file Name with full path ><Schema
   Name>
   ```

   Note:

   - The HUGO file name should not have any special characters such as '=' in its file name.

5. The hugo_loader.sh can be run with or without Oracle Wallet. The following parameters need to be passed when hugo_loader.sh is run:

   a. The name and complete path of the Hugo data file

   b. The schema name

   c. 1 if Oracle Wallet is set up else 0

   d. If the previous parameter is 0 enter the username to connect to the schema

   Examples:

   Following is an example of how hugo_loader.sh should be run when Oracle Wallet is set up:

   ```
   sh hugo_loader.sh  <HUGO file Name with full path ><Schema
   Name>1
   ```

   Following is an example of how hugo_loader.sh should be run when Oracle Wallet is not set up:

   ```
   sh hugo_loader.sh  <HUGO file Name with full path ><Schema
   Name>0<username>
   ```

6. Once the loading is complete, log into SQL developer, or SQP*Plus with ODB Schema and verify that 35000 or more records are populated in W_EHA_HUGO_INFO table. Also check the log file (hugo.log) that is created in the same folder from which the .bat or .sh script was run, which will indicate the number of records that have been read, number of records that have been loaded into the staging table and the number of records that have been discarded.

3.3 GVF Ensembl Loader (PLSQL)

The GVF Ensembl loader is responsible for the input of known variants of any given species for which DNA source records are present. The loader will load only those variant records from the input file, for which matching DNA source records exist in the DB. (that is, those variants that fall into the absolute position ranges of a DNA source record with the same chromosome and species ID). Hence ensure the EMBL loader is run first with the relevant species' EMBL input files.

Since GVF files do not contain information about the species, it is necessary to pass a species_ID value as parameter to run the loader. This requires W_EHA_Species table to have the relevant species record with a primary key ID which is then passed as said parameter.

Any GVF file can be loaded multiple times. For Homo sapiens, the input file can be downloaded from the ensemble FTP link:

ftp://ftp.ensembl.org/pub/release-65/variation/gvf/homo_sapiens/

The link above may not necessarily reflect the most recent version of GVF file available. Oracle recommends that you use the latest GVF file.

For the GVF loaders, the .bat and .sh files have the same parameters except that the BAT script only supports using the Oracle Wallet connection (like all other loaders). The list of parameters are as follow (in this expected order):

1. GVF file to be loaded. Should include full path to allow sqlldr to access correctly.
2. SPECIES_WID to match the primary key ID on the SPECIES table matching the data. This will usually be 1 since only Homo sapiens will be installed.
3. TNS name to connect to the database.
4. 1 to use wallet connection to database, 0 to prompt for password (used only in SH script).
5. Username to connect to the database (used only in SH script).

Following is the command line input for windows:

```
gvf_loader.bat <input_file(.gvf)> <species_wid> <wallet_name> 1
```

Following is the command line input for a shell scipt without the use of a wallet instance:

```
sh gvf_loader.sh <input_file(.gvf)> <species_wid> <dbname> 0
<user>
```

Following is the command line input for a shell scipt with a wallet instance:

```
sh gvf_loader.sh <input_file(.gvf)> <species_wid> <wallet_name>
1
```

Examples:

```
gvf_loader.sh Homo_sapiens.gvf 1 DB001 0 gdm
```

```
>gvf_loader.bat Homo_sapiens.gvf 1 DB001Wallet 1
```

3.4 Pathway Loader

3.4.1 Description

Pathway_loader is the utility for extracting, transforming and loading GSEA standard file formats.

The data used in our design case is can be downloaded from `http://www.pathwaycommons.org/pc-snapshot/current-release/gsea/by_species/homo-sapiens-9606-gene-symbol.gmt.zip`.

The pathway_loader utility is compatible with Oracle RDBMS 10.2 and above. It is not operating system dependent and works entirely within Oracle database. This section describes the setup procedure and also shows how to use the utility to load data from GSEA file located on your system.

3.4.2 Installing and Running the Loader

The loader is made up of three files:

1. pathway_loader.bat

2. pathway_loader.sh
3. gseaload.ctl

The pathway_loader.bat and pathway_loader.sh script first run sql loader that loads the data into the staging table from the data file. The scripts then call stored procedure to load the data from the staging table to the target table.

The pathway_loader.bat file requires the logon credentials to be stored in Oracle Wallet. The following parameters need to be passed when the bat file is run:

1. The name and complete path of the data file
2. The schema name

Following is an example of how the pathway_loader.bat file should be run:

```
C:\> pathway_loader.bat <Data file Name with full path ><Schema
Name>
```

The pathway_loader.sh script can be run with or without Oracle Wallet being set up. The following parameters need to be passed when pathway_loader.sh is run:

1. The name and complete path of the Hugo data file
2. The schema name
3. The schema name
4. If the previous parameter is 0 enter the username to connect to the schema

Following is an example of how pathway_loader.sh should be run when Oracle Wallet is set up:

```
Sh pathway_loader.sh  <Data file Name with full path ><Schema
Name>1
```

Following is an example of how hugo_loader.sh should be run when Oracle Wallet is not set up:

```
Sh pathway_loader.sh  <Data file Name with full path ><Schema
Name>0<username>
```

Also check the log file that is created in the same folder from which the bat or sh script was run, which will indicate the number of records that have been read, number of records that have been loaded into the staging table and the number of records that have been discarded.

The procedure to load the data from the staging table to the target will create a function call process_string() and a procedure called load_pathway(). The process_string function shred through a single column tab, space, semicolon, pipe or similar delimiters data and tokenize them. The procedure load_pathway called this function and loop through the tokenized list and insert them into the destination tables accordingly.

The program in the utility is a PL/SQL program utilizing BULK COLLECT method. The program will load 8800 records in 6.77 seconds. The main concept used in the SELECT statement bind the result set of the query to a collection providing much less communication between the PL/SQL and SQL engines. All variables in the INTO clause are collection.

3.4.3 Files and Tables Used in Loading

The following file and tables are used in the loading process.

- Source file: homo-sapiens-9606-gene-symbol.gmt. You can view this file by downloading it from

  `http://www.pathwaycommons.org/pc-snapshot/current-release/gsea/by_species/homo-sapiens-9606-gene-symbol.gmt.zip.`

  The first column and the second column in this file are normal tab delimited but the third column in the file is a string containing delimited values.

- Staging table:

  Table name: W_EHA_PATHWAY_GSEA_STG

  ```
  PATHWAY_NAME                VARCHAR2(100 CHAR)
  PATHWAY_SOURCE_ID           VARCHAR2(15 CHAR)
  PATHWAY_PROTEIN_SYMBOL      CLOB
  );
  ```

- Master destination table:

  Table name: W_EHA_PATHWAY

  ```
  ROW_WID                     NUMBER(38,0)
  PATHWAY_SOURCE_ID           VARCHAR2(50 BYTE)
  PATHWAY_NAME                VARCHAR2(200 BYTE)
  W_INSERT_DT                 DATE
  W_UPDATE_DT                 DATE
  ETL_PROC_WID                NUMBER(10,0)
  ENTERPRISE_ID               NUMBER(38,0)
  );
  ```

- Child destination table:

  Table name: W_EHA_PATHWAY_PROTEIN

  ```
  ROW_WID                     NUMBER(38,0)
  PATHWAY_WID                 NUMBER(38,0)
  HUGO_SYMBOL                 VARCHAR2(50 BYTE)
  W_INSERT_DT                 DATE
  W_UPDATE_DT                 DATE
  ETL_PROC_WID                NUMBER(10,0)
  ENTERPRISE_ID               NUMBER(38,0)
  );
  ```

  For relationship information, refer to the ODB data model.

3.5 Template of Command Line Arguments for Reference Loaders

This section provides a summary of templates for running each data loader, along with an example using sample summary input files.

3.5.1 Glossary of Parameter Templates

The following is a glossary of common data entries and a brief description of their point of origin.

Table 3–1 Glossary of Parameter Templates

Parameter Name	Description
user	DB user name
host_url:dbname	DB connection string (include system-add:port:dbname). to be used if no Wallet is provided. I think this DB connection string is required only for Java loaders. For all other loaders if u enter the schema name its enough.
schema_name	DB instance name
species_file	File with a list of Species names
input_file(.dat)	Input File, with path if not present in same folder (File type)
wallet_name	Name of Wallet created
sqlnet.ora_path	Path to directory with file sqlnet.ora and tnsnames.ora
wallet_dir	Directory with wallet files
species_wid	Primary Key ID from W_EHA_SPECIES
file_type_Code	File_type_code from W_EHA_RSLT_FILE_TYPE
file_version	File_type_version from W_EHA_RSLT_FILE_TYPE
vendor_name	User input of Vendor name for Result files
data_source_name	datasource_name from W_EHA_DATASOURCE
species_name	species_name from W_EHA_SPECIES
specimen_number	If datasource is CDM, give the SPECIMEN_NUMBER under W_EHA_SPECIMEN_PATIENT_H. (Has to be present in CDM table, but should only be given for gene_expression_loader.)
specimen_vendor_number	If datasource is CDM, give the SPECIMEN_VENDOR_NUMBER under W_EHA_SPECIMEN_PATIENT_H. (Has to be present in CDM table.)

3.5.2 SwissProt

There are four scenarios for command line input:

Without Oracle Wallet and with species:

Windows:

```
>swissProt.bat <user> <host_url:dbname> <schema_name> "" ""
<species_file> <input_file(.dat)>
```

Linux:

```
>sh SwissProt.sh <user> <host_url:dbname> <schema_name> "" ""
<species_file> <input_file(.dat)>
```

For example,

```
>swissProt.bat gdm localhost:1521:db001 gdm "" "" Species.dat
uniprot_sprot.dat
```

Without Oracle Wallet and without species:

Windows:

```
>swissProt.bat <user> <host_url:dbname> <schema_name> "" "" ""
<input_file(.dat)>
```

Linux:

```
>sh SwissProt.sh <user> <host_url:dbname> <schema_name> "" "" ""
<input_file(.dat)>
```

Example:

```
>swissProt.bat gdm localhost:1521:db001 gdm "" "" "" uniprot_
sprot.dat
```

With Oracle Wallet and with species:

Windows:

```
>swissProt.bat "" <wallet_name> <schema_name> <sqlnet.ora_path>
<wallet_dir> <species_file> <input_file(.dat)>
```

Linux:

```
>sh SwissProt.sh "" <wallet_name> <schema_name> <sqlnet.ora_
path> <wallet_dir> <species_file> <input_file(.dat)>
```

Example:

```
>swissProt.bat "" DB001Wallet gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets
Species.dat uniprot_sprot.dat
```

With Oracle Wallet and without species:

Windows:

```
>swissProt.bat "" <wallet_name> <schema_name> <sqlnet.ora_path>
<wallet_dir> "" <input_file(.dat)>
```

Linux:

```
>sh SwissProt.sh "" <wallet_name> <schema_name> <sqlnet.ora_
path> <wallet_dir> "" <input_file(.dat)>
```

Example:

```
>swissProt.bat "" DB001Wallet gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets ""
uniprot_sprot.dat
```

3.5.3 EMBL

The input is similar to SwissProt without species as input.

Without Oracle Wallet:

Windows:

```
>embl.bat <user> <host_url:dbname> <schema_name> "" "" <input_
file(.dat)>
```

Linux:

```
>sh embl.sh <user> <host_url:dbname> <schema_name> "" "" <input_
file(.dat)>
```

Example:

```
>embl.bat gdm localhost:1521:db001 gdm "" "" Homo_
sapiens.12000.dat
```

With Oracle Wallet:

Windows:

```
>embl.bat "" <wallet_name> <schema_name> <sqlnet.ora_path>
<wallet_dir> <input_file>
```

Windows:

```
>embl.bat "" <wallet_name> <schema_name> <sqlnet.ora_path>
<wallet_dir> <input_file(.dat)>
```

Linux:

```
>sh embl.sh "" <wallet_name> <schema_name> <sqlnet.ora_path>
<wallet_dir> <input_file(.dat)>
```

Example:

```
>embl.bat "" DB001Wallet gdm C:\ora11g\product\11.2.0\dbhome_
2\NETWORK\ADMIN D:\wallets Homo_sapiens.12000.dat
```

3.5.4 HUGO

The Hugo batch file runs only with an Oracle Wallet instance. The Shell scripts can be executed in two modes:

Without Oracle Wallet:

Linux:

```
>sh hugo_loader.sh <input_file(text)> <schema_name> 0 <user>
```

Example:

```
>hugo_loader.sh hgnc_downloads.cgi DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>hugo_loader.bat <input_file(text)> <wallet_name>
```

Linux:

```
>sh hugo_loader.sh <input_file(text)> <wallet_name> 1
```

Example:

```
>hugo_loader.bat hgnc_downloads.cgi DB001Wallet
```

3.5.5 GVF

Like Hugo, the batch file will only run with a Wallet instance.

Without Oracle Wallet:
Linux:

```
>sh gvf_loader.sh <input_file(.gvf)> <species_wid> <schema_name>
0 <user>
```

Example:

```
>gvf_loader.sh Homo_sapiens.gvf 1 DB001 0 gdm
```

With Oracle Wallet:
Windows:

```
>gvf_loader.bat <input_file(.gvf)> <species_wid> <wallet_name>
```

Linux:

```
>sh gvf_loader.sh <input_file(.gvf)> <species_wid> <wallet_name>
1
```

Example:

```
>gvf_loader.bat Homo_sapiens.gvf 1 DB001Wallet
```

3.5.6 Pathway

The files run similar to HUGO.

Without Oracle Wallet:
Linux:

```
>sh pathway_loader.sh <input_file(.gmt)> <schema_name> 0 <user>
```

Example:

```
>pathway_loader.sh homo-sapiens-9606-gene-symbol.gmt DB001 0 gdm
```

With Oracle Wallet:
Windows:

```
>pathway_loader.bat <input_file(.gmt)> <wallet_name>
```

Linux:

```
>sh pathway_loader.sh <input_file(.gmt)> <wallet_name> 1
```

Example:

```
>pathway_loader.bat homo-sapiens-9606-gene-symbol.gmt
DB001Wallet
```

4 Loaders for Result Data

This chapter includes the following topics:

4.1 Prerequisites

Before loading the result loaders ensure that the reference Ensembl files have been loaded using the java loader.

Note: The reference loaded has to match the reference used for alignment of all other files to be loaded as well.

After the reference is loaded, perform the following steps to initialize your database:

1. Create W_EHA_RSLT_STUDY records. There is a sequence (W_EHA_RSLT_STUDY_S) associated with this table to allow for as many study records as needed for testing. The result data is now partitioned using the FK to study. So the required number of studies should added to this table. The result loaders use the value for RESULT_STUDY_NAME as the value to lookup the corresponding study primary key. For simple testing you only need one record.
2. Configure a W_EHA_DATASOURCE record to identify the CDM schema to be used to validate specimen numbers. Each result record that is to be loaded, must have the specimen exist in the CDM schema in the W_EHA_SPECIMEN_PATIENT_H table. Note that the CDM schema needs the v1.01 patch installed, which adds a SPECIMEN_VENDOR_NUMBER field to be used for vendor specific information. The W_EHA_DATASOURCE can use a database link if the CDM schema is in another instance.

3. Ensure that the ODB schema has SELECT privileges on the W_EHA_SPECIMEN_PATIENT_H table in the CDM schema.
4. All of the specimens required for the example files should be added into the W_EHA_SPECIMEN_PATIENT_H table in the CDM schema.

To install the loader, Copy the **Results_Loader** folder into a directory of your choice. The program should be run from the directory it is installed in.

4.2 Overview of Result Loaders

There are four result loaders that are developed, one for each file type:

1. Gene Expression
2. CGI
3. MAF
4. VCF

These four loaders can be run using the .bat file in Windows or the shell scripts in Linux.

The .bat files to be run in Windows are as follows:

- gene_expression_loader.bat
- CGI_loader.bat
- MAF_loader.bat
- VCF_loader.bat

Each of the four result loaders will require the following parameters to be passed in the order:

- Name of the Result file - This can include a sub directory created on the directory used. Any sub-directory has to match the syntax of the operating system on the database server. The database server must have full access privileges on this directory.
- The Oracle Directory object - For more information refer to Chapter 2, "Prerequisites for Loading Data". Oracle recommends that you always use capitalized names for Oracle directory objects.
- Species Type - The actual species name should be passed. Usually this will be "Homo sapiens" if reference is loaded for humans. The name can be found in the W_EHA_SPECIES table.
- Study name - This should be a value in the W_EHA_RSLT_STUDY.RESULT_STUDY_NAME column.
- Data Source Name - This is a value inW_EHA_DATASOURCE.DATASOURCE_NM.
- Specimen Number - should be given for gene expression loader only. If CDM is referenced, this value should be present for a record in W_EHA_SPECIMEN_PATIENT_H table under SPECIMEN_NUMBER.
- Specimen Vendor Number - If CDM is referenced, this value should be present for a record in W_EHA_SPECIMEN_PATIENT_H table under SPECIMEN_VENDOR_NUMBER.
- Wallet Name

Note: Oracle Wallet must be set up before the batch files can be run successfully.

The shell scripts to be run in Linux are as follows:

- gene_expression_loader.sh
- CGI_loader.sh
- MAF_loader.sh
- VCF_loader.sh

Each of the four result loaders will require the following parameters to be passed in the order:

- Name of the Result file - This can include a sub directory created on the directory used. Any sub-directory has to match the syntax of the OS on the database server. The database server must have full access privileges on this directory.
- The Oracle Directory object - For more information refer to Chapter 2, "Prerequisites for Loading Data". Oracle recommends that you always use capitalized names for Oracle directory objects.
- Species Type - The actual species name should be passed. Usually this will be "Homo sapiens" if reference is loaded for humans. The name can be found in the W_EHA_SPECIES table.
- Name of the Study
- Data Source Name
- Specimen Number - Should be given for gene_expression_loader ONLY. If CDM is referenced, this empty value should be present for a record in W_EHA_SPECIMEN_PATIENT_H table under SPECIMEN_NUMBER.
- Specimen Vendor Number
- Schema/Wallet name - If Oracle Wallet is set up, enter Wallet name else enter the schema name.
- 1 if Oracle Wallet is set up else 0
- If previous parameter was 0 then enter schema username

If Oracle Wallet is set up the shell script uses those credentials to run the Sqlldr and Sqlplus. If Oracle Wallet is not set up the script prompts for a password and connects to Sqlldr and Sqlplus.

4.3 Probe Loader

The probe loader is used to populate the W_EHA_PROBE table. You can use probe_loader.bat to run in Windows or probe_loader.sh to run in Linux. Probe loader is somewhat of a reference loader rather than result but it does vary with vendors, for example, Affymetrix, Illumina.

Running Probe Loader on Windows

To run probe_loader.bat you need to pass the following parameters in order:

1. Name and path of the probe data file
2. Wallet name

The probe loader then calls the control file to populate the data into the staging table and then calls the procedure that loads data into W_EHA_PROBE. The bat file requires Oracle Wallet to be set up before it can run successfully.

The bat file can be run as follows:

```
C:\>probe_loader.bat <Name and path of probe data file><Wallet
name>
```

Running Probe Loader on Linux

To run probe_loader.sh you must pass the following parameters in order:

1. Name and path of the probe data file
2. Schema/Wallet name - if Oracle Wallet is set up enter the wallet name else enter the schema name
3. If Oracle Wallet is set up, then 1 else 0
4. If the previous parameter is 0 then enter the username of the schema

The shell script calls the control file to populate the data into the staging table and then calls the procedure that loads data into W_EHA_PROBE.

If Oracle Wallet is set up the shell script uses those credentials to run the Sqlldr and Sqlplus. If Oracle Wallet is not set up the script prompts for a password and connects to Sqlldr and Sqlplus.

If Oracle Wallet is set up, then the shell script can be run as follows :

```
sh probe_loader.sh <Name and path of the probe data file><Wallet
name>1
```

If Oracle Wallet is not set up, the shell script can be run as follows :

```
sh probe_loader.sh <Name and path of the probe data file><Schema
Name>0<Schema username>
```

Once the installer scripts are run, check the log file (probe_loader.log) that is created in the same folder from which the bat or sh script was run. This log file will indicate the number of records that have been read, number of records that have been loaded into the staging table and the number of records that have been discarded.

4.3.1 Assumptions for Data File

Following are the assumptions for the data file for the Gene Expression Loader:

1. The file is tab separated.
2. The first row is always the header.

Mappings for Probe Loader

Table 4–1 Mappings for Probe Loader

Data File	W_EHA_PROBE table
PROBESET	W_EHA_PROBE.PROBE_NAME
ACC	W_EHA_PROBE.ACCESSION
DESCP	W_EHA_PROBE.PROBE_DESC
GENEID	W_EHA_PROBE.PRIMARY_HUGO_ NAME

4.4 Gene Expression Loader

The gene expression loader loads the W_EHA_RSLT_FILE, and W_EHA_RSLT_GENE_EXP tables. These tables can be loaded by running the gene_expression_loader.bat file in Windows or the gene_expression_loader.sh file in Linux.

Note: The gene expression loader assumes that probe loader (see Section 4.3, "Probe Loader") has already populated W_EHA_PROBE table with probe names corresponding to genes.

4.4.1 Running Gene Expression Loader on Windows

The batch file requires the following parameters to be passed in the order. If any of the parameters have spaces in them enclose them in "":

- File name of the gene expression data file
- Oracle Directory Object
- Study name
- Datasource Name
- Specimen Number
- Specimen Vendor Number
- Wallet Name

The w_eha_rslt_err_log table will contain error records if the records were not successfully loaded into the target tables.

Note: Oracle Wallet must be set up before the batch files can be run successfully.

You can run the gene_expression_bat as follows:

