Oracle® Health Sciences Omics Data Bank

Programmer's Guide

Release 3.0.2.1

E35680-12

March 2016

ORACLE®

Oracle Health Sciences Omics Data Bank Programmer's Guide Release 3.0.2.1

E35680-12

Contents

3 Loaders for Reference Data

4 Loaders for Result Data

7 Miscellaneous Topics

A Additional Result Tables

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, Developers, and Data Modelers.

Disclaimer Regarding Third Party Data

Public Domain 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 that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=accid=info or visit http://www.oracle.com/pls/topic/lookup?ctx=accid=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

The *Oracle Health Sciences Translational Research Center Online Documentation Library* documentation set includes:

- *Oracle® Health Sciences Translational Research Center User's Guide*
- *Oracle® Health Sciences Translational Research Center Administrator's Guide*
- *Oracle® Health Sciences Translational Research Center Release Notes*
- *Oracle® Health Sciences Translational Research Center Installation Guide*

- *Oracle® Health Sciences Translational Research Center Release Content Document*

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, called the reference set, provides the genomic feature metadata required to link specimen sample results to specific regions of the genome, gene definitions, pathway or protein definitions. The second set of tables, called the result set, captures the specimen sample genomic results, and links each result to an object in the reference model. Each specimen sample is linked back to the patient and (or) subject. The patient or subject link is accomplished by linking ODB with Cohort Data Model (part of Oracle Health Sciences Cohort Explorer).

Omics Data Bank (ODB) consists of the data model and loaders. It does not include any front-end user interfaces and contains only command line APIs. A set of scripts, included with the model, can be used to load reference genomic data from specific sources and several types of result data from a limited set of formats.

The model is intended to handle very large amounts of data (of the order of terabytes and more). To gauge the scale of the data, one should consider that the human genome has 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 are described in detail. This holds true for each protein, gene component, pathway and so on, thus a lot of supporting reference data is loaded for each gene, protein or 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.

ODB 3.0 can handle multiple versions of reference data in the same instance, for example GRCh36, GRCh37. You can load multiple versions of each reference data, even the same version multiple times, and link the results to all copies of a specific

reference version based on the requirement. For example, user with sequencing data mapped to GRCh36.p7 links the results to GRCh36.p7 reference data loaded from ENSEMBL. The multiple reference version support is extended to ENSEMBL, SwissProt, HUGO, Pathway, SIFT or PolyPhen, COSMIC, HGMD, ADF, and probe loader.

The ODB model can also handle multiple species concurrently, both on the reference and result side of the data model.

1.1.1 Reference Data

Reference data is loaded from the following 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 databases. The model loads this cross reference information, including known variation data, to let queries use specific database references, if needed. Files in the genuine EMBL format from other sources can be usually loaded in the same way (However, their successful loading cannot be guaranteed). Each release of Ensembl data is treated as a separate 'DNA' reference version. Current and alternate releases are available at their FTP publication repository. Each newly loaded 'DNA' reference version should be given a new label. When loading variants or proteins, each given variant or protein record is linked to a specific version of the DNA reference.
2. The second source is the online SwissProt database (http://www.ebi.ac.uk/uniprot/) from which the model obtains protein information. This database project is also a consortium of various groups including the European Bioinformatics Institute. An FTP link to the current release of the SwissProt file is provided on their download page. Older SwissProt releases are stored as large compressed files in their FTP repository.
3. The third source is the HUGO Gene Nomenclature Committee (http://www.genenames.org/). This source provides reference seed information required to identify human gene locations annotation only. The HUGO gene names are required to find various cross references and the correct chromosome number for each gene. The HUGO Gene Nomenclature Committee is the authoritative group for all gene names. Since the HGNC HUGO dataset is continuously updated, it does not have an archive of older versions or releases. Any dataset taken, is as current as the date it was retrieved.
4. The fourth reference source is PathwayCommons (http://www.pathwaycommons.org/pc/), which is used as the source for published pathways and proteins or genes participating in each pathway. The coverage of pathway as a reference is minimal. Pathway provides a list of current and previous releases of the dataset to be retrieved from its download page.
5. The fifth reference source is Human Genome Mutation Database (HGMD) (http://www.hgmd.cf.ac.uk/ac/index.php) This reference source currently provides information on inherited genomic variants, as well as information connecting inherited mutations and genes with human diseases and pharmacological effects. The latter is obtained from HGMD's commercial partner, BioBase International.
6. The sixth reference source is Catalogue of Somatic Mutations in Cancer (COSMIC) (http://cancer.sanger.ac.uk/cancergenome/projects/cosmic), which is a

source of information related to human cancers including somatic mutation, sample annotation, and publication ID links. This provides datasets on coding and non-coding variants, and corresponding annotative information for each, consisting of sample origin, histology, publication, and genomic data.