```
C:\>gene_expression_loader.bat <File name of the gene expression
data file><SourceOracle Directory Object><Study Source
Name><Species Type><Specimen Number><Specimen Vendor
Number><Walleta Name>
```

4.4.2 Running Gene Expression Loader on Linux

The gene_loader.sh will require the following parameters to be passed in the order. If any of the parameters have spaces in them enclose them in "":

- File name along with the gene expression data file
- Oracle Directory Object
- Study Name
- Datasource Name
- Specimen Number
- Specimen Vendor Number
- Schema/Wallet Name - If Oracle Wallet is set up enter the Wallet name else enter the schema name.

- 1 if Oracle Wallet is set up else 0
- If previous parameter was 0 then enter schema username

The w_eha_rslt_err_log table will contain error records if the records were not successfully loaded into the target tables.

If Oracle Wallet is set up the shell script uses those credentials to run the Sqlldr and Sqlplus. If Oracle Wallet is not set up the script prompts for a password and connects to Sqlldr and Sqlplus.

If the Oracle wallet is set up, you can run the gene_expression_loader.sh script as follows:

```
gene_expression_loader.sh <File nameof the gene expression data
file><Oracle Directory Object><Study Name><Data Source
Name><Specimen Number><Specimen Vendor Number><Wallet Name>1
```

If Oracle Wallet is not set up, you can run the gene_expression_loader.sh script as follows:

```
gene_expression_loader.sh <File name  of the gene expression
data file><Oracle Directory Object><Study Name><Data Source
Name> <Specimen Number><Specimen Vendor Number><Schema
Name>0<Schema Username>
```

4.4.3 Assumptions for Data File

Following are the assumptions for the data file for the Gene Expression Loader:

1. The file is tab separated.
2. The first row is always the header.
3. The first column is named DATA.
4. Each hybridization present in the data file should have three columns in the following order:
   a. Intensity - Header value should be the Hybridization Name
   b. Call
   c. P-Value
5. The first column for each hybridization should contain only the hybridization name. The values in this column with be the hybridization intensity value.
6. The total size of the header in the data file should not be greater than 32000 characters.

Mappings for Gene Expression Loader

Table 4–2 Mappings for Gene expression Loader

Data File	W_EHA_RSLT_GENE_EXP
DATA	There will be a look up in W_EHA_PROBE, corresponding ROW_WID will be populated in W_EHA_RSLT_GENE_EXP.PROBE_WID
HYBRIDIZATION - Header	W_EHA_RSLT_GENE_EXP.HYBRIDIZATION_NAME
HYBRIDIZATION - Data Values	W_EHA_RSLT_GENE_EXP.INTENSITY

Table 4–2 (Cont.) Mappings for Gene expression Loader

Data File	W_EHA_RSLT_GENE_EXP
HYBRIDIZATION_Call	W_EHA_RSLT_GENE_EXP.CALL
HYBRIDIZATION_P-VALUE	W_EHA_RSLT_GENE_EXP.P_VALUE

4.5 CGI Sequence Data Loader

The CGI 2.0 file format is described here:

`ftp://ftp2.completegenomics.com/`

Each section in a CGI file is self-contained and separate. There are three types of sections:

- Comment lines - beginning with #
- Header - beginning with >
- Actual result data with information about the Zygocity, Variant Type, Reference, Alleles, Scores and Count.

The main challenge for loading a CGI file is to parse the #SAMPLE information from the comments section and then map it with the rest of the data. This sample information is important to retrieve the Specimen_Id from the data source you have mentioned while executing the batch file.

4.5.1 Files to Load

The execution call of the stored procedure `odb_result_util.process_cgi_stage()` is designed in the script file load_cgi.sql. This stored procedure will accept FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, DATA SOURCE and SPECIMEN VENDOR as an input parameter.

Note: This stored procedure will create an external table dynamically and upload data from the source file to it. External tables allow Oracle to query data that is stored outside the database in flat files. The ORACLE_LOADER driver can be used to access any data stored in any format that can be loaded by the SQL*Loader. No DML can be performed on the external tables but they can be used for query, join and sort operations.

Two external tables will be created dynamically. The first table will store the specimen number which will be parsed from the comments section of the file. It will also store the #Sample tag and the related specimen number. The second external table will store the complete result data. `Odb_util.get_specimen_wid()` function will retrieve the specimen_id for the corresponding specimen number and insert a record into w_EHA_RSLT_SPECIMEN table.

Two multi-table insert statements will be written dynamically. One will insert records into W_EHA_VARIANT_STG, W_EHA_RSLT_NOCALL and W_EHA_RSLT_NOCALL_X tables and the other will insert records into W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SEQYUENCING_X table.After inserting the record into W_EHA_VARIANT_STG table, a `PROCESS_VARIANT()` procedure will be called which will populate the W_EHA_VARIANT table.

4.5.2 Data Load

Any allele that is marked as "ref" indicates that the section of the genome matches the reference sequence. This data will not be saved to the database. Any allele not matching the database will generate two records in the ODB. Specimen identifiers are stored in the CGI data file header. There are several values used to get the correct Primary Key value of the corresponding SPECIMEN record which is external to the ODB schema. The SPECIMEN identifier is the first field used to look up the external database. This will most likely be a barcode or some other natural key. This value is not necessarily unique in the other database (especially coming from HDM) where different vendors can have different barcode systems that may overlap. So there is also a specimen vendor number used to look up the correct SPECIMEN record. The last value needed is used to specify if multiple sources are used to provide SPECIMEN records. Each result table will store a FK to the correct datasource for the SPECIMEN as well as the Primary Key value for the SPECIMEN record. The loading code will must use all three fields to find the correct values for the result records.

The alleles identified as "no-call" will create a W_EHA_RSLT_NOCALL and W_EHA_RSLT_NOCALL_X record. All other non-ref alleles will create a W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SEQUENCING_X record.

The other columns are mapped as shown below. If both alleles are not homogeneous and are both non-references, then a total of four records will be created for such a row in the master variant.

Note: The batch file requires Oracle Wallet to be set up to run correctly.

The batch file requires the following parameters to be passed in order. If any of the parameters have spaces in them enclose them in "":

- File name of the CGI data file
- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Wallet Name

Following command will be used to execute the batch file to upload the CGI data.

```
d:\> CGI_loader.bat <CGI file Name with full path ><Oracle
Directory Object>< Species Type ><Study Name><Data Source
Name><Species name><Specimen Vendor Number><Wallet Name>
```

For example,

```
d:\>cgi_loader.bat "masterVarBeta-NA19240-L2-200-37-ASM
small.tsv"ODB_LOAD" "'Homo sapiens'" STUDY1" "CDM"
"vendor2","db004_w"
```

The shell script requires the following parameters to be passed in order. Enclose spaces in "":

- File name of the CGI data file

- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Schema/Wallet Name- If Oracle Wallet is set up enter the Wallet name else enter the schema name 1 if Oracle Wallet is set up else 0
- If previous parameter was 0 then enter schema username

The shell script can be run with or without Oracle Wallet being set up. The following command will be used to execute the shell script to upload CGI Data when Oracle Wallet is set up.

```
sh CGI_loader.sh <CGI file Name >< Directory Object >< Species
Type ><Study Name><Data Source Name><Specimen Vendor
Number><Wallet Name>1
```

For example,

```
sh cgi_loader.sh "masterVarBeta-NA19240-L2-200-37-ASM
small.tsv"ODB_LOAD" "'Homo sapiens'" STUDY1" "CDM"
"vendor2","db004_w",1
```

The following command will be used to execute the shell script to upload the CGI Data when Oracle Wallet is not set up.

```
sh CGI_loader.sh <CGI file Name ><Oracle Directory
Object><Species Type ><Study Name><Data Source Name><Specimen
Vendor Number><Schema Name>0<schema username>
```

For example,

```
sh CGI_loader.sh "masterVarBeta-NA19240-L2-200-37-ASM
small.tsv"ODB_LOAD" "'Homo sapiens'" STUDY1" "CDM"
"vendor2","db004",0,gdm
```

Note: In Linux, it is not required to use 'Homo sapiens' in double quotes. That requirement is only for Windows.

Table 4–3 Mapping of CGI Result File

Column Name	Table and Column Name in ODB	Description
chromosome	W_EHA_RSLT_ NOCALL.CHROMOSOME_ WID W_EHA_RSLT_ SEQUENCING.CHROMOSO ME_WID W_EHA_ VARIANT.CHROMOSOME	For no-call results this is stored directly in the CHROMOSOME field. For non-reference alleles, this field is used with the begin position to find the correct DNA_ SOURCE record to find a VARIANT record or create a VARIANT record. Three values are needed to find existing VARIANT records. The chromosome, the begin position and the replace tag which is notation combining reference and allele sequences. For NOVEL variants, a new record is created in W_EHA_ VARIANT table with chromosome value.
begin	W_EHA_RSLT_ NOCALL.START_POSITION W_EHA_RSLT_ SEQUENCING.START_ POSITION W_EHA_VARIANT_ STG.START_POSITION W_EHA_ VARIANT.ABSOLUTE_ POSITION	The value of this field must add 1 since CGI uses zero based offsets and all other references use one based offsets. This field is used as described above. For no-call results this is stored in the START_POSITION field (after adding 1). For NOVEL variants, 'begin' is stored as it is ABSOLUTE_ POSITION column in W_ EHA_VARIANT table, while W_EHA_VARIANT.START_ POSITION is calculated relative to W_EHA_DNA_ SOURCE.START_POSITION using the 'begin' value.
end	W_EHA_RSLT_ NOCALL.END_POSITION W_EHA_VARIANT_ STG.END_POSITION	This value is used for no-call results and stored in the END_ POSITION field. This value is also used to calculate the relative end position based on W_EHA_DNA_ SOURCE.START_POSITION for END_POSITION in W_ EHA_VARIANT.END_ POSITION for novel variants.
zygosity	W_EHA_RSLT_ SEQUENCING.ZYGOSITY	Stored for sequencing alleles in ZYGOSITY field.
varType	W_EHA_RSLT_ SEQUENCING. VARIANT_ TYPE W_EHA_RSLT_NOCALL. NOCALL_TYPE	Stored either in NOCALL_ TYPE or VARIANT_TYPE.

Table 4–3 (Cont.) Mapping of CGI Result File

Column Name	Table and Column Name in ODB	Description
reference	W_EHA_VARIANT. REPLACE_TAG	This value is used in conjunction with allele1Seq and allele2Seq to construct a replace tag value used to find existing VARIANT records in W_EHA_VARIANT table. For CGI there are two overlap value computed for two alleles of each row of the result file. Using that overlap value, the reference sequence and the allele sequence is shortened and the start position and end position is incremented. This overlap value is used to create a replace tag with shortened reference and allele sequence. For insertions, the reference sequence uses a "-" and for deletions the allele sequence uses "-". This is standard notation used in most references. At some in-dels the representation can be as follows: ins can be "AT/ATCTA" and del can be "ATCTA/AT". The logic for checking and inserting variants into the file is in the called procedure, the procedure should handle varying representations for variants coming from any of the sequencing file types This field in some cases is empty for insertions.
allele1Seq	W_EHA_ VARIANT.REPLACE_TAG W_EHA_RSLT_ SEQUENCING_X.. ALLELE/ W_EHA_RSLT_ SEQUENCING_X. ALLELE_ CLOB W_EHA_RSLT_NOCALL_X. ALLELE/ W_EHA_RSLT_ NOCALL_X. ALLELE_CLOB	For sequencing results, this value is used to construct the replace tag. In some cases, this field is empty for deletions.

Table 4–3 (Cont.) Mapping of CGI Result File

Column Name	Table and Column Name in ODB	Description
allele2Seq	W_EHA_ VARIANT.REPLACE_TAG W_EHA_RSLT_ SEQUENCING_X. ALLELE/ W_EHA_RSLT_ SEQUENCING_X. ALLELE_ CLOB W_EHA_RSLT_NOCALL_X. ALLELE/ W_EHA_RSLT_ NOCALL_X. ALLELE_clob	For sequencing results, this value is used to construct the replace tag. In some cases, this field is empty for deletions.
allele1VarScoreVAF	W_EHA_RSLT_ SEQUENCING.SCORE_VAF	Used for sequencing results and stored in the SCORE_VAF field. If the variant is homozygous, then as only single allele record is stored, in such cases the least score value out of two scores is stored.
Allele2VarScoreVAF	W_EHA_RSLT_ SEQUENCING.SCORE_VAF	Used for sequencing results and stored in the SCORE_VAF field. If the variant is homozygous, then as only single allele record is stored, in such cases the least score value out of two scores is stored.
allele1VarScoreEAF	W_EHA_RSLT_ SEQUENCING.SCORE_EAF	Used for sequencing results and stored in the SCORE_EAF field. If the variant is homozygous, then as only single allele record is stored, in such cases the least score value out of two scores is stored.
Allele2VarScoreEAF	W_EHA_RSLT_ SEQUENCING.SCORE_EAF	Used for sequencing results and stored in the SCORE_EAF field. If the variant is homozygous, then as only single allele record is stored, in such cases the least score value out of two scores is stored.
allele1HapLink	W_EHA_RSLT_ SEQUENCING_X.HAPLINK W_EHA_RSLT_NOCALL_ X.HAPLINK	Used for both no-call and sequencing results and stored in the HAPLINK field.
allele2HapLink	W_EHA_RSLT_ SEQUENCING_X.HAPLINK W_EHA_RSLT_NOCALL_ X.HAPLINK	Used for both no-call and sequencing results and stored in the HAPLINK field.
allele1ReadCount	W_EHA_RSLT_ SEQUENCING. ALLELE_ READ_COUNT	Used for sequencing results and stored in the ALLELE_ READ_COUNT field.

Table 4–3 (Cont.) Mapping of CGI Result File

Column Name	Table and Column Name in ODB	Description
Allele2ReadCount	W_EHA_RSLT_SEQUENCING. ALLELE_READ_COUNT	Used for sequencing results and stored in the ALLELE_READ_COUNT field.
referenceAlleleReadCount	W_EHA_RSLT_SEQUENCING. REFERENCE_READ_COUNT	Used for sequencing results and stored in the REFERENCE_READ_COUNT field.

4.6 MAF Sequence Data Loader

Mutation Annotation Format (MAF) is created by TCGA. MAF files store variation data for multiple samples.The MAF file format is described here:

http://tcga-data.nci.nih.gov/tcga/dataAccessMatrix.htm

The format of a MAF file is tab-delimited columns. This file is the simplest among all result files. ODB supports import of MAF from 2.0 to 2.2 versions.

4.6.1 Files to Load

The execution call of the stored procedure `odb_result_util.process_maf()` is designed in one of the script files (load_maf.sql). This stored procedure will accept FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, DATA SOURCE and SPECIMEN VENDOR as an input parameter.

This stored procedure will create an external table dynamically and upload data from the source file into it. External tables allow Oracle to query data that is stored outside the database in flat files. The ORACLE_LOADER driver can be used to access data stored in any format that can be loaded by the SQL*Loader. No DML can be performed on external tables but they can be used for query, join and sort operations.

Only one external table is created dynamically and will hold the complete result data. A Global Temporary table named W_EHA_MAF_SPECIMEN is created explicitly to store the Normal and Tumor sample barcodes. There will be two pass through the MAF file. One will create a W_EHA_VARIANT_STG record and collect each unique specimen number into the global temporary table. The bulk collect will then call Odb_util.GET_SPECIMEN_WID for all of the specimen numbers in one statement.

Another bulk insert statement will then insert the data into W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING _X. After inserting the record into W_EHA_VARIANT_STG table, a PROCESS_VARIANT() procedure will be called which will populate the W_EHA_VARIANT table.

4.6.2 Data Load

Each row of the MAF file has two allele information for two sample types — tumor and normal sample. These two sample IDs are specified in each row of the file. Sample ID of tumor is specified in "Tumor_Sample_Barcode" and that of normal is specified in "Matched_Norm_Sample_Barcode" column respectively. So, for a single row, there can be a maximum of eight records created in ODB, depending on the heterozygosity and resemblance with reference sequence. Four for tumor and four for Normal sample.

For allele sequences, "-" for a deletion represent a variant. "-" for an insertion represents wild-type allele. If an allele sequence is the same as the Reference_Allele sequence, then that allele information is not stored in the data bank. There is no information on NOCALL in MAF data hence all the data will go to W_EHA_ VARIANT ,W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SEQUENCING_X tables.

Note: The batch file requires Oracle Wallet to be set up to run correctly.

The batch file requires the following parameters to be passed in order. Any spaces present in the parameters should be enclosed within "" :

- File name of the MAF data file
- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Wallet Name

The following command will be use to execute the batch file to upload the MAF data.

```
C:\> MAF_loader.bat <MAF file Name><File Type Code><File
Version><Vendor Name><Data Source Name><Specimen Type><"Specimen
Number "><Specimen Vendor Number><Schema Name>
```

For example,

```
d:\>maf_loader.bat "hgsc.bcm.edu__Applied_Biosystems_Sequence_
data_level3","" "ODB_LOAD" "'Homo sapiens'" "STUDY1" "CDM"
"vendor3" "db004_w"
```

The shell script requires the following parameters to be passed in order. Any spaces present in the parameters should be enclosed within "" :

- File name of the MAF data file
- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Schema/Wallet Name- If Oracle Wallet is set up enter the Wallet name else enter the schema name 1 if Oracle Wallet is set up else 0
- If previous parameter was 0 then enter schema username

The shell script can be run with or without Oracle Wallet being set up. The following command will be used to execute the shell script to upload the MAF Data when Oracle Wallet is set up.

```
Sh maf_loader.sh <MAF file Name><File Type Code><File
Version><Vendor Name><Data Source Name><Specimen Type><"
"><Specimen Vendor Number><Schema Name>1
```

For example,

```
sh maf_loader.sh "hgsc.bcm.edu__Applied_Biosystems_Sequence_
data_level3.maf" "ODB_LOAD" "'Homo sapiens'" STUDY1" "CDM"
"vendor2","db004",1
```

The following command will be used to execute the shell script to upload the MAF Data when Oracle Wallet is not set up.

```
sh maf_loader.sh <MAF file Name ><Oracle Directory
Object><Species Type><Study Name><Data Source Name><Specimen
Vendor Number><Schema Name>0<schema username>
```

For example,