In general, the above files use similar features to represent 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 have 3NF structures, which are cumbersome 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 the ODB schema.

Most of these online databases let you download complete references, or specific references for sections of the genome. Since some customers may only need some genes or proteins, and some may not need any protein information at all, the model permits any combination of specific data to be loaded and 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 be expanded as required by the customer. The HUGO, PathwayCommons, and HGMD databases do not keep older versions but provide only the latest versions for download.

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, or how they differentially express under different conditions. For sequencing results, while there are many different types of sequencing techniques, the net effect is to record all of the variants which include SNPs, small indels, large structural variations, structural re-arrangements, and also non-variant information detected for each sample being tested, copy number variation and other related features.

The model is designed to facilitate easy querying of all the above result types in a single SQL statement across multiple versions of references or within the specific reference version.

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, and link results to reference. However, there is one exception, the W_EHA_ VARIANT reference table, where the sequencing 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. Only table names are shown in the figure.

Figure 1–1 Reference Data Logical Model (Core Tables Only)

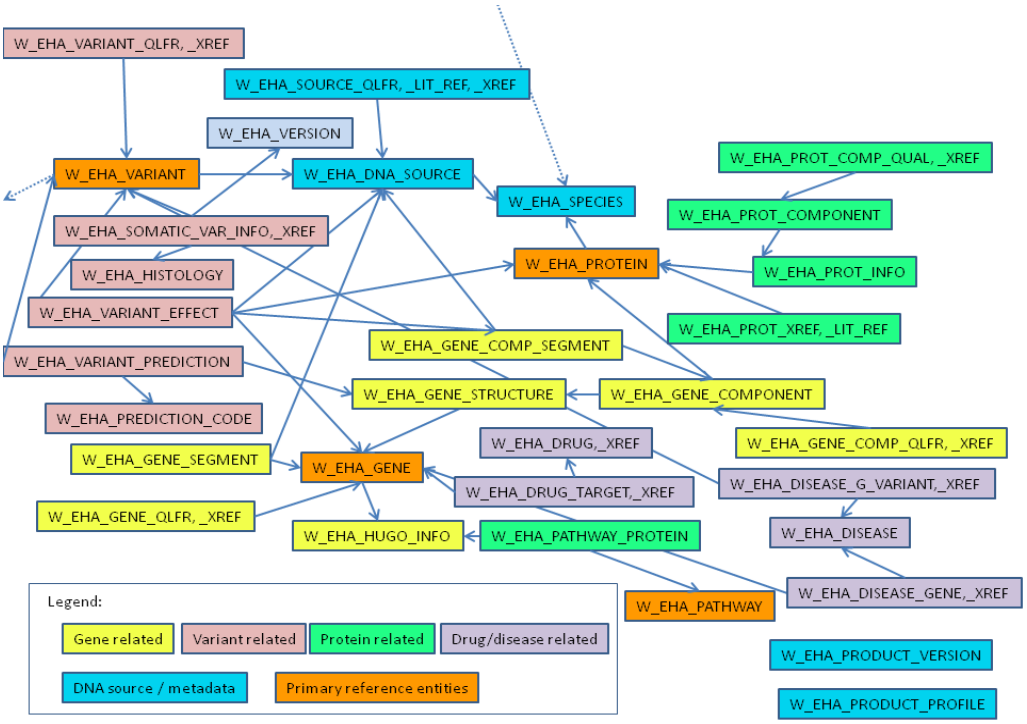
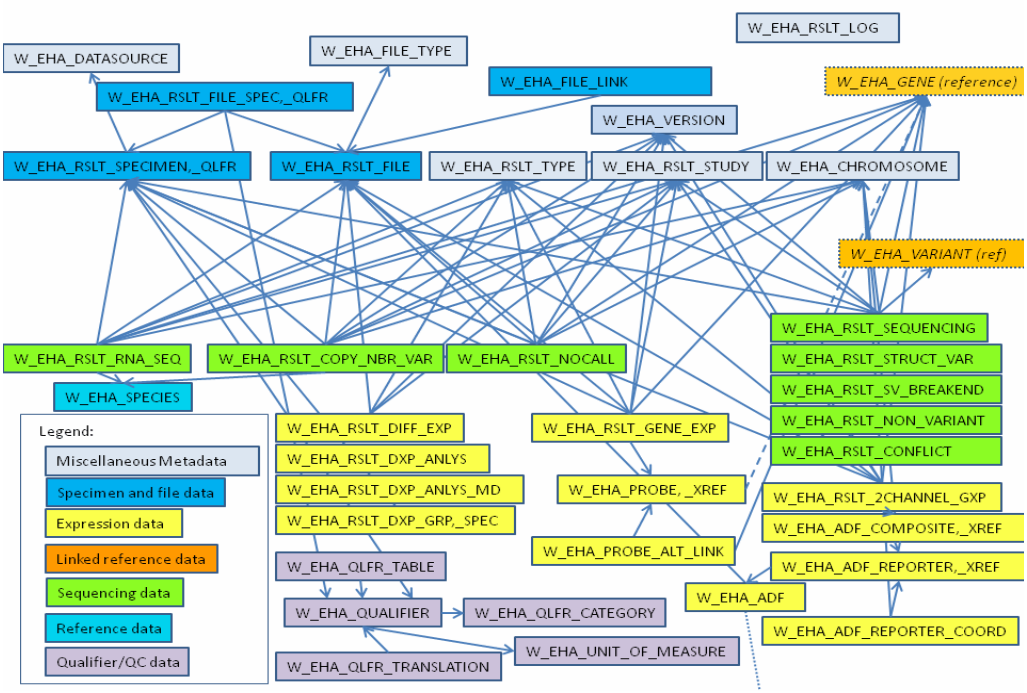