```
sh maf_loader.sh "hgsc.bcm.edu__Applied_Biosystems_Sequence_
data_level3.maf" "ODB_LOAD" "'Homo sapiens'" STUDY1" "CDM"
"vendor2","db004",0,gdm
```

Table 4–4 Mapping of MAF Result File

Column Name in Result File	Table and Column Name in ODB	Description
Chromosome	W_EHA_RSLT_SEQUENCING.CHROMOSOME_WID W_EHA_VARIANT.CHROMOSOME	This field is used with the begin position to find the correct DNA_SOURCE record to find a VARIANT record or create a VARIANT record. Note that 3 values are needed to find existing VARIANT records. The chromosome, the begin position and the replace tag which is notation combining reference and allele sequences. For NOVEL variants, a new record is created in W_EHA_VARIANT table with chromosome value.
Start_Position	W_EHA_RSLT_SEQUENCING.START_POSITION W_EHA_VARIANT_STG.START_POSITION W_EHA_VARIANT.ABSOLUTE_POSITION	This field is used as described above. For no-call results this is stored in the START_POSITION field (after adding 1). For NOVEL variants, 'Start_Position' is stored as it is ABSOLUTE_POSITION column in W_EHA_VARIANT table, while W_EHA_VARIANT.START_POSITION is calculated relative to W_EHA_DNA_SOURCE.START_POSITION using the 'Start_Position' value.
End_Position	W_EHA_VARIANT_STG.END_POSITION	This value is used for no-call results and stored in the END_POSITION field. This value is also used to calculate the relative end position based on W_EHA_DNA_SOURCE.START_POSITION for END_POSITION in W_EHA_VARIANT.END_POSITION for novel variants.

Table 4–4 (Cont.) Mapping of MAF Result File

Column Name in Result File	Table and Column Name in ODB	Description
Strand	W_EHA_VARIANT.STRAND W_EHA_VARIANT_STG.STRAND	This value is used to indicate forward or reverse strand.
Variant_Type	W_EHA_RSLT_SEQUENCING.VARIANT_TYPE	Type of variant including snp, insertion, or deletion. Stored in VARIANT_TYPE in W_EHA_RSLT_SEQUENCING table.
Reference_Allele	W_EHA_VARIANT.REPLACE_TAG	This value is used for REPLACE_TAG, REPLACE_TAG is used twice, one for the tumor and the other for normal sample. This value is used in conjunction with Tumor_Seq_Allele1 and Tumor_Seq_Allele2 to find existing VARIANT records for tumor sample. Similarly for normal sample the Reference allele is used for REPLACE_TAG. Note that for insertions the reference sequence uses a "-" and for deletions the allele sequence uses "-". This is standard notation used in most references. Logic for loader will be implemented in called procedure to variant table. For deletion this value has deleted sequence and for insertion it has "-".
Tumor_Seq_Allele1	W_EHA_RSLT_SEQUENCING_X.ALLELE/ W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_VARIANT.REPLACE_TAG	For sequencing results this value is used to construct the replace tag. '-' value represents a deletion.
Tumor_Seq_Allele2	W_EHA_RSLT_SEQUENCING_X.ALLELE / W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_VARIANT.REPLACE_TAG	For sequencing results this value is used to construct the replace tag. '-' value represents a deletion.
Tumor_Sample_Barcode	W_EHA_RSLT_SPECIMEN.SPECIMEN_NUMBER	This value represents tumor sample ID This barcode ID involves TCGA-SiteID-PatientID-SampleID-PortionID-PlateID-CenterID.

Table 4–4 (Cont.) Mapping of MAF Result File

Column Name in Result File	Table and Column Name in ODB	Description
Matched_Norm_Sample_ Barcode	W_EHA_RSLT_ SEQUENCING.RESULT_ SPEC_WID W_EHA_RSLT_ SEQUENCING.SPECIMEN_ WID	This value represents normal sample ID This barcode ID involves TCGA-SiteID-PatientID-SampleID-PortionID-PlateID-CenterID. The complete barcode ID as is foreign key to RSLT_ SPECIMEN record.
Match_Norm_Seq_Allele1	W_EHA_RSLT_ SEQUENCING_X.ALLELE / W_EHA_RSLT_ SEQUENCING_X.ALLELE_ CLOB W_EHA_ VARIANT.REPLACE_TAG	For sequencing results this value is used to construct the replace tag. . '-' value represents a deletion.
Match_Norm_Seq_Allele2	W_EHA_RSLT_ SEQUENCING_X.ALLELE / W_EHA_RSLT_ SEQUENCING_X.ALLELE_ CLOB W_EHA_RSLT_ SEQUENCING.ALLELE W_EHA_ VARIANT.REPLACE_TAG	For sequencing results this value is used to construct the replace tag. . '-' value represents a deletion.
Score	W_EHA_RSLT_ SEQUENCING.SCORE_VAF	This is mapped to W_EHA_ RSLT_ SEQUENCING.SCORE_VAF.

4.7 VCF Sequence Data Loader

The VCF file format is described here:

http://tcga-data.nci.nih.gov/tcga/dataAccessMatrix.htm

The mapping currently is as per the 1000 genomics genotype data supporting v4.1. Two kinds of VCF files are available at 1000 Genomes, *sites* and *genotypes*. Sites file does not contain genotype data and sample details. However, the genotype vcf file contains individual genotype data along with the sample information. Hence the current loader supports genotype vcf files from 1000 genomes. The current model will support only SNP information and will not support Structural Variation including InDel whose representation differs from SNP in VCF file.

4.7.1 Files to Load

The execution call of the stored procedure `odb_result_util.process_vcf()` is designed in one of the script files (load_vcf.sql). fileThis stored procedure will accept FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, DATA SOURCE and SPECIMEN VENDOR as an input parameter.

This stored procedure will create an external table dynamically and upload data from the source file into it. External tables allow Oracle to query data that is stored outside the database in flat files. The ORACLE_LOADER driver can be used to access data stored in any format that can be loaded by the SQL*Loader. No DML can be performed on external tables but they can be used for query, join and sort operations. The loader will support 6000 specimens per VCF file.

The stored procedure will dynamically create seven external tables, where each table stores the common columns like chromosome, ref and so on along with 990 specimens. The first external table will store common attributes and the first 990 specimens, the second external table will store the common attributes and the second 990 specimens. Dynamic sql is used to first parse the header row in the data file and store it in an external table named VCF_SPEC_!!SEQ!! that will contain eight header columns. These columns are then appended and parsed to obtain the specimen identifiers. The second external table named VCF_DATA_!!SEQ!! will store the complete result data. This table will map all the fields existing in the result file.

Note: In both the external tables the"!!SEQ!!" string will be replaced by ETL_PROC_ID at the run time.

There will be two multi-table insert statements written dynamically. One will insert records into W_EHA_VARIANT_STG and other will insert record into W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SEQUENCING_X tables.

A select statement which inserts data into W_EHA_VARIANT_STG will compute the overlap value comparing reference with allele sequence. Using that overlap value, the reference sequence and the allele sequence is shortened and the start position and end position is incremented. Also this overlap value is used to create a replace tag with shortened reference and allele sequence.

After inserting the record into W_EHA_VARIANT_STG table, a `PROCESS_VARIANT()` procedure will be called which will populate the W_EHA_VARIANT table.

Another dynamic sql is used to parse the data columns from the data file into an external table (VCF_DATA_!!SEQ!!) and is also used to populate the data from the external table into W_EHA_RSLT_SEQUENCING and W_EHA_SEQUENCING_X tables using multi insert statements.

The procedure allows to load any number of samples in batches of 30.The dynamic sql to create and load the data into an external table and populate the W_EHA_RSLT_SEQUENCING and W_EHA_SEQUENCING_X tables.

4.7.2 Data Load

Two kinds of VCF files are available at 1000 Genomes, namely sites and genotypes. Sites file does not contain genotypic data and sample details whereas the genotype vcf file contains individual genotypic data along with sample information. Therefore, the current loader supports genotype vcf files from 1000 genomes. The current model will support only SNP information including InDels and will not support Structural Variation whose representation differs from SNP in VCF file. If the user has a merged file containing SNP, INDEL and SV data, then the SV data should be removed from the file to load results.

The sample information is present on the header row of the VCF data following the "FORMAT" column. Each row represents one sample.

Data type representation format and its order for each sample is specified in the "FORMAT" column. All the alleles for all samples are stored in the ALT column, but to get the allele information for each sample the GT identifier from the FORMAT column for each sample is used. The allele value is represented in numerals (for example, 0/1, 1/2), where '0' represents reference allele and '1' and '2' represent alleles specified order in ALT.

Note: The batch file requires Oracle Wallet to be set up to run correctly.

The batch file requires the following parameters to be passed in order. Any spaces present in the parameters should be enclosed within "" :

- File name of the VCF data file
- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Wallet Name

The following command will be used to execute the batch file to upload the VCF data.

```
C:\> VCF_loader.bat <VCF file Name><Oracle Directory
Object><Study name><Data Source Name><Species Name><Specimen
Vendor Number><Wallet Name>
```

For example,

```
d:\>vcf_loader.bat "d:\trc-v2\testdata\ ALL.chr20.merged_beagle_
mach.20101123.snps_indels_svs.genotypes.vcf","ODB_
LOAD","Study1","cdm","Homo sapiens"," ","vendor3","db004_w"
```

The shell script requires the following parameters to be passed in order. Any spaces present in the parameters should be enclosed within "" :

- Name of the VCF data file
- Oracle Directory Object
- Study name
- Datasource Name
- Species Name
- Specimen Vendor Number
- Schema/Wallet Name- If Oracle Wallet is set up enter the Wallet name else enter the schema name
- 1 if Oracle Wallet is set up else 0
- If previous parameter was 0 then enter schema username

The shell script can be run with or without Oracle Wallet being set up. The following command will be used to execute the shell script to uploadthe VCF Data when Oracle Wallet is set up.

```
sh vcf_loader.sh <VCF file Name><Oracle Directory Object><Study
Name><Data Source Name><Species name><Specimen Vendor
Number><Schema Name>1
```

For example,

```
sh vcf_loader.sh "/trc-v2/testdata/ ALL.chr20.merged_beagle_
mach.20101123.snps_indels_svs.genotypes.vcf","ODB_
LOAD","Study1","cdm","Homo sapiens"," ","vendor3","db004_w",1
```

The following command will be used to execute the shell script to upload the VCF Data when Oracle Wallet is not set up.

```
sh vcf_loader.sh <VCF file Name><Oracle Directory Object><Study
Name><Data Source Name><Species Name><Specimen Vendor
Number><Schema Name>0<schema username>
```

For example,

```
sh vcf_loader.sh "/trc-v2/testdata/ ALL.chr20.merged_beagle_
mach.20101123.snps_indels_svs.genotypes.vcf","ODB_
LOAD","Study1","cdm","Homo sapiens"," ","vendor3","db004_
w",0,gdm
```

Table 4–5 Mapping of VCF Result File

Column Name in Result File	Table and Column Name in ODB	Description
CHROM	W_EHA_ VARIANT.CHROMOSOME W_EHA_RSLT_ SEQUENCING_ CHROMOSOSME_WID	This field is used with the POS to find the correct DNA_ SOURCE record to find a VARIANT record or create a VARIANT record. Note that 3 values are needed to find existing VARIANT records. The chromosome, the POS and the replace tag which is notation combining reference and allele sequences. For NOVEL variants, a new record is created in W_EHA_ VARIANT table with chromosome value.
POS	W_EHA_VARIANT_ STG.START_POSITION W_EHA_VARIANT_ STG.END_POSITION W_EHA_ VARIANT.ABSOLUTE_ POSITION	This field is used as described above in Chromosome column. For NOVEL variants, 'POS' is stored as it is ABSOLUTE_POSITION column in W_EHA_VARIANT table, while W_EHA_ VARIANT.START_POSITION is calculated relative to W_ EHA_DNA_SOURCE.START_ POSITION using the 'POS' value. Since, VCF does not have the end position information, while inserting novel variants in VARIANT table, END_ POSITION need to be calculated based on POS information and number of bases in REF.

Table 4–5 (Cont.) Mapping of VCF Result File

Column Name in Result File	Table and Column Name in ODB	Description
REF	W_EHA_ VARIANT.REPLACE_TAG	This value is used in conjunction with ALT sequence to construct a replace tag value used to find existing VARIANT records. The replace tag is constructed with reference first followed by "/" and then the allele sequence. Note that for insertions the reference sequence uses a "-" and for deletions the allele sequence uses "-". This is standard notation used in most references. At some in-dels the representation can be as follows: ins can be "AT/ATCTA" and del can be "ATCTA/AT". The logic will be implemented in procedure which can be called with any of the above formats.
ALT	W_EHA_RSLT_ SEQUENCING_X.ALLELE W_EHA_ VARIANT.REPLACE_TAG	For sequencing results, this value is used to construct the replace tag and to store the value in the results table as per rules.
FILTER	W_EHA_RSLT_ SEQUENCING_X.FILTER	All variants, whether "PASS" or "FAIL" shall be loaded. "PASS/FAIL". Based on quality value and % of samples having data. Eg: "q10;s50". If filtering is not done then the field has "." (also load).
INFO	This field is not mapped with any database column. But will be used in cursor to filter out the SV data.	This is mainly used for integrated VCF files which contains SNP, INDEL and SV. The first field before semicolon specifies the variation type ID This will be used to filter out the SV data from the integrated VCF file.

Table 4–5 (Cont.) Mapping of VCF Result File

Column Name in Result File	Table and Column Name in ODB	Description
FORMAT.GT	W_EHA_ VARIANT.REPLACE_TAG W_EHA_RSLT_ SEQUENCING_X.ALLELE	Used to get the allele information for each sample. It is represented as '<allele1_ num>/<allele2_num>'. In some cases instead of "/" there could be "|". 1. For diploid: "0|0" represents both the alleles from REF. 2. "0|1" represents one allele from REF and other from ALT allele. 3. "0/2" represents one allele from REF and other from ALT allele 2. 4. For haploid: only one allele number is represented. 5. '.' is specified if a call cannot be made for a sample at that locus.
FORMAT.FT	W_EHA_RSLT_ SEQUENCING_X. GENOTYPE_FILTER	Sample genotype filter indicating if this genotype was "called" (similar in concept to the FILTER field). Again, use PASS to indicate that all filters have been passed, a semi-colon separated list of codes for filters that fail, or"." to indicate that filters have not been applied. These values should be described in the meta-information in the same way as FILTERs (String, no white-space or semi-colons permitted).
FORMAT.GQ	W_EHA_RSLT_ SEQUENCING.SCORE_VAF	This is mapped to W_EHA_ RSLT_ SEQUENCING.SCORE_VAF

4.8 Typical Errors Associated with Result Loaders (CGI, MAF, VCF, Gene Expression)

For each loader, a new VARCHAR2 (100) variable is defined. This variable is set before each and every SQL call to specify the context information. It is then used in standard error logging where the RECORD_DETAIL column of W_EHA_RSLT_ERR_LOG table will be populated with a simple context error message. Some common error messages are 'Generating ETL PROC ID', 'Verifying result file type and creating result file record', 'Verifying Species Name', 'Verifying Study Name', 'Processing Variant Staging records', 'Processing variant records', 'Processing result records', 'Dropping external tables'.

Examples of common error log records which are used in CGI, MAF, and VCF loaders are shown in Table 4–6.

Table 4–6 Columns Generated while Parsing CGI Files

Column Name	Description	Examples
ROW_WID	Record identifier	
RESULT_TYPE_NAME	Result Type Name	For CGI — CGI masterVar
SPECIES_NAME	Species Name	Homo sapiens
SPECIMEN_NUMBER	Specimen Number parsed from the comments section of file.	GS00706-DNA_C01
SPECIMEN_VENDOR_ NUMBER	Name of vendor passed as parameter with batch file	For CGI : CGI
DATASOURCE_NM	Data source Name	CDM
ERROR_DESC	Error message	ORA-01403 NO DATA FOUND
RECORD_DETAIL		Verifying Study Name
ETL_PROC_WID	Each load identifier.	

Following are some errors which are handled as per the loader's processes.

CGI

When an external table processing for Specimen Number fails, an error will be logged as **Processing external table for specimen header**. When the external table processing for the entire CGI result record fails, an error will be logged as **Processing external table for specimen data**. If the process fails to retrieve the specimen ID from W_EHA_ RSLT_SPECIMEN table, the error **Retrieving specimen number from external table** will be logged along with the ORA error in W_EHA_RSLT_ERR_LOG table.

VCF

Similar to CGI, when an external table processing for Specimen Number fails, an error will be logged as **Processing external table for specimen header**. If the process fails to retrieve the specimen ID from W_EHA_RSLT_SPECIMEN table, an error **Retrieving specimen number from external table** will be logged along with the ORA error in W_ EHA_RSLT_ERR_LOG table.

After retrieving the specimen ID from the header table, if the process fails while dropping the external header table, an error **Dropping specimen header external table** will be logged into the error table. For VCF, a batch of 30 specimens are processed at a time, so if the process fails the first time then an error **Processing data table for first set of specimen** will be logged into the error table.

After processing each batch of specimens, the loader drops the external table and recreates for the next set of specimens. If the process fails while dropping the external table, then the error **Processing data table for first set of specimen** will get logged into the error table. If the process fails during the next specimen batch processing, then **Processing data table for next set of specimen** will be logged into error table.

At the end of the result processing, the loader drops the last version of external table. If the process fails, the error **Dropping last version of data table** will be logged into error table.

MAF

Only one external table is created for MAF result data. If the process fails at this step, then the error **Processing MAF external data table** will be logged into error table. A first BULK insert statement will create a variant staging and specimen record. If the process fails at this stage, then the error **Processing variant staging and specimen staging record** will be logged into the error table.

If the process fails while retrieving the specimen ID for a record from global temporary table, an error **Processing variant staging and specimen staging record** will be logged into error table.

Gene Expression

If the process fails while retrieving the specimen ID, an error **Verifying Specimen ID** will be logged into the error table. If the process fails while the processing the hybridization header external table, the **Processing hybridization header table** will be logged. If the process fails while retrieving the hybridization name from the header table then the error **Retrieving hybridization name from header table** will be logged. Similar to VCF, a batch of 45 expression data will be processed at a time and if the process fails at this stage, then an error **Processing data table for the set of gene expression** will be logged. If the process fails while dropping expression and hybridization external tables, then an error messages **Dropping expression data table** and **Dropping hybridization table** respectively, will be logged.

Note: For the above loaders, the REC_DETAILS column of W_EHA_RSLT_ERR_LOG table will only be described the context in which the process failed.

4.9 Template of Command Line Arguments for Result Loaders

This section provides a summary of templates for running each data loader, along with an example using sample summary input files.

4.9.1 Glossary of Parameter Templates

The following is a glossary of common data entries and a brief description of their point of origin.

Table 4–7 Glossary of Parameter Templates

Parameter Name	Description
user	DB user name
host_url:dbname	DB connection string (include system-add:port:dbname). to be used if no Wallet is provided.
schema_name	DB instance name
species_file	File with a list of Species names
input_file(.dat)	Input File, with path if not present in same folder (File type)
wallet_name	Name of Wallet created
sqlnet.ora_path	Path to directory with file sqlnet.ora and tnsnames.ora
wallet_dir	Directory with wallet files

Table 4–7 (Cont.) Glossary of Parameter Templates

Parameter Name	Description
species_wid	Primary Key ID from W_EHA_SPECIES
file_type_Code	File_type_code from W_EHA_RSLT_FILE_TYPE
file_version	File_type_version from W_EHA_RSLT_FILE_TYPE
vendor_name	User input of Vendor name for Result files
data_source_name	datasource_name from W_EHA_DATASOURCE
species_name	species_name from W_EHA_SPECIES
specimen_number	If datasource is CDM, give the SPECIMEN_NUMBER under W_EHA_SPECIMEN_PATIENT_H. (Has to be present in CDM table, but should only be given for gene_expression_loader.)
specimen_vendor_number	If datasource is CDM, give the SPECIMEN_VENDOR_NUMBER under W_EHA_SPECIMEN_PATIENT_H. (Has to be present in CDM table.)

4.9.2 Probe Annotation

Loads probe annotation files to W_EHA_PROBE table

Without Oracle Wallet:

Linux:

```
>sh probe_loader.sh <input_file(text)> <schema_name> 0 <user>
```

For example,

```
>probe_loader.sh homo-sapiens-9606-gene-symbol.gmt DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>probe_loader.bat <input_file(text)> <wallet_name>
```

Linux:

```
>sh probe_loader.sh <input_file(text)> <wallet_name> 1
```

Example:

```
>probe_loader.bat probeset_annotation_summary_ref.txt
DB001Wallet
```

4.9.3 Gene Expression

Loads probe annotation files to W_EHA_PROBE table.

Without Oracle Wallet:

Linux:

```
>sh gene_expression_loader.sh <input_file(text)> <oracle_
directory_object> <study_name> <data_source_name> <specimen_
number> <specimen_vendor_number> <schema_name> 0 <user>
```

Example:

```
>sh gene_expression_loader.sh mas5_expression_summary_part1.txt
"sapiensODB_LOAD" "study1""CDM" "exp01" "affy01" DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>gene_expression_loader.bat <input_file(text)> <oracle_
directory_object> <study_name> <data_source_name> <species_name>
<specimen_number> <specimen_vendor_number> <wallet_name>
```

Linux:

```
>sh gene_expression_loader.sh <input_file(text)> <oracle_
directory_object> <study_name> <data_source_name> <specimen_
number> <specimen_vendor_number> <wallet_name> 1
```

Example:

```
>sh gene_expression_loader.sh mas5_expression_summary_part1.txt
"ODB_LOAD" "study1" "CDM" "exp01" "affy01" DB001Wallet 1
```

4.9.4 CGI masterVar

Loads variation/mutation results from CGI masterVar files to RSLT_Sequencing and RSLT_Nocall tables.

Without Oracle Wallet:

Linux:

```
>sh CGI_loader.sh <input_file(.tsv)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <schema_name> 0 <user>
```

Example:

```
>sh CGI_loader.sh "masterVarBeta-NA19240-L2-200-37-ASM.tsv"
"ODB_LOAD" "Homo Sapiens" "Study1" "CDM" "CGI" DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>CGI_loader.bat <input_file(.tsv)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <wallet_name>
```

Linux:

```
>sh CGI_loader.sh <input_file(.tsv)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <wallet_name> 1
```

Example:

```
>sh CGI_loader.sh "masterVarBeta-NA19240-L2-200-37-ASM.tsv"
"ODB_LOAD" "Homo Sapiens" "Study1" "CDM" "CGI" DB001Wallet 1
```

4.9.5 MAF

Loads variation/mutation results from MAF files to RSLT_Sequencing and RSLT_Nocall tables.

Without Oracle Wallet:

Linux:

```
>sh MAF_loader.sh <input_file(.maf)> <file_type_Code> <file_
version> <vendor_name> <data_source_name> <species_name>
<specimen_number> <specimen_vendor_number> <schema_name> 0
<user>
```

Example:

```
>sh MAF_loader.sh "hgsc.bcm.edu__Applied_Biosystems_Sequence_
data_level3.maf" "MAF" "2.2" "MAF" "CDM" "Homo sapiens" "" "MAF"
DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>MAF_loader.bat <input_file(.maf)> <file_type_Code> <file_
version> <vendor_name> <data_source_name> <species_name>
<specimen_number> <specimen_vendor_number> <wallet_name>
```

Linux:

```
>sh MAF_loader.sh <input_file(.maf)> <file_type_Code> <file_
version> <vendor_name> <data_source_name> <species_name>
<specimen_number> <specimen_vendor_number> <wallet_name> 1
```

Example:

```
>sh MAF_loader.sh "hgsc.bcm.edu__Applied_Biosystems_Sequence_
data_level3.maf" "MAF" "2.2" "MAF" "CDM" "Homo sapiens" "" "MAF"
DB001Wallet 1
```

4.9.6 VCF

Loads variation/mutation results from VCF files to RSLT_Sequencing and RSLT_Nocall tables.

Without Oracle Wallet:

Linux:

```
>sh VCF_loader.sh <input_file(.vcf)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <schema_name> 0 <user>
```

Example:

```
>sh VCF_loader.sh "ALL.chr20.merged_beagle_mach.20101123.snps_
indels_svs.genotyp_copy.vcf" "ODB_LOAD" "Homo Sapiens" "Study1"
"CDM" "VCF" DB001 0 gdm
```

With Oracle Wallet:

Windows:

```
>VCF_loader.bat <input_file(.vcf)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <wallet_name>
```

Linux:

```
>sh VCF_loader.sh <input_file(.vcf)> <oracle_directory_object>
<species_name> <study_name> <data_source_name> <specimen_vendor_
number> <wallet_name> 1
```

Example:

```
>sh VCF_loader.sh "ALL.chr20.merged_beagle_mach.20101123.snps_
indels_svs.genotyp_copy.vcf" ODB_LOAD" "Homo Sapiens" "Study1
"CDM" "VCF" DB001Wallet 1
```

5 Model Dictionary

This chapter describes all entities in all the tables of Oracle Health Sciences Omics Data Bank. It contains the following topics:

5.1 System Columns

System fields common to all tables are described in the following table:

Table 5–1 System Columns

Table	Column	Data Type	Description	Required (Not null)
Any table	ROW_WID	NUMBER	System. Field will be set by sequence (not mapped)	Y
Any table	W_INSERT_DT	DATE	System. This column stores the date on which the record was inserted in the data warehouse table.	N
Any table	W_UPDATE_DT	DATE	System. This column stores the date on which the record was last updated in the data warehouse table.	N
Any table	ETL_PROC_WID	NUMBER	System. This column is the unique identifier for the specific ETL process used to create or update this data.	Y
Any table	ENTERPRISE_ID	NUMBER	System. Unique identifier for an enterprise in a multi-enterprise environment.	N

5.2 Reference Data Tables

The following table describes entities in the reference data tables:

Table 5–2 Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_HUGO_INFO	APPROVED_SYMBOL	VARCHAR2(200)	Approved Symbol for a gene in HUGO (Human Genome Organization). Also foreign key to W_EHA_GENE as HUGO_NAME.	Y
W_EHA_HUGO_INFO	HGNC_ID	VARCHAR2(100)	HUGO Gene Nomenclature Committee Identifier	N
W_EHA_HUGO_INFO	APPROVED_NAME	VARCHAR2(200)	Approved Name	N
W_EHA_HUGO_INFO	STATUS	VARCHAR2(50)	Status	N
W_EHA_HUGO_INFO	LOCUS_TYPE	VARCHAR2(50)	Locus Type e.g. RNA, complex locus constituent, fragile site	N
W_EHA_HUGO_INFO	LOCUS_GROUP	VARCHAR2(50)	Grouping locus types into supertypes, e.g protein-coding gene, non-coding RNA etc	N
W_EHA_HUGO_INFO	PREVIOUS_SYMBOLS	VARCHAR2(200)	Previous Symbols	N
W_EHA_HUGO_INFO	PREVIOUS_NAMES	VARCHAR2(1000)	Previous Names	N
W_EHA_HUGO_INFO	SYNONYMS	VARCHAR2(200)	Synonyms	N
W_EHA_HUGO_INFO	NAME_SYNONYMS	VARCHAR2(500)	Name Synonyms	N
W_EHA_HUGO_INFO	CHROMOSOME	VARCHAR2(20)	Chromosome	N
W_EHA_HUGO_INFO	APPROVED_DT	DATE	Date Approved	N
W_EHA_HUGO_INFO	MODIFIED_DT	DATE	Date Modified	N
W_EHA_HUGO_INFO	SYMBOL_CHANGED_DT	DATE	Date Symbol Changed	N
W_EHA_HUGO_INFO	NAME_CHANGED_DT	DATE	Date Name Changed	N
W_EHA_HUGO_INFO	ACCESSION_NUMBERS	VARCHAR2(200)	Accession Numbers	N

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_HUGO_INFO	ENZYME_IDS	VARCHAR2(200)	Enzyme IDs	N
W_EHA_HUGO_INFO	ENTREZ_GENE_ID	VARCHAR2(100)	Entrez database Gene ID	N
W_EHA_HUGO_INFO	ENSEMBL_GENE_ID	VARCHAR2(100)	Ensembl database Gene ID	N
W_EHA_HUGO_INFO	MGI_ID	VARCHAR2(100)	Mouse Genome Database Gene ID	N
W_EHA_HUGO_INFO	SPECIALIST_DB_LINKS	VARCHAR2(1000)	Specialist Database Links	N
W_EHA_HUGO_INFO	SPECIALIST_DB_IDS	VARCHAR2(500)	Specialist Database IDs	N
W_EHA_HUGO_INFO	PUBMED_IDS	VARCHAR2(200)	Pubmed database IDs	N
W_EHA_HUGO_INFO	REFSEQ_IDS	VARCHAR2(200)	RefSeq database IDs	N
W_EHA_HUGO_INFO	GENE_FAMILY_TAG	VARCHAR2(100)	Gene Family Tag	N
W_EHA_HUGO_INFO	GENE_FAMILY_DESC	VARCHAR2(200)	Gene family description	N
W_EHA_HUGO_INFO	RECORD_TYPE	VARCHAR2(50)	Record Type	N
W_EHA_HUGO_INFO	PRIMARY_IDS	VARCHAR2(200)	Primary Indentifiers separated by comma	N
W_EHA_HUGO_INFO	SECONDARY_IDS	VARCHAR2(200)	Secondary Identifiers separated by comma	N
W_EHA_HUGO_INFO	CCDS_IDS	VARCHAR2(400)	Consensus Coding Sequence (CCDS) Identifers	N
W_EHA_HUGO_INFO	VEGA_IDS	VARCHAR2(200)	Vertebrae Genome Annotation database IDs	N
W_EHA_HUGO_INFO	LOCUS_SPECIFIC_DBS	VARCHAR2(1000)	Locus Specific Databases	N

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_HUGO_INFO	MAPPED_GDB_ID	VARCHAR2(100)	Genome Database ID (mapped data)	N
W_EHA_HUGO_INFO	MAPPED_ENTREZ_GENE_ID	VARCHAR2(100)	Entrez Gene ID (mapped data supplied by NCBI)	N
W_EHA_HUGO_INFO	MAPPED_OMIM_ID	VARCHAR2(100)	Online Mendelian Inheritance in Man ID (mapped data supplied by NCBI)	N
W_EHA_HUGO_INFO	MAPPED_REFSEQ_ID	VARCHAR2(100)	National Center for Biotechnology Information (NCBI) Reference Sequence database (mapped data supplied by NCBI)	N
W_EHA_HUGO_INFO	MAPPED_UNIPROT_ID	VARCHAR2(100)	Universal Protein Resource database ID (mapped data supplied by UniProt)	N
W_EHA_HUGO_INFO	MAPPED_ENSEMBL_ID	VARCHAR2(100)	Ensembl genome database ID (mapped data supplied by Ensembl)	N
W_EHA_HUGO_INFO	MAPPED_UCSC_ID	VARCHAR2(100)	University of California, Santa Cruz database ID (mapped data supplied by UCSC)	N
W_EHA_HUGO_INFO	MAPPED_MGI_ID	VARCHAR2(100)	Mouse Genome Database ID (mapped data supplied by MGI)	N
W_EHA_HUGO_INFO	MAPPED_RGD_ID	VARCHAR2(100)	Rat Genome Database ID (mapped data supplied by RGD)	N
W_EHA_DNA_SOURCE	SPECIES_WID	NUMBER	Foreign Key to W_EHA_SPECIES table	Y
W_EHA_DNA_SOURCE	CHROMOSOME	VARCHAR2(50)	Chromosome identifier e.g. X; parsed out of each line in EMBL source file; 3rd section after 2nd colon.	N
W_EHA_DNA_SOURCE	ACCESSION	VARCHAR2(200)	Accession identifier for sequence, parsed out of each line in EMBL source file; the whole AC line contents, except for the 'chromosome:' prefix.	N
W_EHA_DNA_SOURCE	START_POSITION	NUMBER	Start location of sequence, parsed out of each line in EMBL source file; 4th section after 3rd colon.	N
W_EHA_DNA_SOURCE	END_POSITION	NUMBER	End position of sequence, parsed out of each line in EMBL source file; 5th section after 4th colon.	N

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ DNA_ SOURCE	RELEASED_DT	DATE	Released Date which is text that needs to be converted to DATE column.	N
W_EHA_ DNA_ SOURCE	VERSION	VARCHAR2(30)	Version of sequence, parsed out of each line in EMBL source file; 2nd section of AC line after 1st colon.	N
W_EHA_ DNA_ SOURCE	DNA_ SEQUENCE	CLOB	DNA sequence corresponding to all bases in the Start-End position (should match difference in length) This should end up with proper DNA base pairs as well as other letters for special meaning.	N
W_EHA_ DNA_ SOURCE	DNA_ SOURCE_ COMMENT	CLOB	Comment lines concatenated	N
W_EHA_ PROT_INFO	PROTEIN_WID	NUMBER	Foreign key W_EHA_ PROTEIN	Y
W_EHA_ PROT_INFO	PROTEIN_ NAME	VARCHAR2(200)	Name of Protein, set from each line using the keyword portion e.g. INS_HUMAN.	N
W_EHA_ PROT_INFO	ACCESSION	VARCHAR2(200)	Accession Identifier(s), set from the AC line. For this example the field would be set to P01308; Q5EEX2;	N
W_EHA_ PROT_INFO	RELEASED_DT	DATE	Released date, set from the DT line of the SwissProt file, while often there are multiple dates for different revisions, only the last date is used for update.	N
W_EHA_ PROT_INFO	PROT_INFO_ COMMENT	CLOB	Comments applicable to individual protein, as specified in the SwissProt reference file	N
W_EHA_ PROTEIN	SPECIES_WID	NUMBER	Foreign key to W_EHA_ SPECIES. Note that there is a SPECIES FK in this table and also in the DNA_SOURCE table. This is because the user can load SwissProt information separately from EMBL information. The SwissProt data would have no link to the DNA_SOURCE directly and the FK to SPECIES is needed for PROTEIN records as well.	Y
W_EHA_ PROTEIN	SEQUENCE_ CHECKSUM	VARCHAR2(32)	Checksum derieved from AMINO ACID SEQUENCE in order to quickly match up protein in SwissProt with Ensembl record	N

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ PROTEIN	SEQUENCE_ LENGTH	NUMBER	Length of AMINO ACID SEQUENCE	N
W_EHA_ PROTEIN	AMINO_ACID_ SEQUENCE	CLOB	Sequence of Amino Acids; This field is set from the SQ section of the SwissProt file or the "/translation" qualifier for coding segment (CDS) gene components.	N
W_EHA_ GENE	GENE_NAME	VARCHAR2(200)	Ensembl recognized gene identifier	Y
W_EHA_ GENE	HUGO_NAME	VARCHAR2(200)	Recognized HUGO gene name which can be used to look up proper chromosome information in the HUGO_ INFO table.	N
W_EHA_ GENE_ COMP_ SEGMENT	GENE_ COMPONENT_ WID	NUMBER	Foreign key to the parent gene component that has this segment definition.	Y
W_EHA_ GENE_ COMP_ SEGMENT	SOURCE_WID	NUMBER	Foreign key to the DNA source buffer that this segment refers to. Note that external source references will create DNA_SOURCE placeholder records that do not have the real sequence data loaded.	Y
W_EHA_ GENE_ COMP_ SEGMENT	NUMBER_IN_ SEQUENCE	NUMBER	Number to keep track of the order of each segment.	N
W_EHA_ GENE_ COMP_ SEGMENT	START_ POSITION	NUMBER	Stores the starting position of the segment.	N
W_EHA_ GENE_ COMP_ SEGMENT	END_ POSITION	NUMBER	Stores the end position of the segment.	N
W_EHA_ GENE_ COMP_ SEGMENT	COMPLEMENT	NUMBER(1)	This column Is used as a flag to indicate reverse transcription. Either 1 or 0.	N
W_EHA_ GENE_ COMPONE NT	STRUCTURE_ WID	NUMBER	This is the foreign key to the parent STRUCTURE record which is created for each gene component that has the same TRANSCRIPT_ID.	Y
W_EHA_ GENE_ COMPONE NT	COMPONENT_ TYPE	VARCHAR2(100)	This field stores the keyword described for the gene component. Possible values are CDS (coding sequence), exon, intron, etc.	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ GENE_ COMPONENT	PROTEIN_WID	NUMBER	Foreign key to the protein record. Most gene components are grouped together that are involved with some protein product. For example, it can link to a PROTEN record created for the Ensembl reference.	N
W_EHA_ GENE_ SEGMENT	GENE_WID	NUMBER	Foreign key to the parent GENE record. For example, FK can link to a GENE record with a name of "ENSG00000237037".	Y
W_EHA_ GENE_ SEGMENT	SOURCE_WID	NUMBER	Foreign key to the W_EHA_ DNA_SOURCE for each segment.	Y
W_EHA_ GENE_ SEGMENT	NUMBER_IN_ SEQUENCE	NUMBER	This field is used to number each gene segment as reported to construct the correct sequence of DNA bases for the gene definition.	N
W_EHA_ GENE_ SEGMENT	START_ POSITION	NUMBER	This field is the starting offset into the parent DNA_ SOURCE buffer.	N
W_EHA_ GENE_ SEGMENT	END_ POSITION	NUMBER	This field is the ending offset into the parent DNA_ SOURCE buffer.	N
W_EHA_ GENE_ SEGMENT	COMPLEMENT	NUMBER(1)	This field is used to indicate reverse transcription. For the example above this field will be set to 0.	N
W_EHA_ GENE_ STRUCTURE	GENE_WID	NUMBER	Foreign key to the parent GENE record.	Y
W_EHA_ GENE_ STRUCTURE	TRANSCRIPT_ ID	VARCHAR2(1000)	This field is used to store the value for TRANSCRIPT_ID used as the anchor for all of the gene components.	N
W_EHA_ PROT_ COMPONENT	PROTEIN_ INFO_WID	NUMBER	Foreign key to the parent PROTEIN INFO record.	Y
W_EHA_ PROT_ COMPONENT	COMPONENT_ TYPE	VARCHAR2(100)	This field is used to store the keyword used for each component of the protein molecule. Component types are detemined by what types exist in SwissProt file. Example types include: SITE, CHAIN, MOD_RES	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_PROT_COMPONENT	START_POSITION	VARCHAR2(100)	This field is used to indicate the start amino acid position in the protein molecule which identifies this component.	N
W_EHA_PROT_COMPONENT	END_POSITION	VARCHAR2(100)	This field is used to indicate the end amino acid position in the protein molecule which identifies this component.	N
W_EHA_PROT_COMPONENT	COMPONENT_DESC	VARCHAR2(2000)	This field is used to capture any remaining data on the line about the feature. There is often a description text which would simply be stored in this field.	N
W_EHA_SPECIES	SPECIES_NAME	VARCHAR2(200)	This field is derived from the OS line in the file. This will not include common names in parenthesis.	N
W_EHA_SPECIES	COMMON_NAME	VARCHAR2(200)	This field has the common names of an organism that would be parsed in the parenthesis from the OS line. For example, human.	N
W_EHA_SPECIES	PROMOTER_OFFSET	NUMBER	This field will be set on load of the database to allow for a default value to be used to compute promoter regions for each gene of this organism. Note that not every gene has documented promoter features in Ensembl. So each lab usually has some offset to compute an acceptable region before the start of the first coding segment to be used as a promoter region. Usually there is a different offset for Eukaryote and Prokaryote organisms.	N
W_EHA_VARIANT	SOURCE_WID	NUMBER	Foreign key to the W_EHA_DNA_SOURCE buffer used for reporting this information.	Y
W_EHA_VARIANT	START_POSITION	NUMBER	This field will store the starting position of which DNA base is referenced in the variant.	Y
W_EHA_VARIANT	END_POSITION	NUMBER	This field will store the ending position of which DNA base is referenced in the variant.	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ VARIANT	STATUS	VARCHAR2(100)	Status on variant, if downloaded from Ensembl then should be set to KNOWN, if uploaded from result files b/c found that it does not exist in Ensembl then it would be set to NOVEL	N
W_EHA_ VARIANT	REPLACE_TAG	VARCHAR2(1000)	The sequence of the reference. Used with W_ EHA_ALLELE_SEQ to indicate the changes in base pairs, deletions, or insertions. If insertion, it is set to '-'.	N
W_EHA_ VARIANT	STRAND	VARCHAR2(1)	Strand indicator, '+' for forward strand and '-' for reverse-complement strand	N
W_EHA_ VARIANT	CHROMOSOME	VARCHAR2(50)	Chromosome identifier e.g. X; parsed out of each line in gvf file for Known variants, for unknown, it is deduced based on position	N
W_EHA_ VARIANT	ABSOLUTE_ POSITION	NUMBER	Absolute position on the chromosome, obtained by adding the DNA buffer source position on the chromosome and START_ POSITION	N
W_EHA_ PATHWAY	PATHWAY_ SOURCE_ID	VARCHAR2(50)	This names the source of pathway as specified in pathwaycommons file. Example sources are: REACTOME, NCI_NATURE, CELLMAP	Y
W_EHA_ PATHWAY	PATHWAY_ NAME	VARCHAR2(200)	As specified in pathwaycommons.com file, 1st column, example: Nef Mediated CD4 Down-regulation	N
W_EHA_ PATHWAY_ PROTEIN	HUGO_ SYMBOL	VARCHAR2(200)	Foreign key into W_EHA_ HUGO_INFO table used to look up gene information.	Y
W_EHA_ PATHWAY_ PROTEIN	PATHWAY_ WID	NUMBER	Foreign key into W_EHA_ PATHWAY table	Y
W_EHA_ SOURCE_ LIT_REF	SOURCE_WID	NUMBER	Foreign key to parent record. Note that the name is different for each literature reference table.	Y
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ NUMBER	NUMBER	Number of the reference source from Ensembl file.	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ POSITION	VARCHAR2(2000)	RP line (Reference Position)	N
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ TITLE	VARCHAR2(2000)	RT line (Reference Title)	N
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ AUTHOR	VARCHAR2(2000)	RA line (Reference Author)	N
W_EHA_ SOURCE_ LIT_REF	CROSS_REF	VARCHAR2(2000)	RX line (Reference Cross Reference)	N
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ GROUP	VARCHAR2(2000)	RG line (Reference Group)	N
W_EHA_ SOURCE_ LIT_REF	REFERENCE_ LOCATION	VARCHAR2(2000)	RL line (Reference Location).	N
W_EHA_ PROT_LIT_ REF	PROTEIN_ INFO_WID	NUMBER	Foreign key to parent record. Note that the name is different for each literature reference table.	Y
W_EHA_ PROT_LIT_ REF	REFERENCE_ NUMBER	NUMBER	Number of the reference source from Ensembl file.	Y
W_EHA_ PROT_LIT_ REF	REFERENCE_ POSITION	VARCHAR2(2000)	RP line (Reference Position)	N
W_EHA_ PROT_LIT_ REF	REFERENCE_ TITLE	VARCHAR2(2000)	RT line (Reference Title)	N
W_EHA_ PROT_LIT_ REF	REFERENCE_ AUTHOR	VARCHAR2(2000)	RA line (Reference Author)	N
W_EHA_ PROT_LIT_ REF	CROSS_REF	VARCHAR2(2000)	RX line (Reference Cross Reference)	N
W_EHA_ PROT_LIT_ REF	REFERENCE_ GROUP	VARCHAR2(2000)	RG line (Reference Group)	N
W_EHA_ PROT_LIT_ REF	REFERENCE_ LOCATION	VARCHAR2(2000)	RL line (Reference Location).	N
W_EHA_ VARIANT_ QLFR	VARIANT_WID	NUMBER	Foreign key to parent record. Note that the name is different for each qualifier table.	Y
W_EHA_ VARIANT_ QLFR	QUALIFIER_ TAG	VARCHAR2(100)	Text after the "/" tag in Ensembl file and up until the equal sign. For example, "/note"	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ VARIANT_ QLFR	QUALIFIER_ VALUE	VARCHAR2(4000)	Any text after the equal sign in Ensembl reference file removing the double quotes. Note that some qualifiers can extend more than one line. The double quotes will indicate the end of the qualifier value.	N
W_EHA_ SOURCE_ QLFR	SOURCE_WID	NUMBER	Foreign key to parent record. Note that the name is different for each qualifier table.	Y
W_EHA_ SOURCE_ QLFR	QUALIFIER_ TAG	VARCHAR2(100)	Text after the "/" tag in Ensembl file and up until the equal sign. For example, "/note"	Y
W_EHA_ SOURCE_ QLFR	QUALIFIER_ VALUE	VARCHAR2(4000)	Any text after the equal sign in Ensembl reference file removing the double quotes. Note that some qualifiers can extend more than one line. The double quotes will indicate the end of the qualifier value.	N
W_EHA_ GENE_ QLFR	GENE_WID	NUMBER	Foreign key to parent record. Note that the name is different for each qualifier table.	Y
W_EHA_ GENE_ QLFR	QUALIFIER_ TAG	VARCHAR2(100)	Text after the "/" tag in Ensembl file and up until the equal sign. For example, "/note"	Y
W_EHA_ GENE_ QLFR	QUALIFIER_ VALUE	VARCHAR2(4000)	Any text after the equal sign in Ensembl reference file removing the double quotes. Note that some qualifiers can extend more than one line. The double quotes will indicate the end of the qualifier value.	N
W_EHA_ GENE_ COMP_ QLFR	GENE_ COMPONENT_ WID	NUMBER	Foreign key to parent record. Note that the name is different for each qualifier table.	Y
W_EHA_ GENE_ COMP_ QLFR	QUALIFIER_ TAG	VARCHAR2(100)	Text after the "/" tag in Ensembl file and up until the equal sign. For example, "/note"	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_GENE_COMP_QLFR	QUALIFIER_VALUE	VARCHAR2(4000)	Any text after the equal sign in Ensembl reference file removing the double quotes. Note that some qualifiers can extend more than one line. The double quotes will indicate the end of the qualifier value.	N
W_EHA_PROT_COMP_QLFR	PROT_COMPONENT_WID	NUMBER	Foreign key to parent record. Note that the name is different for each qualifier table.	Y
W_EHA_PROT_COMP_QLFR	QUALIFIER_TAG	VARCHAR2(100)	Text after the "/" tag in Ensembl file and up until the equal sign. For example, "/note"	Y
W_EHA_PROT_COMP_QLFR	QUALIFIER_VALUE	VARCHAR2(4000)	Any text after the equal sign in Ensembl reference file removing the double quotes. Note that some qualifiers can extend more than one line. The double quotes will indicate the end of the qualifier value.	N
W_EHA_VARIANT_XREF	VARIANT_WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_VARIANT_XREF	REFERENCE_ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y
W_EHA_VARIANT_XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y
W_EHA_VARIANT_XREF	REFERENCE_SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y
W_EHA_SOURCE_XREF	SOURCE_WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_SOURCE_XREF	REFERENCE_ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y
W_EHA_SOURCE_XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ SOURCE_ XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y
W_EHA_ PROT_ COMP_ XREF	PROT_ COMPONENT_ WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_ PROT_ COMP_ XREF	REFERENCE_ ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y
W_EHA_ PROT_ COMP_ XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y
W_EHA_ PROT_ COMP_ XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y
W_EHA_ GENE_XREF	GENE_WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_ GENE_XREF	REFERENCE_ ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y
W_EHA_ GENE_XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y
W_EHA_ GENE_XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y
W_EHA_ GENE_ COMPONE NT_XREF	GENE_ COMPONENT_ WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_ GENE_ COMPONE NT_XREF	REFERENCE_ ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y

Table 5–2 (Cont.) Reference Entities

Table	Column	Data Type	Description	Required (Not null)
W_EHA_ GENE_ COMPONE NT_XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y
W_EHA_ GENE_ COMPONE NT_XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y
W_EHA_ PROT_XREF	PROTEIN_WID	NUMBER	Foreign key to parent record. Note that this is different for each cross reference table.	Y
W_EHA_ PROT_XREF	REFERENCE_ ID	VARCHAR2(200)	This data is the 2nd part of the DB_XREF qualifier (after the first colon) and also the 2nd part of the DR line in Ensembl file.	Y
W_EHA_ PROT_XREF	DATABASE	VARCHAR2(100)	This data is the first part of the /DB_XREF and the DR line. For example, EMBL.	Y
W_EHA_ PROT_XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Some references will have a suffix portion after another colon. This data is stored here as it may be involved in filter criteria on the corresponding database.	Y

5.3 Result Data Tables

The following table describes entities from the result data tables:

Table 5–3 Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_PROBE	PRIMARY_HUGO_ NAME	VARCHAR2(200)	Hugo name for a gene corresponding to probe	N
W_EHA_PROBE	PROBE_NAME	VARCHAR2(200)	Name of gene probe, single probe should match to one gene only	N
W_EHA_PROBE	START_POSITION	NUMBER	Start position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_PROBE	SEQUENCE	VARCHAR2(200)	Sequence of bases for the particular probe	N
W_EHA_PROBE	ACCESSION	VARCHAR2(200)	List of accession Ids associated with the probe	N
W_EHA_PROBE	PROBE_DESC	VARCHAR2(2000)	Description of the gene for probe	N
W_EHA_PROBE_ XREF	PROBE_WID	NUMBER(38)	Foreign key to W_EHA_ PROBE	Y
W_EHA_PROBE_ XREF	REFERENCE_ID	VARCHAR2(200)	Gene ID associated to probe	Y

Table 5–3 (Cont.) Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_PROBE_ XREF	DATABASE	VARCHAR2(100)	Database for Gene ID	Y
W_EHA_PROBE_ XREF	REFERENCE_ SUFFIX	VARCHAR2(200)	Suffix applied to Gene IDs in a particular database, For example, _HUMAN	N
W_EHA_RSLT_ GENE_EXP	SPECIMEN_WID	NUMBER(38)	Foreign key to Specimen table in CDM1.1	Y
W_EHA_RSLT_ GENE_EXP	SPEC_DATASRC_ WID	NUMBER(38)	Foreign key to W_EHA_ DATASOURCE table	Y
W_EHA_RSLT_ GENE_EXP	PROBE_WID	NUMBER(38)	Foreign key to W_EHA_ PROBE	Y
W_EHA_RSLT_ GENE_EXP	HYBRIDIZATION_ NAME	VARCHAR2(200)	Hybridization experiment identifier in gene expression file	N
W_EHA_RSLT_ GENE_EXP	INTENSITY	NUMBER	Intensity value associated with particular probe at given hybridization	N
W_EHA_RSLT_ GENE_EXP	P_VALUE	NUMBER	P-value associated with particular probe at a given hybridization	N
W_EHA_RSLT_ GENE_EXP	CALL	VARCHAR2(200)	Call value associated with particular probe at a given hybridization, P for present, A for absent	N
W_EHA_ DATASOURCE	DATASOURCE_CD	VARCHAR2(80)	Data source for Specimen	Y
W_EHA_ DATASOURCE	DATASOURCE_NM	VARCHAR2(80)	Name of datasource for Specimen	N
W_EHA_ DATASOURCE	DATASOURCE_ DESC	VARCHAR2(2000)	Description of datasource for specimen	N
W_EHA_ DATASOURCE	SCHEMA_NAME	VARCHAR2(30)	Schema name for datasource for specimen	N
W_EHA_ DATASOURCE	DB_LINK_NAME	VARCHAR2(30)	Link to schema for specimen datasource	N
W_EHA_RSLT_ TYPE	RESULT_TYPE_ NAME	VARCHAR2(200)	Type of result For example, sequencing, no-call, gene expression, cnv	Y
W_EHA_RSLT_ TYPE	RESULT_TYPE_ DESC	VARCHAR2(2000)	Result type description	N
W_EHA_RSLT_FILE	FILE_TYPE_WID	NUMBER(38)	Foreign key to W_EHA_FILE_ TYPE	Y
W_EHA_RSLT_FILE	FILE_STORAGE_ FLG	VARCHAR2(1)	Type of file storage, S - Secure Files, E - External	N
W_EHA_RSLT_FILE	FILE_PATH	VARCHAR2(2000)	Path to file including filename	N
W_EHA_RSLT_FILE	VENDOR_NAME	VARCHAR2(200)	Name of vendor which supplied results in file, For example, Complete Genomics	N
W_EHA_RSLT_FILE	FILE_CONTENT_ID	RAW(128)	File identifier used in Secure File system	N
W_EHA_RSLT_FILE_ TYPE	FILE_TYPE_CODE	VARCHAR2(80)	Code for type of file	Y
W_EHA_RSLT_FILE_ TYPE	FILE_TYPE_NAME	VARCHAR2(200)	Full name of type of file	N

Table 5–3 (Cont.) Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_RSLT_FILE_TYPE	FILE_TYPE_DESC	VARCHAR2(2000)	Description of file	N
W_EHA_RSLT_FILE_TYPE	FILE_TYPE_VERSION	VARCHAR2(30)	Version of file format	Y
W_EHA_RSLT_SEQUENCING	RESULT_STUDY_WID	NUMBER(38)	Foreign key to RESULT_STUDY_WID in W_EHA_RSLT_STUDY	Y
W_EHA_RSLT_SEQUENCING	RESULT_SPEC_WID	NUMBER(38)	Foreign key to RESULT_SPEC_WID in W_EHA_RSLT_SPECIMEN table	Y
W_EHA_RSLT_SEQUENCING	RESULT_TYPE_WID	NUMBER(38)	Foreign key to RESULT_TYPE_WID in W_EHA_RSLT_TYPE table	Y
W_EHA_RSLT_SEQUENCING	RESULT_FILE_WID	NUMBER(38)	Foreign key to RESULT_FILE_WID in W_EHA_RSLT_FILE table	Y
W_EHA_RSLT_SEQUENCING	VARIANT_WID	NUMBER(38)	Foreign key to W_EHA_VARIANT table	Y
W_EHA_RSLT_SEQUENCING	CHROMOSOME_WID	NUMBER(38)	Foreign key to CHROMOSOME_WID in W_EHA_CHROMOSOME table	Y
W_EHA_RSLT_SEQUENCING	START_POSITION	NUMBER	Start position of variant, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_SEQUENCING	VARIANT_TYPE	VARCHAR2(100)	Type of variant including snp, insertion, or deletion	N
W_EHA_RSLT_SEQUENCING	ZYGOSITY	VARCHAR2(100)	Similarity of alleles in an organism to reference. If both same - hom etc.	N
W_EHA_RSLT_SEQUENCING	SCORE_VAF	NUMBER	Positive integer representing confidence in the call from CGI masterVar file	N
W_EHA_RSLT_SEQUENCING	SCORE_EAF	NUMBER	Positive or negative integer representing confidence in the call from CGI masterVar file	N
W_EHA_RSLT_SEQUENCING	ALLELE_READ_COUNT	NUMBER	Number of reads that support the given allele	N
W_EHA_RSLT_SEQUENCING	REFERENCE_READ_COUNT	NUMBER	Number of reads that support the reference sequence	N
W_EHA_RSLT_SEQUENCING_X	RESULT_STUDY_WID	NUMBER(38)	Foreign key to RESULT_STUDY_WID in W_EHA_RSLT_STUDY	Y
W_EHA_RSLT_SEQUENCING_X	ALLELE	VARCHAR2(1000)	Sequence of allele if less than or equal to 500 char	N
W_EHA_RSLT_SEQUENCING_X	ALLELE_CLOB	CLOB	Sequence of allele if greater than 500 char	N
W_EHA_RSLT_SEQUENCING_X	HAPLINK	VARCHAR2(100)	Integer ID that links the allele to alleles of other loci that are known to be on same haplotype	N
W_EHA_RSLT_SEQUENCING_X	FILTER	VARCHAR2(200)	Indicates PASS if position has passed all specified filters, otherwise lists FAIL, filters and results	N

Table 5–3 (Cont.) Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_RSLT_ SEQUENCING_X	GENOTYPE_ FILTER	VARCHAR2(200)	Indicates PASS if position has passed all genotype filters, otherwise lists FAIL, filters and results	N
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_SPEC_ WID	NUMBER(38)	Foreign key to RESULT_SPEC_ WID in W_EHA_RSLT_ SPECIMEN table	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_TYPE_ WID	NUMBER(38)	Foreign key to RESULT_TYPE_ WID in W_EHA_RSLT_TYPE table	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_FILE_WID	NUMBER(38)	Foreign key to RESULT_FILE_ WID in W_EHA_RSLT_FILE table	Y
W_EHA_RSLT_ COPY_NBR_VAR	CHROMOSOME_ WID	NUMBER(38)	Foreign key to CHROMOSOME_WID in W_ EHA_CHROMOSOME table	Y
W_EHA_RSLT_ COPY_NBR_VAR	START_POSITION	NUMBER	Start position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ COPY_NBR_VAR	END_POSITION	NUMBER	End position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ COPY_NBR_VAR	AVG_ NORMALIZED_ CVG	NUMBER	Baseline-normalized average coverage over the interval from START to END position	N
W_EHA_RSLT_ COPY_NBR_VAR	GC_CORRECTED_ CVG	NUMBER	GC-corrected average coverage of a window width based on START and END positions	N
W_EHA_RSLT_ COPY_NBR_VAR	FRACTION_ UNIQUE	NUMBER	Fraction of coverage due to unique mappings	N
W_EHA_RSLT_ COPY_NBR_VAR	RELATIVE_CVG	NUMBER	AVG_NORMALIZED_CVG divided by estimate of diploid median average adjusted coverage, can be null if source is N	N
W_EHA_RSLT_ COPY_NBR_VAR	CALLED_PLOIDY	NUMBER	Called ploidy for the segment, usually an integer from 0 to maximum ploidy value, if N in source file, leave blank	N
W_EHA_RSLT_ COPY_NBR_VAR	CALLED_CNV_ TYPE	VARCHAR2(100)	Classification of CALLED_ PLOIDY into one of six categories	N
W_EHA_RSLT_ COPY_NBR_VAR	PLOIDY_SCORE	NUMBER	Phred-like confidence that the segment has the CALLED_ PLOIDY correct	N
W_EHA_RSLT_ COPY_NBR_VAR	CNV_TYPE_SCORE	NUMBER	Phred-like confidence that CALLED_CNV_TYPE is correct	N
W_EHA_RSLT_ CNV_X	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y

Table 5–3 (Cont.) Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_RSLT_CNV_X	REPEATS	VARCHAR2(2000)	Percent of called CNV segments that overlaps with each category of genomic repeats. Format is: Repeat category:XX. Stored if number of characters less than or equal to 2000.	N
W_EHA_RSLT_CNV_X	REPEATS_CLOB	CLOB	Percent of called CNV segments that overlaps with each category of genomic repeats. Format is: Repeat category:XX. Stored if number of characters greater than 2000.	N
W_EHA_RSLT_NOCALL	RESULT_STUDY_WID	NUMBER(38)	Foreign key to RESULT_STUDY_WID in W_EHA_RSLT_STUDY	Y
W_EHA_RSLT_NOCALL	RESULT_SPEC_WID	NUMBER(38)	Foreign key to RESULT_SPEC_WID in W_EHA_RSLT_SPECIMEN table	Y
W_EHA_RSLT_NOCALL	RESULT_TYPE_WID	NUMBER(38)	Foreign key to RESULT_TYPE_WID in W_EHA_RSLT_TYPE table	Y
W_EHA_RSLT_NOCALL	RESULT_FILE_WID	NUMBER(38)	Foreign key to RESULT_FILE_WID in W_EHA_RSLT_FILE table	Y
W_EHA_RSLT_NOCALL	CHROMOSOME_WID	NUMBER(38)	Foreign key to CHROMOSOME_WID in W_EHA_CHROMOSOME table	Y
W_EHA_RSLT_NOCALL	START_POSITION	NUMBER	Start position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_NOCALL	END_POSITION	NUMBER	End position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_NOCALL	NOCALL_TYPE	VARCHAR2(100)	Type of no-call	N
W_EHA_RSLT_NOCALL_X	RESULT_STUDY_WID	NUMBER(38)	Foreign key to RESULT_STUDY_WID in W_EHA_RSLT_STUDY	Y
W_EHA_RSLT_NOCALL_X	ALLELE	varchar2(500)	Sequence of allele if less than or equal to 500 char	N
W_EHA_RSLT_NOCALL_X	ALLELE_CLOB	CLOB	Sequence of allele if greater than 500 char	N
W_EHA_RSLT_NOCALL_X	HAPLINK	VARCHAR2(100)	Integer ID that links the allele to alleles of other loci that are known to be on same haplotype	N
W_EHA_PROBE_ALT_LINK	HUGO_NAME	VARCHAR2(200)	Hugo name for a gene corresponding to probe	Y
W_EHA_PROBE_ALT_LINK	PROBE_WID	NUMBER(38)	Foreign key to W_EHA_PROBE	Y
W_EHA_RSLT_STUDY	RESULT_STUDY_NAME	VARCHAR2(200)	Name of the study under which results will be loaded	Y
W_EHA_RSLT_STUDY	RESULT_STUDY_DESC	VARCHAR2(2000)	Description of the study under which results will be loaded	N
W_EHA_RSLT_SPECIMEN	SPECIMEN_WID	NUMBER(38)	Foreign key to Specimen table in CDM1.1	Y

Table 5–3 (Cont.) Result Entity Descriptions

Table	Column	Data Type	Description	Required (Not Null)
W_EHA_RSLT_ SPECIMEN	SPEC_DATASRC_ WID	NUMBER(38)	Identifier for specimen datasource	Y
W_EHA_RSLT_ SPECIMEN	SPECIMEN_ NUMBER	VARCHAR2(80)	Specimen identifier in linked specimen datasource database	Y
W_EHA_RSLT_ SPECIMEN	SPECIMEN_ VENDOR_NUMBER	VARCHAR2(80)	Vendor identifier in linked specimen datasource database	Y
W_EHA_ CHROMOSOME	CHROMOSOME	VARCHAR2(50)	Chromosome number or identifier	Y

5.4 Table Descriptions

The following table contains descriptions of all tables in ODB:

Table 5–4 Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_HUGO_ INFO	Reference	Stores the Hugo Names for all genes with cross reference information. This table matches the data from the Human Gene Nomenclature Committee.	Master populate script	N
W_EHA_DNA_ SOURCE	Reference	Stores the DNA sequence source buffer where multiple genes and other features may reference. The identifciation and locus of the sequence source is given with the accession field.	Reference loader in Java	N
W_EHA_PROT_INFO	Reference	Stores PROTEIN information for each named version of a protein molecule.	Reference loader in Java	N
W_EHA_PROTEIN	Reference	Stores PROTEIN molecule information for amino acid sequences transcribed from genes.	Reference loader in Java	N
W_EHA_GENE	Reference	Stores the parent gene records as loaded from the reference. The gene table will store the hugo name as well as the the name used in the reference data. Othere tables will have foreign keys to this table such as GENE_SEGMENT and GENE_ STRUCTURE.	Reference loader in Java	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_GENE_ COMP_SEGMENT	Reference	Maps each GENE_ COMPONENT to a specific region in the DNA_SOURCE buffer. Each GENE_ COMPONENT can have many segments that are not contiguous.	Reference loader in Java	N
W_EHA_GENE_ COMPONENT	Reference	Stores each component of the gene (i.e. mRNA, coding segments, etc...).	Reference loader in Java	N
W_EHA_GENE_ SEGMENT	Reference	Maps each GENE to the DNA_SOURCE buffers used as reference. Note that a GENE definition may span multiple source references.	Reference loader in Java	N
W_EHA_GENE_ STRUCTURE	Reference	Stores each transcript of the GENE used to create different PROTEIN molecules. Each GENE_ STRUCTURE is indicated with a QUALIFIER indicating the transcript_id. All of the GENE_ COMPONENT records with the same transcript_id will be grouped with the same pareint GENE_ STRUCTURE record.	Reference loader in Java	N
W_EHA_PROT_ COMPONENT	Reference	Stores the components that are imported from SwissProt files for all Protein definitions.	Reference loader in Java	N
W_EHA_SPECIES	Reference	Stores SPECIES information for each genome stored in database.	Reference loader in Java	N
W_EHA_VARIANT	Reference	Stores novel and known variants for the genomes loaded. The actual variation is stored in the REPLACE_TAG field.	Reference loader in Java, Sequencing loaders: VCF, MAF, CGI	N
W_EHA_PATHWAY	Reference	Stores PATHWAY names for protein network.	Reference loader in Java	N
W_EHA_PATHWAY_ PROTEIN	Reference	Stores genes or proteins that belong to a given pathway based on import from GSEA files.	Reference loader in Java	N
W_EHA_SOURCE_ LIT_REF	Reference	Stores literature references for each DNA_SOURCE definition.	Reference loader in Java	N
W_EHA_PROT_LIT_ REF	Reference	Stores literature references for each PROTEIN definition.	Reference loader in Java	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_VARIANT_QLFR	Reference	Stores qualifier attributes for each VARIANT	Reference loader in Java	N
W_EHA_SOURCE_QLFR	Reference	Stores all of the qualifier attributes for each DNA_SOURCE definition.	Reference loader in Java	N
W_EHA_GENE_QLFR	Reference	Stores all of the qualifier attributes for each gene.	Reference loader in Java	N
W_EHA_GENE_COMP_QLFR	Reference	Stores all of the qualifier attributes for each component of the gene (i.e. mRNA, coding segments, etc...).	Reference loader in Java	N
W_EHA_PROT_COMP_QLFR	Reference	Stores all of the qualifier attributes for each component of the protein.	Reference loader in Java	N
W_EHA_VARIANT_XREF	Reference	Cross reference information for VARIANT records	Reference loader in Java	N
W_EHA_SOURCE_XREF	Reference	Stores cross reference information for DNA_SOURCE buffers.	Reference loader in Java	N
W_EHA_PROT_COMP_XREF	Reference	Stores the PROTEIN COMPONENT cross reference information linked to other databases.	Reference loader in Java	N
W_EHA_GENE_XREF	Reference	Stores the GENE cross reference information linked to other databases.	Reference loader in Java	N
W_EHA_GENE_COMPONENT_XREF	Reference	Stores GENE_COMPONENT cross reference links to other databases.	Reference loader in Java	N
W_EHA_PROT_XREF	Reference	Stores PROTEIN cross reference information	Reference loader in Java	N
W_EHA_PROBE	Result/Reference	Stores PROBE information and link to corresponding gene.	Result loader for probes (expression)	N
W_EHA_PROBE_XREF	Reference	Stores cross reference information for PROBE records.	Currently not populated	N
W_EHA_RSLT_GENE_EXP	Result	Stores gene expression intensity value along with p-value, call if available.	Result loader for gene expression	N
W_EHA_DATASOURCE	Result	Stores specimen datasource information and link, if needed.	Pre-seeded	N
W_EHA_RSLT_TYPE	Result	Stores identifiers for types of result, pre-seeded.	Pre-seeded	N
W_EHA_RSLT_FILE	Result	Stores links and other information about the result file.	Sequence loaders: CGI, MAF, VCF, gene expression loader	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_RSLT_FILE_TYPE	Result	Stores type of file information, including version, vendor etc, pre-seeded.	Pre-seeded	N
W_EHA_RSLT_SEQUENCING	Result	Stores sequencing results, namely substitutions, insertions, deletions coming from CGI, MAF, VCF file formats.	Sequence loaders: CGI, MAF, VCF	N
W_EHA_RSLT_COPY_NBR_VAR	Result	Stores copy number variation results from sequencing.	Currently not populated	N
W_EHA_RSLT_NOCALL	Result	Stores no-call results from CGI sequencing.	CGI sequence loader	N
W_EHA_PROBE_ALT_LINK	Result	Used when there is more than one gene that is matched to a given probe.	Currently not populated	N
W_EHA_SEQUENCING_X	Result	Stores varchar data of sequencing results coming from CGI, MAF, VCF file formats.	Sequence loaders: CGI, MAF, VCF	Y
W_EHA_RSLT_COPY_NBR_VAR_X	Result	Stores varchar data for copy number variation results from sequencing.	Currently not populated	Y
W_EHA_RSLT_NOCALL_X	Result	Stores varchar data of no-call results from CGI sequencing.	CGI sequence loader	Y
W_EHA_RSLT_STUDY	Result	Stores study information under which results pertaining to sequencing and expression are loaded.	User defined data	Y
W_EHA_RSLT_SPECIMEN	Result	Stores the Specimen details and links datasource table. All results would have specimen_wid in their tables.	Sequence loaders: CGI, MAF, VCF, gene expression loader	Y
W_EHA_CHROMOSOME	Result	Stores all possible chromosome numbers for which result sequencing data is imported. CHROMOSOME_WID is used to link to result sequencing tables.	Pre-seeded	Y
W_EHA_RSLT_SEQUENCING	REFERENCE_READ_COUNT	NUMBER	Number of reads that support the reference sequence	N
W_EHA_RSLT_SEQUENCING_X	RESULT_STUDY_WID	NUMBER(38)	Foreign key to RESULT_STUDY_WID in W_EHA_RSLT_STUDY	Y
W_EHA_RSLT_SEQUENCING_X	ALLELE	VARCHAR2(1000)	Sequence of allele if less than or equal to 500 char	N
W_EHA_RSLT_SEQUENCING_X	ALLELE_CLOB	CLOB	Sequence of allele if greater than 500 char	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_RSLT_ SEQUENCING_X	HAPLINK	VARCHAR2(100)	Integer ID that links the allele to alleles of other loci that are known to be on same haplotype	N
W_EHA_RSLT_ SEQUENCING_X	FILTER	VARCHAR2(200)	Indicates PASS if position has passed all specified filters, otherwise lists FAIL, filters and results	N
W_EHA_RSLT_ SEQUENCING_X	GENOTYPE_FILTER	VARCHAR2(200)	Indicates PASS if position has passed all genotype filters, otherwise lists FAIL, filters and results	N
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_SPEC_WID	NUMBER(38)	Foreign key to RESULT_ SPEC_WID in W_EHA_ RSLT_SPECIMEN table	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_TYPE_WID	NUMBER(38)	Foreign key to RESULT_ TYPE_WID in W_EHA_ RSLT_TYPE table	Y
W_EHA_RSLT_ COPY_NBR_VAR	RESULT_FILE_WID	NUMBER(38)	Foreign key to RESULT_ FILE_WID in W_EHA_ RSLT_FILE table	Y
W_EHA_RSLT_ COPY_NBR_VAR	CHROMOSOME_ WID	NUMBER(38)	Foreign key to CHROMOSOME_WID in W_EHA_CHROMOSOME table	Y
W_EHA_RSLT_ COPY_NBR_VAR	START_POSITION	NUMBER	Start position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ COPY_NBR_VAR	END_POSITION	NUMBER	End position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ COPY_NBR_VAR	AVG_ NORMALIZED_ CVG	NUMBER	Baseline-normalized average coverage over the interval from START to END position	N
W_EHA_RSLT_ COPY_NBR_VAR	GC_CORRECTED_ CVG	NUMBER	GC-corrected average coverage of a window width based on START and END positions	N
W_EHA_RSLT_ COPY_NBR_VAR	FRACTION_ UNIQUE	NUMBER	Fraction of coverage due to unique mappings	N
W_EHA_RSLT_ COPY_NBR_VAR	RELATIVE_CVG	NUMBER	AVG_NORMALIZED_CVG divided by estimate of diploid median average adjusted coverage, can be null if source is N	N
W_EHA_RSLT_ COPY_NBR_VAR	CALLED_PLOIDY	NUMBER	Called ploidy for the segment, usually an integer from 0 to maximum ploidy value, if N in source file, leave blank	N
W_EHA_RSLT_ COPY_NBR_VAR	CALLED_CNV_ TYPE	VARCHAR2(100)	Classification of CALLED_ PLOIDY into one of six categories	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_RSLT_ COPY_NBR_VAR	PLOIDY_SCORE	NUMBER	Phred-like confidence that the segment has the CALLED_PLOIDY correct	N
W_EHA_RSLT_ COPY_NBR_VAR	CNV_TYPE_SCORE	NUMBER	Phred-like confidence that CALLED_CNV_TYPE is correct	N
W_EHA_RSLT_CNV_ X	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y
W_EHA_RSLT_CNV_ X	REPEATS	VARCHAR2(2000)	Percent of called CNV segments that overlaps with each category of genomic repeats. Format is: Repeat category:XX. Stored if number of characters less than or equal to 2000.	N
W_EHA_RSLT_CNV_ X	REPEATS_CLOB	CLOB	Percent of called CNV segments that overlaps with each category of genomic repeats. Format is: Repeat category:XX. Stored if number of characters greater than 2000.	N
W_EHA_RSLT_ NOCALL	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y
W_EHA_RSLT_ NOCALL	RESULT_SPEC_WID	NUMBER(38)	Foreign key to RESULT_ SPEC_WID in W_EHA_ RSLT_SPECIMEN table	Y
W_EHA_RSLT_ NOCALL	RESULT_TYPE_WID	NUMBER(38)	Foreign key to RESULT_ TYPE_WID in W_EHA_ RSLT_TYPE table	Y
W_EHA_RSLT_ NOCALL	RESULT_FILE_WID	NUMBER(38)	Foreign key to RESULT_ FILE_WID in W_EHA_ RSLT_FILE table	Y
W_EHA_RSLT_ NOCALL	CHROMOSOME_ WID	NUMBER(38)	Foreign key to CHROMOSOME_WID in W_EHA_CHROMOSOME table	Y
W_EHA_RSLT_ NOCALL	START_POSITION	NUMBER	Start position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ NOCALL	END_POSITION	NUMBER	End position of DNA sequence, with the 1st base in chromosome having position 1.	N
W_EHA_RSLT_ NOCALL	NOCALL_TYPE	VARCHAR2(100)	Type of no-call	N
W_EHA_RSLT_ NOCALL_X	RESULT_STUDY_ WID	NUMBER(38)	Foreign key to RESULT_ STUDY_WID in W_EHA_ RSLT_STUDY	Y
W_EHA_RSLT_ NOCALL_X	ALLELE	varchar2(500)	Sequence of allele if less than or equal to 500 char	N
W_EHA_RSLT_ NOCALL_X	ALLELE_CLOB	CLOB	Sequence of allele if greater than 500 char	N

Table 5–4 (Cont.) Table Descriptions

Table	Type	Description	Loaders Populate	Included in the Release 1.0.1
W_EHA_RSLT_ NOCALL_X	HAPLINK	VARCHAR2(100)	Integer ID that links the allele to alleles of other loci that are known to be on same haplotype	N
W_EHA_PROBE_ ALT_LINK	HUGO_NAME	VARCHAR2(200)	Hugo name for a gene corresponding to probe	Y
W_EHA_PROBE_ ALT_LINK	PROBE_WID	NUMBER(38)	Foreign key to W_EHA_ PROBE	Y
W_EHA_RSLT_ STUDY	RESULT_STUDY_ NAME	VARCHAR2(200)	Name of the study under which results will be loaded	Y
W_EHA_RSLT_ STUDY	RESULT_STUDY_ DESC	VARCHAR2(2000)	Description of the study under which results will be loaded	N
W_EHA_RSLT_ SPECIMEN	SPECIMEN_WID	NUMBER(38)	Foreign key to Specimen table in CDM1.1	Y
W_EHA_RSLT_ SPECIMEN	SPEC_DATASRC_ WID	NUMBER(38)	Identifier for specimen datasource	Y
W_EHA_RSLT_ SPECIMEN	SPECIMEN_ NUMBER	VARCHAR2(80)	Specimen identifier in linked specimen datasource database	Y
W_EHA_RSLT_ SPECIMEN	SPECIMEN_ VENDOR_NUMBER	VARCHAR2(80)	Vendor identifier in linked specimen datasource database	Y
W_EHA_ CHROMOSOME	CHROMOSOME	VARCHAR2(50)	Chromosome number or identifier	Y

6 Use Case Examples

This chapter lists use case for Oracle Health Sciences Omics Data Bank. It contains the following topics:

6.1 Use Cases

This section contains the following use case scenarios:

Note: To run some of the use case queries you need to create global temporary tables. For more information, refer to Section 6.2, "Global Temporary Table Creation Code Snippets,"

- Scenario 1 - Find patients that are poor responders for drug A and have a mutation in the promoter region of gene A.
- Scenario 2 - Expression level of TP53 mutant by cancer tissue.
- Scenario 3 - Show me how many patients with Moderate Alzheimer's, who have a smoking habit were treated with Doxorubicin? Which is there most recent Cholesterol and Hb counts? Of those patients, which have samples and what is the differential expression of genes in those samples compared to my controls?
- Scenario 4 - MAGEA3 differential expression between basal cell carcinoma and metastatic carcinoma samples.
- Scenario 5 - Compare CFL2 gene expresion between two cohorts. Cohort1: subjects with primary melanoma or basal cell carcinoma or squamous cell carcinoma; Cohort2: subjects with metastatic melanoma.
- Scenario 6 - Ability to query subjects for established molecular tests, for example the presence of known myeloma mutations such as the t(4;14) translocation or mutations in oncogenes such as RAS.
- Scenario 7 - Ability to research a gene in the sample set.
- Scenario 8 - Dr. Smith is evaluating gene expression data set for a group of tumors in his research project. He views pattern of gene expression data.
- Scenario 9 - Dr. Smith started a trial for patients with metastatic melanoma. His hypothesis is that certain mutations make a patient a better candidate for a new treatment protocol compared to patients without the mutation. Dr. Storm searches the dat awarehouse using the HRI solution tools and identifies a group of patients

clinically eligible for his study when identifies a subgroup of patients with certain gene mutaitons who also fit the clinical eligibility criteria for study. He contacts this group through the TCC protocol coordinator and commences his study.

- Scenario 10 - Dr. Smith is attempting to identify clusters of genes that are activated with maglinant melanoma (skin cancer), in order to find successful interventions for prevention or treatment. Dr. Merdock is utilizing a Hidden Markov model to identify temporal patterns in the gene expression. He will visualize these patterns with a hierarchical clustering chart in order to more easily identify contributing factors for the illness. Dr. Merdock processes a large number of sequence results and uses the BI tools to identify data he wants to pull into his analysis. He runs statistical and advanced visualization of this data, archives the data utilized in the analysis and is able to find patterns that would be extremely difficult to find without these tools. These analyses would assist in determining patient prognosis, best therapies and potentially new druagable targets, especially in patients who are non-responsive to therapy.
- Scenario 11 - CohortID: the researcher needs to be able to select a set of chips based on the certain criteria. The researcher will then devide the cohort into different groups (this can also be done at the cohort level where the researchers select several different cohorts and saves them) based on some criteria. The researcher will then see what genes (probes) are most differently regulated between these classes (cohorts). The research should also be able to select a sepcific gene and see its expression values across the samples. The results should also be shown in scatter plots or bar graphs.
- Scenario 12 - The researcher should be able to select a patient cohort based on the expression level for a set of genes. The same selection as the example above is needed but with the addition filters for gene express levels.
- Scenario 13 - I have a set of mutations meeting a set of criteria (for example, frequency, and so on), I need to annotated them with gene and function information.
- Scenario 14 - Select mutation with deep functional annotation (for eample, polyphen, sift).
- Scenario 15 - I have a pathway. What mutations are present in the pathway and which study were they identified in (for example, what tumor types)?
- Scenario 16 - I have a mutation I have deemed interesting, I want to know if it perturbs gene expression (in the same project or in other projects).
- Scenario 17 - Examine gene expression across cell lines within each panel.
- Scenario 18 - Examine copy number across cell lines within each panel.
- Scenario 19 - Do KRAS mutant cell lines express more IL-6 than wildtype.
- Scenario 20 - Do RAS high cell lines expression more IL-6 than low RAS lines.
- Scenario 21 - Are cMET amplified cell line more sensitive to METi?
- Scenario 22 - Show HER+ lines have ERBB2 copy number gains.
- Scenario 23 - Label cell lines for PTEN, PIK3CA mutations.
- Scenario 24 - What is the frequency of co-mutation of two genes in a data set?
- Scenario 25 - The steps for Initial data processing to identify sequence variants should be fully automated by writing scripts to interact with public database. The variants could then be presented in a graphic interface that summarized their characteristics.

- Scenario 26 - Map the variant coordinates to gene, exon, amino acid, chromosomal location, region of protein; and obtain CCDS (Consensus Coding Sequence) and RefSeq IDs. Example: PDEGFR D842V, exon 18 out of 23 total exons, 4q11-q13, protein tyrosine kinase domain; CCDS3495.1, NP_006197.
- Scenario 27 - Determine whether the variant is a known SNP or a previously characterized somatic mutation in cancer. Also determine the total number of reported somatic variants in the entire gene, and whether any are near the variant in question.
- Scenario 28 - Show me all patients whose cancer cells had a deletion in gene X.

6.1.1 Scenario 1

Use Case - Find patients that are poor responders for drug A and have a mutation in the promoter region of gene A.

Areas

- Variant
- Gene Annotation
- Test and Results
- Drug Info

Output queries tables from - ODB+CDM

Query -