Figure 1–2 shows how the result tables link to create the ODB result tables section. Only table names are shown in the figure.

Figure 1–2 Result Data Logical Model (Core Tables Only)



1.3 Reference Data Tables

The reference data starts with the W_EHA_SPECIES and W_EHA_DNA_SOURCE tables.

W_EHA_SPECIES

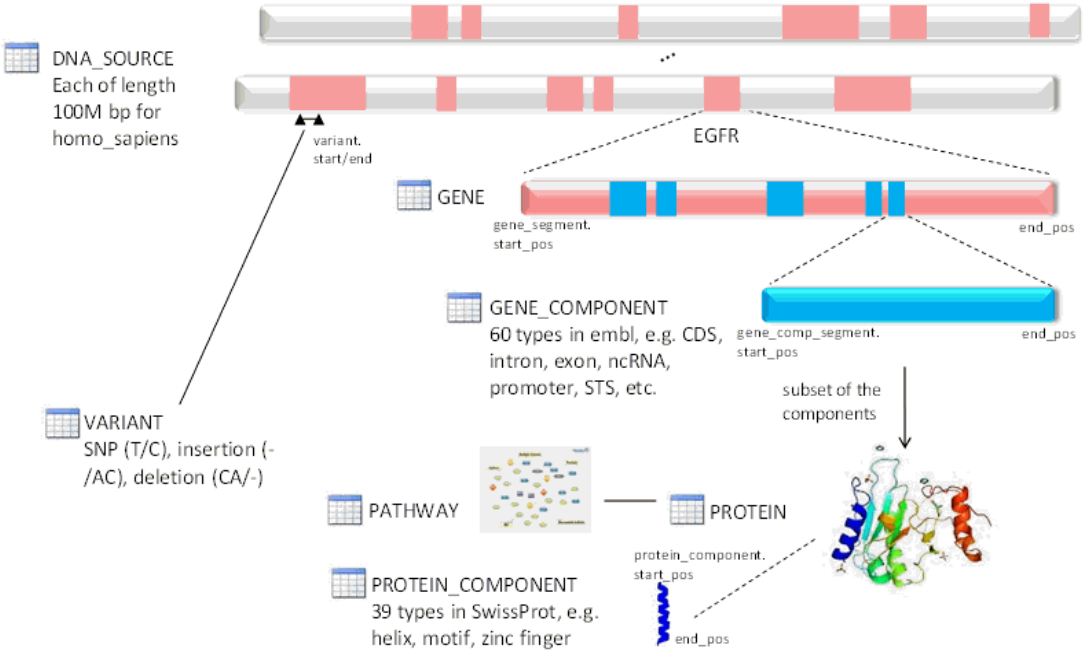
The species table stores information about each genome in the database. The current model permits any number of species genomes to be loaded. You must specify species in queries if there are similar genes between the organisms being tested.

The table also stores the promoter offset value that is used to define the promoter region of each gene, linked to the species record, if no PROMOTER component is defined. If set, this value overrides the global promoter offset defined in W_EHA_PRODUCT_PROFILE table.