```
insert into query_1

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ vrt.RESULT_
SPEC_WID, vi.start_position, vi.end_position, vi.replace_tag

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.ROW_WID, v.start_position, v.end_position, v.replace_
tag

  FROM W_EHA_VARIANT v, (

    SELECT gcsg.SOURCE_WID, gcsg.START_POSITION, gcsg.END_
POSITION

    FROM W_EHA_GENE g, w_eha_gene_structure gst, w_eha_gene_
component gc,  w_eha_gene_comp_segment gcsg

    WHERE g.HUGO_NAME IN ('BRCA2')

    AND gc.COMPONENT_TYPE IN ('CDS')

    AND gc.STRUCTURE_WID = gst.ROW_WID

    AND gst.GENE_WID = g.ROW_WID

    AND gcsg.GENE_COMPONENT_WID = gc.ROW_WID

  )sg

  WHERE v.SOURCE_WID = sg.SOURCE_WID

    AND v.START_POSITION <= sg.END_POSITION

    AND sg.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study
```

```
WHERE vrt.VARIANT_WID = vi.ROW_WID
```

AND w_eha_rslt_study.result_study_name = 'study_3'

```
AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

AND vrt.RESULT_SPEC_WID = 250
```

6.1.2 Scenario 2

Use Case - Expression level of TP53 mutant by cancer tissue.

Areas

- Variant
- Gene Annotation
- Gene Expression
- Biospecimen Data

Output queries tables from - ODB+CDM

Query 2 -

```
insert into query_2

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp,query_2

where p.primary_hugo_name in ('TP53'))

AND r_exp.probe_wid = p.row_wid

AND r_exp.specimen_wid = query_2_a.specimen_wid ;
```

Query 2a -

```
insert into query_2_1

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ distinct
vrt.RESULT_SPEC_WID, vi.start_position, vi.end_position,
vi.replace_Tag

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN  ('TP53')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study
```

```
WHERE w_eha_rslt_study.result_study_name = 'study_1'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

AND vrt.VARIANT_WID = vi.ROW_WID;
```

6.1.3 Scenario 3

Use Case - Show me how many patients with Moderate Alzheimer's, who have a smoking habit were treated with Doxorubicin? Which is there most recent Cholesterol and Hb counts? Of those patients, which have samples and what is the differential expression of genes in those samples compared to my controls?

Areas

- Diagnosis
- Biospecimen Data
- Drug Info
- Risk Factor
- Test and Results
- Expression
- Gene Annotation

Output queries tables from - ODB+CDM

Query -

```
insert into query_7

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('COX15')))

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_10'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 950

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('CHAT')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_10'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 950

union ALL
```

```
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('WNT8B')
   AND r_exp.probe_wid = p.row_wid
   AND w_eha_rslt_study.result_study_name = 'study_10'
   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
   AND r_exp.result_spec_wid = 950
union ALL
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('TRBV1')
   AND r_exp.probe_wid = p.row_wid
   AND w_eha_rslt_study.result_study_name = 'study_10'
   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
   AND r_exp.result_spec_wid = 950
```

6.1.4 Scenario 4

Use Case - MAGEA3 differential expression between basal cell carcinoma and metastatic carcinoma samples.

Areas

- Gene Annotation
- Expression
- Biospecimen Data

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
insert into query_9
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('MAGEA3')
   AND r_exp.probe_wid = p.row_wid
   AND w_eha_rslt_study.result_study_name = 'study_10'
   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
   AND r_exp.result_spec_wid = 950
```

6.1.5 Scenario 5

Use Case - Compare CFL2 gene expresion between two cohorts.

Cohort1: subjects with primary melanoma or basal cell carcinoma or squamous cell carcinoma;

Cohort2: subjects with metastatic melanoma

Areas

- Expression
- Biospecimen Data
- Cancer Diagnosis

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
insert into query_10

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('ZNF211')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_10'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 950
```

6.1.6 Scenario 6

Use Case - Ability to query subjects for established molecular tests, for example the presence of known myeloma mutations such as the t(4;14) translocation or mutations in oncogenes such as RAS.

Areas

- Variant
- Chromosomal Rearrangement
- Gene Annotation

Output queries tables from - ODB+CDM

Query -

```
create global temporary table query_11(result_spec_wid NUMBER,
start_position NUMBER, end_position NUMBER, replace_tag
varchar2(1000)) ON COMMIT DELETE ROWS;


INSERT INTO Query_11

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ vrt.RESULT_
SPEC_WID, vi.start_position,

vi.end_position, vi.replace_tag

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (
    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID
    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g
    WHERE g.HUGO_NAME IN  ('KRAS')
    AND gsg.GENE_WID = g.ROW_WID
  )gs
  WHERE v.SOURCE_WID = gs.SOURCE_WID
    AND v.START_POSITION <= gs.END_POSITION
    AND gs.START_POSITION <= v.END_POSITION
) vi,w_eha_rslt_study
WHERE w_eha_rslt_study.result_study_name = 'study_1'
AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND vrt.VARIANT_WID = vi.ROW_WID
AND vrt.RESULT_SPEC_WID = 50;
```

6.1.7 Scenario 7

Use Case - Ability to research a gene in the sample set.

Areas

- Gene Annotation

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ count(distinct
w_eha_rslt_study.result_study_name)
FROM W_EHA_RSLT_SEQUENCING vrt, (
  SELECT v.*, gs.hugo_name
  FROM W_EHA_VARIANT v, (
    SELECT g.hugo_name, gsg.START_POSITION, gsg.END_POSITION,
gsg.SOURCE_WID
    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g
    WHERE g.HUGO_NAME IN ('BRCA2')
    AND gsg.GENE_WID = g.ROW_WID
  )gs
  WHERE v.SOURCE_WID = gs.SOURCE_WID
    AND v.START_POSITION <= gs.END_POSITION
    AND gs.START_POSITION <= v.END_POSITION
) vi,w_eha_rslt_study
WHERE vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND vrt.VARIANT_WID = vi.ROW_WID;
union all
```

```
/* 6 sec */

select distinct w_eha_rslt_study2.result_study_name

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
w_eha_rslt_study2

where p.primary_hugo_name in ('BRCA2')

AND r_exp.probe_wid = p.row_wid

AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study2.ROW_WID

union all

/* 1082sec */

SELECT  /*+ index (r_cnv W_EHA_RSLT_COPY_NBR_VAR_M1),index (r_
cnv W_EHA_RSLT_COPY_NBR_VAR_M2)*/

distinct w_eha_rslt_study3.result_study_name

FROM W_EHA_RSLT_COPY_NBR_VAR r_cnv, w_eha_rslt_study w_eha_rslt_
study3,

(SELECT gsg.START_POSITION gsg_START_POSITION, gsg.END_POSITION
gsg_END_POSITION,

  s.START_POSITION s_START_POSITION, s.END_POSITION s_END_
POSITION

  FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g, W_EHA_DNA_SOURCE s

  WHERE g.HUGO_NAME IN ('BRCA2')

  AND gsg.GENE_WID = g.ROW_WID

  AND gsg.SOURCE_WID= s.ROW_WID

) sg

WHERE r_cnv.START_POSITION <= (sg.gsg_end_position+sg.s_start_
position)

AND (sg.gsg_start_POSITION+sg.s_start_position) <= r_cnv.end_
position

AND r_cnv.RESULT_STUDY_WID = w_eha_rslt_study3.ROW_WID
```

6.1.8 Scenario 8

Use Case - Dr. Smith is evaluating gene expression data set for a group of tumors in his research project. He views pattern of gene expression data.

Areas

- Expression
- Biospecimen Data

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
insert into query_13

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
```

```
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('TP53')
AND r_exp.probe_wid = p.row_wid
AND w_eha_rslt_study.result_study_name = 'study_11'
AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND r_exp.result_spec_wid = 1050
union ALL
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('BRCA1')
AND r_exp.probe_wid = p.row_wid
AND w_eha_rslt_study.result_study_name = 'study_11'
AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND r_exp.result_spec_wid = 1050
union ALL
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('BRCA2')
AND r_exp.probe_wid = p.row_wid
AND w_eha_rslt_study.result_study_name = 'study_11'
AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND r_exp.result_spec_wid = 1050
union ALL
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity
from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study
where p.primary_hugo_name in ('BYSL')
AND r_exp.probe_wid = p.row_wid
AND w_eha_rslt_study.result_study_name = 'study_11'
AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND r_exp.result_spec_wid = 1050
```

6.1.9 Scenario 9

Use Case - Dr. Smith started a trial for patients with metastatic melanoma. His hypothesis is that certain mutations make a patient a better candidate for a new treatment protocol compared to patients without the mutation. Dr. Storm searches the dat awarehouse using the HRI solution tools and identifies a group of patients clinically eligible for his study when identifies a subgroup of patients with certain gene mutaitons who also fit the clinical eligibility criteria for study. He contacts this group through the TCC protocol coordinator and commences his study.

Areas

- Cancer Diagnosis
- TCC Consent
- Variant
- Gene Annotation
- Treatment Protocol

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
create global temporary table query_14(result_spec_wid NUMBER,
start_position NUMBER, end_position NUMBER, replace_tag
varchar2(1000)) ON COMMIT DELETE ROWS;

/* 32.1 sec */

INSERT INTO Query_14

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ vrt.RESULT_
SPEC_WID, vi.start_position,

vi.end_position, vi.replace_tag

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN  ('CHAT')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study

WHERE w_eha_rslt_study.result_study_name = 'study_4'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

AND vrt.VARIANT_WID = vi.ROW_WID

AND vrt.RESULT_SPEC_WID = 350;
```

6.1.10 Scenario 10

Use Case - Dr. Smith is attempting to identify clusters of genes that are activated with maglinant melanoma (skin cancer), in order to find successful interventions for prevention or treatment. Dr. Merdock is utilizing a Hidden Markov model to identify temporal patterns in the gene expression. He will visualize these patterns with a hierarchical clustering chart in order to more easily identify contributing factors for the illness. Dr. Merdock processes a large number of sequence results and uses the BI tools to identify data he wants to pull into his analysis. He runs statistical and advanced visualization of this data, archives the data utilized in the analysis and is able to find patterns that would be extremely difficult to find without these tools. These analyses would assist in determining patient prognosis, best therapies and potentially new druagable targets, especially in patients who are non-responsive to therapy.

Areas

- Cancer Diagnosis
- Test and Results

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
insert into query_15

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('TP53')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_14'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1350

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('BRCA1')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_14'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1350

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('GPR4')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_14'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1350

union ALL
```

```
select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('PABPC1')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_14'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1350

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('SOBP')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_14'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1350
```

6.1.11 Scenario 11

Use Case - CohortID: the researcher should be able to select a set of chips based on the following (and potential other) criteria's:

1. demographic (age, sex, and so on);
2. clinical data;
3. tumor type;
4. tumor properties;
5. chip quality.

The researcher will then divide the cohort into different groups (this can also be done at the cohort level where the researchers select several different cohorts and saves them) based on some criteria. The researcher will then see what genes (probes) are most differently regulated between these classes (cohorts). The research should also be able to select a sepecific gene and see its expression values across the samples. The results should also be shown in scatter plots or bar graphs.

Areas

- Cancer Diagnosis
- Demographic
- Biospecimen Data
- QC Data
- Gene Annotation
- Expression

Output queries tables from - ODB+CDM

Query -

```
insert into query_16

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('LEPR')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_20'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1950

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('MSSE')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_20'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1950

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('PCCA')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_20'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1950

union ALL

select r_exp.result_spec_wid, p.primary_hugo_name, r_
exp.intensity

from W_EHA_PROBE p, W_EHA_RSLT_GENE_EXP r_exp, w_eha_rslt_study

where p.primary_hugo_name in ('SLC1A1')

   AND r_exp.probe_wid = p.row_wid

   AND w_eha_rslt_study.result_study_name = 'study_20'

   AND r_exp.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

   AND r_exp.result_spec_wid = 1950
```

6.1.12 Scenario 12

Use Case - The researcher should be able to select a patient cohort based on the expression level for a set of genes. The same selection as the example above is needed but with the addition filters for gene express levels.

Areas

- Cancer Diagnosis
- Demographic
- Biospecimen Data
- QC Data
- Gene Annotation
- Expression

Output queries tables from - ODB+CDM

Query -

```
select count(distinct r_exp2.result_spec_wid)

from W_EHA_PROBE p2, W_EHA_RSLT_GENE_EXP r_exp2, w_eha_rslt_
study w_eha_rslt_study2,

(select avg(r_exp1.intensity) exp_avg

from W_EHA_PROBE p1, W_EHA_RSLT_GENE_EXP r_exp1, w_eha_rslt_
study w_eha_rslt_study1

where p1.primary_hugo_name in
('TP53','BRCA1','GPR4','PABPC1','SOBP')

   AND r_exp1.probe_wid = p1.row_wid

   AND w_eha_rslt_study1.result_study_name = 'study_15'

   AND r_exp1.RESULT_STUDY_WID = w_eha_rslt_study1.ROW_WID)
intensity

where p2.primary_hugo_name in
('TP53','BRCA1','GPR4','PABPC1','SOBP')

   AND r_exp2.probe_wid = p2.row_wid

   AND r_exp2.intensity > intensity.exp_avg

   AND w_eha_rslt_study2.result_study_name = 'study_15'

   AND r_exp2.RESULT_STUDY_WID = w_eha_rslt_study2.ROW_WID
```

6.1.13 Scenario 13

Use Case - I have a set of mutations meeting a set of criteria (for example, frequency, and so on), I need to annotate them with gene and function information.

Areas

- Variant
- Variant Population Info
- Demographic
- Gene Annotation

- Functional Annotation

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
select DISTINCT hu.*, gi.reference_id from W_EHA_HUGO_INFO hu, (

   select DISTINCT g.hugo_name, vref.reference_id from w_eha_
gene g, w_eha_variant v, w_eha_variant_xref vref, W_EHA_DNA_
SOURCE s, W_EHA_GENE_SEGMENT gs, W_EHA_SPECIES sp

   where

      vref.reference_id = 'rs11169'

      AND vref.variant_wid = v.row_wid

      AND gs.SOURCE_WID= v.SOURCE_WID

      AND s.row_wid = gs.source_wid

      AND v.END_POSITION >= gs.START_POSITION

      AND gs.END_POSITION >= v.START_POSITION

      AND gs.GENE_WID=g.ROW_WID

      AND  sp.common_name = 'human'

      AND s.SPECIES_WID=sp.ROW_WID

      )gi

   WHERE

      hu.APPROVED_SYMBOL = gi.HUGO_NAME;
```

6.1.14 Scenario 14

Use Case - Select mutation with deep functional annotation (for example, polyphen, sift)

Areas

- Variant
- Gene Annotation

Output queries tables from - ODB: REFERENCE

Query -

```
SELECT vg.REFERENCE_ID, vg.absolute_position, vg.chromosome,
vg.replace_tag, pi.ACCESSION, p.AMINO_ACID_SEQUENCE

FROM W_EHA_GENE_COMPONENT gc, W_EHA_PROTEIN p, W_EHA_PROT_INFO
pi, (

  SELECT gcs.GENE_COMPONENT_WID, vx.reference_id, v.absolute_
position, v.chromosome, v.replace_tag FROM W_EHA_GENE_COMP_
SEGMENT gcs, W_EHA_VARIANT v, W_EHA_VARIANT_XREF vx

  WHERE vx.reference_id = 'rs111690037'

    AND vx.VARIANT_WID = v.ROW_WID

    AND v.SOURCE_WID = gcs.SOURCE_WID

    AND v.START_POSITION <= gcs.END_POSITION
    AND gcs.START_POSITION <= v.END_POSITION
)vg
WHERE gc.ROW_WID = vg.GENE_COMPONENT_WID
  AND gc.component_type = 'CDS'
  AND gc.PROTEIN_WID = p.ROW_WID
  AND pi.protein_wid = p.row_wid
```

6.1.15 Scenario 15

Use Case - I have a pathway. What mutations are present in the pathway and which study were they identified in (for example, what tumor types)?

Areas

- Variant
- Gene Annotation
- Pathway Annotation

Output queries tables from - ODB: REFERENCE + RESULT

Query -

Firstly, identify the genes from the selected pathway then

```
truncate table query_25;
insert into query_25
select /*+ index(vr W_EHA_RSLT_SEQUENCING_F2) */ vr.result_spec_
wid, vg.hugo_name, vg.absolute_position, vg.chromosome,
vg.replace_tag from (
  SELECT g.hugo_name, v.row_wid, v.absolute_position,
v.chromosome, v.replace_tag from w_eha_variant v, w_eha_gene g,
w_eha_gene_segment gs, w_eha_dna_source s, w_eha_species sp
     where g.HUGO_NAME IN (
      select pthp.hugo_symbol from w_eha_pathway pth, w_eha_
pathway_protein pthp
      where pth.pathway_name = 'Apoptosis'
      AND pthp.pathway_wid = pth.row_wid
    )
   AND g.ROW_WID = gs.GENE_WID
   AND gs.SOURCE_WID = v.SOURCE_WID
   AND v.END_POSITION >= gs.START_POSITION
   AND gs.END_POSITION >= v.START_POSITION
   AND gs.source_wid = s.row_wid
   AND s.SPECIES_WID = sp.ROW_WID
   AND sp.SPECIES_NAME = 'Homo sapiens'
   ) vg, w_eha_rslt_sequencing vr, w_eha_rslt_study rst
```

```
where vr.VARIANT_WID = vg.row_wid

AND rst.result_study_name = 'study_1'

AND vr.result_study_wid = rst.row_wid

AND vr.result_spec_wid = 2;
```

6.1.16 Scenario 16

Use Case - I have a mutation I have deemed interesting, I want to know if it perturbs gene expression (in the same project or in other projects).

Areas

- Variant
- Expression
- Gene Annotation

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
truncate table query_29;

insert into query_29

SELECT /*+ index(vr W_EHA_RSLT_SEQUENCING_F2) */  exr.result_
spec_wid, exr.intensity, exr.p_value, exr.call, exr.probe_wid

from W_EHA_RSLT_GENE_EXP exr, W_EHA_VARIANT v, w_eha_rslt_
sequencing vr,

  W_EHA_VARIANT_XREF vx, W_EHA_GENE_SEGMENT gs, W_EHA_GENE g, W_
EHA_PROBE prb,  w_eha_rslt_study rst

    where vx.REFERENCE_ID = 'rs111690037'

    AND vx.VARIANT_WID = v.ROW_WID

    AND v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND v.END_POSITION >= gs.START_POSITION

    AND gs.GENE_WID = g.ROW_WID

    AND g.HUGO_NAME = prb.PRIMARY_HUGO_NAME

    and exr.probe_wid = prb.row_wid

    AND v.ROW_WID = vr.VARIANT_WID

    AND rst.result_study_name = 'study_1'

    AND vr.result_study_wid = rst.row_wid

    AND vr.result_spec_wid = exr.result_spec_wid

  AND vr.result_spec_wid > 0

  AND vr.result_spec_wid <= 100;
```

6.1.17 Scenario 17

Use Case - Examine gene expression across cell lines within each panel.

Areas

- Expression
- Gene Annotation
- Biospecimen Data
- Panel

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
truncate table query_30;

insert into query_30

SELECT  rge.result_spec_wid, p.primary_hugo_name,
rge.hybridization_name, rge.intensity, rge.p_value, rge.call

FROM w_eha_rslt_gene_exp rge, w_eha_probe p, w_eha_rslt_study
rst

WHERE  rge.PROBE_WID = p.row_wid

  AND rst.result_study_name = 'study_1'

  AND rge.result_study_wid = rst.row_wid

   AND rge.result_spec_wid = 6;
```

6.1.18 Scenario 18

Use Case - Examine copy number across cell lines within each panel.

Areas

- Copy Number
- Biospecimen Data
- Panel

Output queries tables from - ODB+CDM

Query -

```
/*[FOR SEQUENCING]*/

truncate table query_31a;

insert into query_31a

SELECT cnv.result_spec_wid, chr.chromosome, cnv.start_position,
cnv.end_position, cnv.avg_normalized_cvg

from w_eha_rslt_copy_nbr_var cnv, w_eha_rslt_study rst, w_eha_
chromosome chr

where rst.result_study_name = 'study_1'

  AND cnv.result_study_wid = rst.row_wid

  AND chr.row_wid = cnv.chromosome_wid

  AND cnv.result_spec_wid = 6;


-----------------------------------------------
```

```
/*[FOR EXPRESSION DATA]*/

truncate table query_31b;

insert into query_31b

SELECT exr.result_spec_wid, exr.hybridization_name,
exr.intensity, exr.p_value, exr.call

from W_EHA_RSLT_GENE_EXP exr, w_eha_rslt_study rst

where rst.result_study_name = 'study_1'

  AND exr.result_study_wid = rst.row_wid

  AND exr.result_spec_wid = 6;
```

6.1.19 Scenario 19

Use Case - Do KRAS mutant cell lines express more IL-6 than wildtype.

Areas

- Variant
- Protein
- Gene Annotation

Output queries tables from - ODB: REFERENCE + RESULT

Query19 -

```
truncate table query_32;

insert into query_32

SELECT rge.result_spec_wid, rge.intensity, rge.p_value, rge.call

FROM W_EHA_RSLT_GENE_EXP rge, W_EHA_PROBE prb, (

  SELECT DISTINCT rs.result_study_wid --, rs.result_spec_wid

    FROM w_eha_prot_info pi, W_EHA_GENE_COMPONENT gc, W_EHA_
GENE_COMP_SEGMENT gcs, W_EHA_VARIANT v, W_EHA_RSLT_SEQUENCING
rs, w_eha_rslt_study rst

  WHERE pi.PROTEIN_NAME = 'RASK_HUMAN'

    AND gc.PROTEIN_WID = pi.protein_wid

    AND gcs.GENE_COMPONENT_WID = gc.ROW_WID

    AND v.SOURCE_WID = gcs.SOURCE_WID

    AND v.START_POSITION <= gcs.END_POSITION

    AND gcs.START_POSITION <= v.END_POSITION

    AND rs.VARIANT_WID = v.ROW_WID

    and rst.result_study_name = 'study_1'

    and rs.result_study_wid = rst.row_wid

    AND rs.result_spec_wid  = 2

) ai

WHERE prb.PRIMARY_HUGO_NAME = 'IL6'
```

```
  AND rge.PROBE_WID = prb.ROW_WID
  AND rge.result_study_wid = ai.result_study_wid
  AND rge.result_spec_wid = 6
```

6.1.20 Scenario 20

Use Case - Do RAS high cell lines expression more IL-6 than low RAS lines.

Areas

- Gene Annotation
- Protein

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
truncate table query_33;
insert into query_33
SELECT er.result_spec_wid, er.intensity, er.p_value, er.call
from W_EHA_RSLT_GENE_EXP er, W_EHA_PROBE prb, w_eha_rslt_study
rst
 where prb.PRIMARY_HUGO_NAME = 'IL6'
  AND er.PROBE_WID = prb.ROW_WID
  AND rst.result_study_name = 'study_1'
  AND er.result_study_wid = rst.row_wid
  AND er.result_spec_wid = 6;
```

6.1.21 Scenario 21

Use Case - Are cMET amplified cell line more sensitive to METi?

Areas

- Gene Annotation
- Copy Number

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
SELECT  /*+ index (r_cnv W_EHA_RSLT_COPY_NBR_VAR_M1),index (r_
cnv W_EHA_RSLT_COPY_NBR_VAR_M2)*/
count(distinct r_cnv.result_spec_wid)
FROM (
select r_cnv1.* from W_EHA_RSLT_COPY_NBR_VAR r_cnv1, w_eha_rslt_
study rst
where rst.result_study_name = 'study_1'
  AND r_cnv1.result_study_wid = rst.row_wid
  AND r_cnv1.called_CNV_Type = 'gain'
) r_cnv, (
SELECT gsg.START_POSITION gsg_START_POSITION, gsg.END_POSITION
gsg_END_POSITION,

  s.START_POSITION s_START_POSITION, s.END_POSITION s_END_
POSITION

FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g, W_EHA_DNA_SOURCE s

WHERE g.HUGO_NAME IN ('MET')

  AND gsg.GENE_WID = g.ROW_WID

  AND gsg.SOURCE_WID= s.ROW_WID

) sg

WHERE r_cnv.START_POSITION <= (sg.gsg_end_position+sg.s_start_
position)

AND (sg.gsg_start_POSITION+sg.s_start_position) <= r_cnv.end_
position;
```

6.1.22 Scenario 22

Use Case - Show HER+ lines have ERBB2 copy number gains.

Areas

- Protein
- Biospecimen Data
- Gene Annotation
- Copy Number

Output queries tables from - ODB+CDM

Query -

```
SELECT  /*+ index (r_cnv W_EHA_RSLT_COPY_NBR_VAR_M1),index (r_
cnv W_EHA_RSLT_COPY_NBR_VAR_M2)*/

distinct r_cnv.called_CNV_Type

FROM W_EHA_RSLT_COPY_NBR_VAR r_cnv, w_eha_rslt_study,

(SELECT gsg.START_POSITION gsg_START_POSITION, gsg.END_POSITION
gsg_END_POSITION,

  s.START_POSITION s_START_POSITION, s.END_POSITION s_END_
POSITION

  FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g, W_EHA_DNA_SOURCE s

  WHERE g.HUGO_NAME IN ('ERBB2')

  AND gsg.GENE_WID = g.ROW_WID

  AND gsg.SOURCE_WID= s.ROW_WID

) sg

WHERE r_cnv.START_POSITION <= (sg.gsg_end_position+sg.s_start_
position)

AND (sg.gsg_start_POSITION+sg.s_start_position) <= r_cnv.end_
position
```

```
AND r_cnv.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

AND w_eha_rslt_study.result_study_name = 'study_6'

AND r_cnv.result_spec_wid = 550
```

6.1.23 Scenario 23

Use Case - Label cell lines for PTEN, PIK3CA mutations.

Areas

- Biospecimen Data
- Variant
- Gene Annotation

Output queries tables from - ODB+CDM

Query -

```
SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ count(distinct
vrt.RESULT_SPEC_WID)

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN  ('PTEN','PIK3CA')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study

WHERE w_eha_rslt_study.result_study_name = 'study_4'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

AND vrt.VARIANT_WID = vi.ROW_WID;
```

6.1.24 Scenario 24

Use Case - What is the frequency of co-mutation of two genes in a data set?

Areas

- Variant

Output queries tables from - ODB: REFERENCE + RESULT

Query -

```
/* concurrent count */

select count(a_and_b.result_SPEC_WID) concurrent_cnt
```

```
from

(SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ distinct
vrt.result_SPEC_WID

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN ('KRAS')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi, w_eha_rslt_study

WHERE vrt.VARIANT_WID = vi.ROW_WID

AND w_eha_rslt_study.result_study_name = 'study_4'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

intersect

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ distinct
vrt.result_SPEC_WID

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN ('BRCA2')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi, w_eha_rslt_study

WHERE vrt.VARIANT_WID = vi.ROW_WID

AND w_eha_rslt_study.result_study_name = 'study_4'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID) a_and_b


/* in A but not in B */
/* 38.5 sec*/

select count(a_not_b.result_SPEC_WID) a_not_b_cnt

from

(SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ distinct
vrt.result_SPEC_WID

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN ('KRAS')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study

WHERE vrt.VARIANT_WID = vi.ROW_WID

AND w_eha_rslt_study.result_study_name = 'study_3'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID

minus

SELECT /*+ index(vrt W_EHA_RSLT_SEQUENCING_F2) */ distinct
vrt.result_SPEC_WID

FROM W_EHA_RSLT_SEQUENCING vrt, (

  SELECT v.*

  FROM W_EHA_VARIANT v, (

    SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID

    FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g

    WHERE g.HUGO_NAME IN ('BCRA2')

    AND gsg.GENE_WID = g.ROW_WID

  )gs

  WHERE v.SOURCE_WID = gs.SOURCE_WID

    AND v.START_POSITION <= gs.END_POSITION

    AND gs.START_POSITION <= v.END_POSITION

) vi,w_eha_rslt_study

WHERE vrt.VARIANT_WID = vi.ROW_WID

AND w_eha_rslt_study.result_study_name = 'study_3'

AND vrt.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID) a_not_b
```

6.1.25 Scenario 25

Use Case - The steps for Initial data processing to identify sequence variants should be fully automated by writing scripts to interact with public database. The variants could then be presented in a graphic interface that summarize their characteristics.

Output queries tables from - ODB: REFERENCE

Query -

```
truncate table query_53

drop table query_53

create global temporary table query_53(reference_id
varchar2(200), reference_data varchar2(100)) ON COMMIT DELETE
ROWS;

insert into query_53

SELECT distinct vx.reference_ID, vx.database

from W_EHA_VARIANT v, w_eha_variant_xref vx, W_EHA_GENE g, W_
EHA_GENE_SEGMENT gs, W_EHA_HUGO_INFO hg

where hg.ENTREZ_GENE_ID = 56624

AND hg.approved_symbol = g.hugo_name

AND g.ROW_WID = gs.GENE_WID

AND gs.source_wid = v.source_wid

AND gs.END_POSITION >= v.START_POSITION

AND v.END_POSITION >= gs.START_POSITION

AND v.row_wid = vx.variant_wid;
```

6.1.26 Scenario 26

Use Case - Map the variant coordinates to gene, exon, amino acid, chromosomal location, region of protein; and obtain CCDS (Consensus Coding Sequence) and RefSeq IDs. Example: PDEGFR D842V, exon 18 out of 23 total exons, 4q11-q13, protein tyrosine kinase domain; CCDS3495.1, NP_006197.