W_EHA_DNA_SOURCE

The W_EHA_DNA_SOURCE table stores multiple records for different reference DNA sequences for each species. Each cell in the species has a copy of this reference DNA. There are buffers of DNA considered to be the reference for each organism. These reference sequences are then used to map detected variations for each organism tested. The W_EHA_DNA_SOURCE table has a foreign key to W_EHA_SPECIES and a CLOB column to store the reference nucleotide sequence information. The DNA table uses a specific notation to keep track of each DNA base in addition to the four standard characters: A, T, C, G. There are additional characters used for sections that have not been sequenced (*N*) and other characters are 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 illustrate how some genes are re-defined, new W_EHA_DNA_SOURCE records are created and linked to the other records, as required. This table also stores the chromosome location, which is described later. The position of features such as variants, segments, and so on in ODB is 1-based, following the standard from Ensembl. With the introduction of multiple reference support, W_EHA_DNA_SOURCE table links each reference DNA Source record to its DNA version, which is stored in the W_EHA_VERSION table under version type 'DNA'.

Figure 1–3 Genome Division by the Data Model



W_EHA_GENE

Each chromosome encodes for many different genes, each of which has a start and end position. The entire start-to-end region of the gene typically does not create the protein directly. Instead, there are recognized sections with the start-to-end region that scientists agree should be considered as part of the gene. The W_EHA_GENE table models how the Ensembl database refers to the gene, and 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 as 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 DNA Source buffers. Some genes are sequenced in multiple buffers and require this joining table to track each segment. This table provides the location in the buffer and a sequence number to keep track of the order of each segment that composes the gene. There is also a COMPLEMENT field that is used to indicate if the coding strand of a gene is on the reverse strand (COMPLEMENT=1) or forward strand (COMPLEMENT=0) of a chromosome.

_XREF, _QLFR (QUALIFIER)

The _XREF suffixed tables associated with many of the different tables are used to list all the cross reference information stored in the Ensembl, SwissProt, and other databases. 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 _QLFR suffixed tables associated with other tables are used to list other attributes. Each object can have an unlimited number of attributes such as */note* that provides information to annotate the object. The database model stores this annotation data 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 by having the same TRANSCRIPT_ID qualifier. A GENE_STRUCTURE record is created to link to the protein and 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 has 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 each has a specific meaning. Views are 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 are given (such as mRNA = MESSENGER_RNA, CDS = CODING_REGION, STS = STRUCTURAL_SEGMENTS, and so on). Various queries can be run 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 W_EHA_DNA_SOURCE records. Many 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 files list the amino acids that comprise each protein molecule, and use an identifier for each protein molecule. The SwissProt files contain both amino acid (AA) sequences and additional information about the protein molecule. There are more descriptive names for each protein that are not stored in the EMBL files (such as, insulin). The W_EHA_PROTEIN table only stores AA sequences, and facilitates merging data from EMBL and SwissProt files for the proteins having the same sequences. This table is unversioned, and the additional data for proteins is stored in a separate table, W_EHA_PROT_INFO.

W_EHA_PROT_INFO

This table stores information about proteins, loaded from SwissProt files. It is linked to the W_EHA_PROTEIN table containing the AA sequences. Multiple W_EHA_PROT_COMPONENT records can be linked to a single W_EHA_PROT_INFO record. This table is versioned, that is, each record in it is associated with one or more versions of type 'PROTEIN' (through the W_EHA_PROT_INFO_VERSION table). This multi-version link is unique to the W_EHA_PROT_INFO table. It is used to prevent duplicates of the Protein information records (and dependent records in other tables) that have not changed from version to version.

W_EHA_PROT_COMPONENT

This table stores all the protein components that are loaded from SwissProt files. This data may also be important for queries or reference. It can be important to show changes that may be occurring 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 generate new VARIANT records, which may be of interest to researchers. There is a STATUS field which is used to indicate NOVEL or KNOWN variants. Since this table is queried frequently, it is quite large and requires partitioning. The VARIANT table has a foreign key to the DNA_SOURCE record, not the GENE record. This is because some genes may overlap, and there may also be several structures that are affected by a variant. This table is used to create result foreign keys as described later.

The Variant table contains a column PRECEDING_BASE, which stores the nucleotide base value preceding an insertion or deletion. The column, SVTYPE, stores the type of the structural variation, including insertions (ins), deletions (del), duplications (dup), copy number variations (cnv). The preceding base value is given, if and only if, SVTYPE value is not null for a variant record.