Output queries tables from - ODB: REFERENCE

Query 26 A -

```
/*a. Map the variant coordinates to gene and obtain CCDS
(Consensus Coding Sequence) and RefSeq IDs. Example: PDEGFR;
CCDS3495.1, NP_006197*/

/* 24.3 sec */

truncate table query_58a

drop table query_58a

create global temporary table query_58a(ROW_WID NUMBER, start_
position NUMBER, end_position NUMBER,

replace_tag varchar2(1000), gcx_reference_id varchar2(200), px_
reference_id varchar2(200)) ON COMMIT DELETE ROWS;

insert into query_58a
```

```
SELECT v.ROW_WID, v.start_position, v.end_position, v.replace_
tag, sg.gcx_reference_id, sg.px_reference_id

FROM W_EHA_VARIANT v, (

  SELECT gcsg.SOURCE_WID, gcsg.START_POSITION, gcsg.END_
POSITION, gcx.reference_id gcx_reference_id, px.reference_id px_
reference_id

  FROM W_EHA_GENE g, w_eha_gene_structure gst, w_eha_gene_
component gc,  w_eha_gene_comp_segment gcsg,

  w_eha_gene_comp_xref gcx, w_EHA_PROT_XREF px

  WHERE g.HUGO_NAME IN ('PTEN')

  AND gc.COMPONENT_TYPE IN ('CDS')

  AND gc.STRUCTURE_WID = gst.ROW_WID

  AND gst.GENE_WID = g.ROW_WID

  AND gcsg.GENE_COMPONENT_WID = gc.ROW_WID

  AND gcx.gene_component_wid = gc.row_wid

  AND gcx.database in ('CCDS')

  and px.database in ('RefSeq')

  and px.protein_wid = gc.protein_wid

)sg

WHERE v.SOURCE_WID = sg.SOURCE_WID

  AND v.START_POSITION <= sg.END_POSITION

  AND sg.START_POSITION <= v.END_POSITION
```

Query 26 B -

```
/*b. Map to exon and obtain CCDS (Consensus Coding Sequence) and
RefSeq IDs.Example: PDGFRA, exon 18 out of 23 total exons ;
CCDS3495.1, NP_006197*/

/* 6.4 sec */

truncate table query_58b

drop table query_58b

create global temporary table query_58b(ROW_WID NUMBER, start_
position NUMBER, end_position NUMBER,

replace_tag varchar2(1000), gcx_reference_id varchar2(200), px_
reference_id varchar2(200)) ON COMMIT DELETE ROWS;

insert into query_58b

SELECT v.ROW_WID, v.start_position, v.end_position, v.replace_
tag, sg.gcx_reference_id, sg.px_reference_id

FROM W_EHA_VARIANT v, (

  SELECT gcsg.SOURCE_WID, gcsg.START_POSITION, gcsg.END_
POSITION, gcx.reference_id gcx_reference_id, px.reference_id px_
reference_id
```

```
  FROM W_EHA_GENE g, w_eha_gene_structure gst, w_eha_gene_
component gc,  w_eha_gene_comp_segment gcsg,

  w_eha_gene_comp_xref gcx, w_EHA_PROT_XREF px

  WHERE g.HUGO_NAME IN ('KRAS')

  AND gc.COMPONENT_TYPE IN ('CDS')

  AND gc.STRUCTURE_WID = gst.ROW_WID

  AND gst.GENE_WID = g.ROW_WID

  AND gcsg.GENE_COMPONENT_WID = gc.ROW_WID

  AND gcsg.number_in_sequence = 2

  AND gcx.gene_component_wid = gc.row_wid

  AND gcx.database in ('CCDS')

  and px.database in ('RefSeq')

  and px.protein_wid = gc.protein_wid

)sg

WHERE v.SOURCE_WID = sg.SOURCE_WID

  AND v.START_POSITION <= sg.END_POSITION

  AND sg.START_POSITION <= v.END_POSITION
```

Query 26 C -

```
/*c. Map to chromosomal location and obtain CCDS (Consensus
Coding Sequence) and RefSeq IDs. 4q11-q13(4:74301933..74321492);
*/

/* 6.6sec */

truncate table query_58c

drop table query_58c

create global temporary table query_58c(hugo_name varchar2(200),
start_position NUMBER, end_position NUMBER, chromosome
varchar2(50),

replace_tag varchar2(1000), gcx_reference_id varchar2(200), px_
reference_id varchar2(200)) ON COMMIT DELETE ROWS;

insert into query_58c

SELECT sg.hugo_name, v.start_position, v.end_position,
v.chromosome, v.replace_tag, sg.gcx_refid, sg.px_refid

from W_EHA_VARIANT v, (

  SELECT distinct g.hugo_name, gcs.source_wid, gcs.start_
position, gcs.end_position, gcx.reference_id as gcx_refid,
px.REFERENCE_ID as px_refid

  from W_EHA_GENE g, w_eha_gene_comp_xref gcx, W_EHA_GENE_
COMPONENT gc, w_eha_gene_comp_segment gcs,  w_eha_prot_xref px,
W_EHA_GENE_STRUCTURE gs, (

    select ds.row_wid, ds.start_position, ds.end_position from
w_eha_dna_source ds
```

```
    where ds.chromosome in ('4')
      and ds.start_position <= 74321492
      and ds.end_position >= 74301933
  ) ref_chr
  where gcs.source_wid = ref_chr.row_wid
  and (ref_chr.start_position + gcs.start_position - 1) <=
74321492
  and (ref_chr.start_position + gcs.end_position - 1) >=
74301933
  and gcs.gene_component_wid = gc.row_wid
  and gc.component_type = 'CDS'-- can also be 'exons'
  and gs.row_wid = gc.structure_wid
  and g.row_wid = gs.gene_wid
  and px.protein_wid = gc.protein_wid
  and px.database = 'RefSeq'
  and gcx.gene_component_wid = gc.row_wid
  and gcx.database = 'CCDS'
)sg
where v.SOURCE_WID = sg.source_wid
AND v.START_POSITION <= sg.END_POSITION
AND sg.START_POSITION <= v.END_POSITION
```

Query 26 D -

```
/*d. Map to region of protein and obtain CCDS (Consensus Coding
Sequence) and RefSeq IDs. protein tyrosine kinase domain;
result: CCDS3495.1, NP_006197 */
/* 44.6 sec*/
truncate table query_58d
drop table query_58d
create global temporary table query_58d(hugo_name varchar2(200),
start_position NUMBER, end_position NUMBER, chromosome
varchar2(50),
replace_tag varchar2(1000), gcx_reference_id varchar2(200), px_
reference_id varchar2(200)) ON COMMIT DELETE ROWS;
insert into query_58d
SELECT distinct sg.hugo_name, v.start_position, v.end_position,
v.chromosome, v.replace_tag, sg.gcx_refid, sg.px_refid
from W_EHA_VARIANT v, (
  select distinct g.hugo_name, gcs.source_wid, gcs.start_
position, gcs.end_position, gcx.reference_id as gcx_refid,
px.REFERENCE_ID as px_refid
```

```
  from W_EHA_GENE g, w_eha_gene_comp_xref gcx, W_EHA_GENE_
COMPONENT gc, w_eha_gene_comp_segment gcs,  w_eha_prot_xref px,
w_eha_gene_structure gs

  where px.protein_wid in (

          select px2.protein_wid from w_eha_prot_xref px2

          where px2.reference_id IN ('PROTEIN_KINASE_TYR')

            and px2.database = 'PROSITE')

    and px.protein_wid = gc.protein_wid

    and px.database = 'RefSeq'

    and gcx.gene_component_wid = gc.row_wid

    and gcx.database = 'CCDS'

    and gc.component_type = 'CDS'-- can also be 'exons'

    and gcs.gene_component_wid = gc.row_wid

    and gs.row_wid = gc.structure_wid

    and g.row_wid = gs.gene_wid

) sg

where v.SOURCE_WID = sg.source_wid

  AND v.START_POSITION <= sg.END_POSITION

  AND sg.START_POSITION <= v.END_POSITION
```

6.1.27 Scenario 27

Use Case - Determine whether the variant is a known SNP or a previously characterized somatic mutation in cancer. Also determine the total number of reported somatic variants in the entire gene, and whether any are near the variant in question.

Output queries tables from - ODB: REFERENCE

Query -

```
/*a) Check if the variant is a known COSMIC variant*/

/* 0.1sec */

SELECT v.row_wid, vx.database from W_EHA_VARIANT v, W_EHA_
VARIANT_XREF vx

where

   v.ROW_WID = vx.VARIANT_WID

   AND vx.DATABASE IN ('COSMIC', 'dbSNP_132')

   AND vx.REFERENCE_ID = 'rs11169';

/*b) Determine total no. of reported somatic variants in entire
gene*/

/* 0.48sec */

SELECT v.*

FROM W_EHA_VARIANT v, w_eha_variant_xref vx, (
  SELECT gsg.START_POSITION, gsg.END_POSITION, gsg.SOURCE_WID
  FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g
  WHERE g.HUGO_NAME IN ('PTEN')
  AND gsg.GENE_WID = g.ROW_WID
)gs
WHERE v.SOURCE_WID = gs.SOURCE_WID
  AND v.START_POSITION <= gs.END_POSITION
  AND gs.START_POSITION <= v.END_POSITION
  AND vx.VARIANT_WID = v.ROW_WID
  AND vx.DATABASE in ('COSMIC', 'dbSNP_132')
```

6.1.28 Scenario 28

Use Case - Show me all patients whose cancer cells had a deletion in gene X.

Output queries tables from - ODB+CDM

Query -

```
SELECT  /*+ index (r_cnv W_EHA_RSLT_COPY_NBR_VAR_M1),index (r_
cnv W_EHA_RSLT_COPY_NBR_VAR_M2)*/
distinct r_cnv.START_POSITION, r_cnv.END_POSITION, r_cnv.called_
CNV_Type
FROM W_EHA_RSLT_COPY_NBR_VAR r_cnv, w_eha_rslt_study,
(SELECT gsg.START_POSITION gsg_START_POSITION, gsg.END_POSITION
gsg_END_POSITION,
  s.START_POSITION s_START_POSITION, s.END_POSITION s_END_
POSITION
  FROM W_EHA_GENE_SEGMENT gsg, W_EHA_GENE g, W_EHA_DNA_SOURCE s
  WHERE g.HUGO_NAME IN ('PTEN')
  AND gsg.GENE_WID = g.ROW_WID
  AND gsg.SOURCE_WID= s.ROW_WID
) sg
WHERE r_cnv.START_POSITION <= (sg.gsg_end_position+sg.s_start_
position)
AND (sg.gsg_start_POSITION+sg.s_start_position) <= r_cnv.end_
position
AND r_cnv.RESULT_STUDY_WID = w_eha_rslt_study.ROW_WID
AND w_eha_rslt_study.result_study_name = 'study_5'
AND r_cnv.RESULT_SPEC_WID = 450;
```

6.2 Global Temporary Table Creation Code Snippets

Following are the code snippets for creating global temporary tables:

- table query_1
- table query_2
- table query_2a
- table query_4
- table query_5
- table query_6
- table query_7
- table query_8
- table query_9
- table query_10
- table query_11
- table query_12
- table query_15
- table query_17
- table query_18
- table query_19a
- table query_26a
- table query_26b
- table query_26c
- table query_26d

6.2.1 table query_1

```
create global temporary table query_1(SPECIMEN_WID NUMBER,
start_position NUMBER, end_position NUMBER, replace_tag
varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.2 table query_2

```
create global temporary table query_2(specimen_wid NUMBER,
primary_hugo_name varchar2(200), intensity number, start_
position NUMBER, end_position NUMBER, replace_tag
varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.3 table query_2a

```
create global temporary table query_2_a(replace_tag
varchar2(1000)  , start_position NUMBER, end_position NUMBER,
specimen_wid number) ON COMMIT DELETE ROWS;
```

6.2.4 table query_4

```
create global temporary table query_4(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.5 table query_5

```
create global temporary table query_5(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.6 table query_6

```
create global temporary table query_6(SPECIMEN_WID NUMBER,
start_position NUMBER, end_position NUMBER, replace_tag
varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.7 table query_7

```
create global temporary table query_7(SPECIMEN_WID NUMBER, hugo_
name varchar2(200), start_position NUMBER, end_position NUMBER,
replace_tag varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.8 table query_8

```
create global temporary table query_8(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.9 table query_9

```
create global temporary table query_9(SPECIMEN_WID NUMBER,
start_position NUMBER, end_position NUMBER, replace_tag
varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.10 table query_10

```
create global temporary table query_10(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.11 table query_11

```
create global temporary table query_11(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.12 table query_12

```
create global temporary table query_12(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), intensity NUMBER)
ON COMMIT DELETE ROWS;
```

6.2.13 table query_15

```
ccreate global temporary table query_15( specimen_wid number,
absolute_position number, chromosome varchar2(50), replace_tag
varchar2(1000))
ON COMMIT DELETE ROWS;
```

6.2.14 table query_17

```
create global temporary table query_17(SPECIMEN_WID NUMBER,
primary_hugo_name varchar2(200), hybridization_name
varchar2(200), intensity NUMBER, p_vall NUMBER, exp_call
varchar2(100))
ON COMMIT DELETE ROWS;
```

6.2.15 table query_18

```
create global temporary table query_18b(SPECIMEN_WID NUMBER,
hybridization_name varchar2(200), intensity NUMBER, p_vall
NUMBER, exp_call varchar2(100))
ON COMMIT DELETE ROWS;
```

6.2.16 table query_19a

```
create global temporary table query_19_a(specimen_wid number) ON
COMMIT DELETE ROWS;
```

6.2.17 table query_26a

```
create global temporary table query_26a(row_wid number, START_
POSITION number, END_POSITION number, replace_tag varchar2(200),
gcx_reference_id  varchar2(200), px_reference_id  varchar2(200))
ON COMMIT DELETE ROWS;
```

6.2.18 table query_26b

```
create global temporary table query_26b(row_wid number, START_
POSITION number, END_POSITION number, replace_tag varchar2(200),
gcx_reference_id  varchar2(200), px_reference_id  varchar2(200))
ON COMMIT DELETE ROWS;
```

6.2.19 table query_26c

```
create global temporary table query_26c(HUGO_NAME
varchar2(200), START_POSITION number, END_POSITION number,
chromosome varchar2(200), replace_tag varchar2(200), gcx_
reference_id  varchar2(200), px_reference_id  varchar2(200))  ON
COMMIT DELETE ROWS;
```

6.2.20 table query_26d

```
create global temporary table query_26d(HUGO_NAME
varchar2(200), START_POSITION number, END_POSITION number,
chromosome varchar2(200) ,replace_tag varchar2(200), gcx_
reference_id  varchar2(200), px_reference_id  varchar2(200))
ON COMMIT DELETE ROWS;
```

7 Miscellaneous Topics

This chapter contains the following topics:

7.1 Querying Database Cross-references for Variations

7.1.1 Ensembl db_xref Qualifier Issue

ENSEMBL GVF file which involves nucleotide variation references from dbSNP, COSMIC and EMBL is used to populate variation tables in Omics Data Bank. The cross-reference information for these variants is identified by *Dbxref* qualifier and is loaded to W_EHA_VARIANT_XREF table. The standard format for *Dbxref* in GVF file is specified below.

```
Dbxref=dbSNP_132:rs79772382;
```

The program to import this data splits it in to DATABASE and REFERENCE_ID using first colon (:) as delimiter. Hence, for the above example W_EHA_VARIANT_XREF columns are being populated with following data:

```
DATABASE = 'dbSNP_132'
```

```
REFERENCE_ID = 'rs79772382'
```

But for some organisms, there is a slight change in the way Dbxref is defined. Following is an example from Rattus norvegicus GVF file.

```
Dbxref=ENSEMBL:celera:ENSRNOSNP2610581;
```

For such cases W_EHA_VARIANT_XREF columns are being populated with following data:

```
DATABASE = 'ENSEMBL'
```

```
REFERENCE_ID = 'celera:ENSRNOSNP2610581'
```

Since REFERENCE_ID may contain suffixed or prefixed data for some of the cases mentioned above, when querying the REFERENCE_ID, we suggest using the SQL LIKE operator.

Note: REFERENCE_SUFFIX column for W_EHA_VARIANT_XREF will not be populated with any data in the current model.

The same scenario exists for SwissProt database cross-reference and hence users are suggested to use SQL LIKE operator for querying against W_EHA_PROT_XREF table.

7.1.2 Swissprot db_xref Qualifier Issue

The database cross-reference information for SwissProt is stored in W_EHA_PROT_XREF. This table populates DATABASE, REFERENCE_ID and REFERENCE_SUFFIX information. The standard format for database cross-reference in SwissProt file is specified below.

```
DR   InterPro; IPR007031; Poxvirus_VLTF3.
```

The program to import this data splits it in to DATABASE, REFERENCE_ID and REFERENCE_SUFFIX using semi-colon (;) as delimiter. Hence, for the above example W_EHA_PROT_XREF columns are being populated with following data:

```
DATABASE = 'InterPro'

REFERENCE_ID = 'IPR007031'

REFERENCE_SUFFIX = 'Poxvirus_VLTF3'
```

There are certain cross-references in SwissProt file which have same REFERENCE_ID but different REFERENCE_SUFFIX. Due to indexing on REFERENCE_ID, only distinctthe first found REFERENCE_ID is stored leaving out other records with the same REFERENCE_ID.

For example:

```
DR   EMBL; AL390732; CAH71826.2; JOINED; Genomic_DNA.

DR   EMBL; AL390732; CAH73848.1; -; Genomic_DNA.
```

For the above example the REFERENCE_ID for both the cross-references is AL390732, hence due to indexing only first line information is stored in the table.

There is some loss of information on the REFERENCE_SUFFIX level but not on REFERENCE_ID, ie., all REFERENCE_ID would be captured in W_EHA_PROT_XREF table.

7.2 Mitochondrial Chromosome Mappings

The references to mitochondrial chromosome are stored as MT in the reference side of the model, namely in W_EHA_VARIANT and W_EHA_HUGO_INFO tables. Any novel variants reported into W_EHA_VARIANT table from the result files will have the chromosome value converted from M to MT. When inserting into result tables, namely W_EHA_RSLT_COPY_NBR_VAR, the chromosome value will be M or as specified in the result file. Thus any queries must ensure to map any reference to M in result tables to MT in reference tables of ODB.

7.3 Promoter Offset

The promoter region information is not available in the reference data set imported from ENSEMBL, therefore a column has been provided in W_EHA_SPECIES table for you to specify the promoter region upstream to the gene for a specific organism. This is by default taken as input parameter while installation of ODB. This value is stored PROMOTER_OFFSET column of W_EHA_SPECIES table. Alternatively, you can change this value later on after installation of ODB by editing W_EHA_ SPECIES.PROMOTER_OFFSET column.

7.4 CGI End Position

End position refers to the last nucleotide in the affected area. CGI counts the end position of variants differently than MAF and GVF, that have end position values. CGI positions are zero based and all other data formats are one based. The code has already compensated for the zero based positions.

CGI treats the end position of insertion differently than single nucleotide variants (SNV). All other formats compute the end position as the same for both SNV and insertions. CGI sets the end position as identical to the start position for insertions, and then has the end position as + 1 for SNV. CGI also has a different computation for end position for each additional size of reference variants. In this release of ODB, the code has been modified to make the CGI loader compute the end position in the same way as VCF and MAF loaders. CGI stores end position as the nucleotide just past the affected variation.

The other formats' method of calculating the end position has been adopted because the GVF file has to be looked at as the authoritative reference since this data comes from Ensembl and we are following the GVF format and position calculation.

A
Additional Result Tables

The appendix contains the following topics:

- Pre-Seeded Tables on page A-1
- Populated by User or Loader on page A-2
- Unpopulated Result Entities or Tables on page A-4

A.1 Pre-Seeded Tables

W_EHA_RSLT_TYPE

To store information regarding type of result stored and is based on what data is being inserted (per row inserted by loader from file). You may choose to seed more types.

Table A–1 W_EHA_RSLT_TYPE

Table Name	Column Name	Values to Seed With	Values to Seed With	Values to Seed With	Values to Seed With
W_EHA_ RSLT_TYPE	RESULT_ TYPE_NAME	SEQUENCIN G	NOCALL	GENE_ EXPRESSIO	COPY_ NUMBER_ VARIATION
W_EHA_ RSLT_TYPE	RESULT_ TYPE_DESC	Sequencing results including simple variants such as snp, insertions, deletions	Nocall result for sequencing given allele	Gene expression results	Copy Number Variation results
Which loader inserts above type(s)		VCF, MAF, CGI loaders	CGI loader	Gene expression loader	No loader support

W_EHA_RSLT_FILE_TYPE

The W_EHA_RSLT_FILE_TYPE table is to be preseeded with file types currently handled by loaders, it should contain 6 rows and the pre-seeded values are mentioned in the following table:

Table A–2 W_EHA_RSLT_FILE_TYPE

Column Name	Description	Values preseeded	Values preseeded	Values preseeded	Values preseeded	Values preseeded	Values preseeded
FILE_TYPE_CODE	Short code for input file type	VCF	MAF	MAF	MAF	CGI masterVar	Tab-delim Expression
FILE_TYPE_NAME	Longer name of file type	Variant Call Format	Mutation Annotation Format	Mutation Annotation Format	Mutation Annotation Format	Complete Genomics MasterVar	Tab delimited Expression file
FILE_TYPE_DESC	Description of file type	File containing variant information including snps, inserts and deletions	Mutation Annotation Format containing snps, inserts, and deletions	Mutation Annotation Format containing snps, inserts, and deletions	Mutation Annotation Format containing snps, inserts, and deletions	Master Variation file from Complete Genomics containing snps, inserts, deletions, and no-call information	Gene Expression tab delimited file format containing probe hybridization results, 3 values per hybridization: Intensity, Call, P-value
FILE_TYPE_VERSION	Version of file type	4.1	2.0	2.1	2.2	2.0	A

W_EHA_RSLT_CHROMOSOME

This table is pre-seeded with all the possible chromosome names in a result. The user needs to insert any non-standard chromosome names contained in the results file.

Table A–3 W_EHA_RSLT_CHROMOSOME

Table Name	Column Name	Description	Values Pre-seeded
W_EHA_RSLT_CHROMOSOME	CHROMOSOME	Name of the chromosome	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y,M

A.2 Populated by User or Loader

W_EHA_DATASOURCE

Tthis table is used to store information about specimen source and is intended to be used primarily with CDM (one record in W_EHA_DATASOURCE). However, if needed, other databases with specimen information can be linked.

Table A–4 W_EHA_DATASOURCE

Table Name	Column Name	Description	If used together with Oracle Health Sciences Cohort Explorer (OHSCE) Cohort Data Model
W_EHA_ DATASOURCE	DATASOURCE_CD	Data source for Specimen	OHSCE Cohort Data Model1.1
W_EHA_ DATASOURCE	DATASOURCE_NM	Name of datasource for Specimen	OHSCE Cohort Data Model1.1
W_EHA_ DATASOURCE	DATASOURCE_ DESC	Description of datasource for specimen	OHSCE v1.1 - Cohort Data Model
W_EHA_ DATASOURCE	SCHEMA_NAME	Name of schema	CDM
W_EHA_ DATASOURCE	DB_LINK_NAME	Link to database if needed	

W_EHA_RSLT_FILE

This table is populated by loaders while loading results.

Table A–5 W_EHA_RSLT_FILE

Table Name	Column Name	Description	If Used Together With Regular Files	If Used Together With SecureFiles
W_EHA_RSLT_ FILE	FILE_ STORAGE_FLG	E for External, S for SecureFiles	E	S
W_EHA_RSLT_ FILE	FILE_PATH	Path to input file	For example, C:/inputfile.txt	
W_EHA_RSLT_ FILE	VENDOR_ NAME	Name of vendor providing file	For example, Affymetrix	For example, Affymetrix
W_EHA_RSLT_ FILE	FILE_ CONTENT_ID	File identifier utilized by Secure File system	Not used	Generated by SecureFiles
W_EHA_RSLT_ FILE	FILE_TYPE_ WID	FK to W_EHA_ RSLT_FILE_ TYPE	Corresponds to WID in RSLT_ FILE_TYPE	Corresponds to WID in RSLT_ FILE_TYPE

W_EHA_RESULT_STUDY

The user should populate study details in this table before loading the results. All imported results fall under the specified study name in the command line argument.

Table A–6 W_EHA_RESULT_STUDY

Table Name	Column Name	Description	Values Pre-seeded
W_EHA_RESULT_ STUDY	RESULT_STUDY_ NAME	Name of the study	<user defined values>
W_EHA_RESULT_ STUDY	RESULT_STUDY_ DESC	Description of the study	<user defined values>

A.3 Unpopulated Result Entities or Tables

The following table indicates result tables that are currently not being populated.

Table A–7 Unpopulated Result Entities or Tables

Table Name	Column Name	Description
W_EHA_PROBE	SEQUENCE	
W_EHA_PROBE	START_POSITION	
W_EHA_PROBE_XREF		No loader for this table
W_EHA_RSLT_COPY_NBR_VAR		No loader for this table
W_EHA_RSLT_CNV_X		No loader for this table
W_EHA_PROBE_ALT_LINK		No loader for this table

Index