W_EHA_VARIANT_X

This table has a foreign key to W_EHA_VARIANT table and only stores the allele value for the large structural variant coming from the VCF file. This value comes from the ALT column present in the VCF file.

W_EHA_HUGO_INFO

This table is very important to store reference seed information needed for identifying gene locations. The EMBL files 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 is loaded as seed data in this table. 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 and the correct chromosome number for each gene. Each gene in the HUGO_INFO table is linked to a specific version of HUGO reference data.

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 the PATHWAY_PROTEIN table which has a foreign key to the PATHWAY table. Each pathway in this table is linked to the W_EHA_VERSION table with specific version of Pathway Commons build release.

W_EHA_PATHWAY_PROTEIN

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

W_EHA_VARIANT_PREDICTION

This table stores information relevant to SIFT (Sorting Intolerant From Tolerant) or Polymorphism Phenotyping (Polyphen) algorithms scores. SIFT or Polyphen are

publicly available algorithms describing the impact of each variant on the resulting gene structure. The impact is evaluated both as a numeric score and a formal annotation, such as deleterious, probably damaging and so on.

Polyphen is an automated tool for predicting the possible impact of an amino acid substitution on the structure and function of a human protein. This prediction is based on straightforward empirical rules which are applied to the sequence, phylogenetic, and structural information characterizing the substitution. Possible annotation values are *probably damaging, possibly damaging, benign, and unknown*.

Sorting Intolerant From Tolerant (**SIFT**) predicts whether an amino acid substitution affects protein function. This prediction is based on the degree of conservation of amino acid residues in sequence alignments derived from closely related sequences, collected through PSI-BLAST. Possible annotations are *tolerated, and, deleterious*.

W_EHA_PREDICTION_CODE

This table stores the possible annotations for SIFT or Polyphen algorithms that are mentioned above, such as *probably damaging*, *possibly damaging*, *deleterious*, *tolerated*, and so on.

W_EHA_VARIANT_EFFECT

This table stores variant impact or effect as computed by an Oracle proprietary script (stored procedure) based on variant effect on the resulting protein. The possible values of net effects are as follows:

- Unknown — used when a variant is not in a coding region, or intronic, or crosses splice boundaries that would affect translation in unknown ways.
- Frame-shift — used for insertion, deletion, indel variants that add or remove a number of nucleotides not divisible by 3 (the size of a codon).
- Nonsynonymous - missense — signifies that a single nucleotide change results in a codon that codes for a different amino acid.
- Nonsynonymous - nonsense — signifies that the stop codon occurs abnormally from the variant data in the coding region.
- Synonymous — signifies that the variant in the coding region does not cause an amino acid change.

> **Note:** The variant impact script has been temporarily removed from ODB 3.0.

W_EHA_PRODUCT_PROFILE

This table stores default global Promoter and Flanking offsets which are the inputs provided during installation. It also stores various log level flags and DBMS output for data load ETLs and other procedure calls. Another column in this table 'VCF_ FORMAT' stores a list of data types for the FORMAT column in the VCF file. Promoter offset is used during querying (through Promoter view: W_EHA_PROMOTER_V) and can be changed at any point. If PROMOTER_OFFSET in W_EHA_SPECIES is set, this promoter offset takes precedence over the one in W_EHA_PRODUCT_PROFILE.

Important

1. Flanking offset should always be set to a value greater than Promoter offset. Promoters are assumed to fit within the Flanking offset region.

2. Changing the Flanking offset requires reloading all results tables which use genomic coordinates, such as sequencing and copy number variation result data. It is imperative to keep Flanking offset *unchanged*.

The logging level flags that can be set by the user include: Warning, and Info (which are set by default to 'Y'); Debug, TRACE, and DBMS output which are set by default to 'N').

W_EHA_PRODUCT_VERSION

This table lists the current version of the Omics Data Model and is used primarily by the Cohort Explorer application user interface (not included in this release). For example, currently it is set to 3.0.

W_EHA_DISEASE

This table stores names of diseases with possible genetic linkage.

W_EHA_DISEASE_GENE

This table stores literature-derived associations between diseases and genes that might contain disease causing mutations. It aggregates the mutation-disease linkage reported in the W_EHA_DISEASE_G_VARIANT table and associates the whole gene sequence with diseases caused by mutations in the gene. It also contains disease linkage for genes with disease causing variants for which no exact genomic coordinates were provided.

W_EHA_DISEASE_G_VARIANT

This table stores disease linkage for variants with known genomic coordinates. It includes linkage confidence provided by HGMD curators.

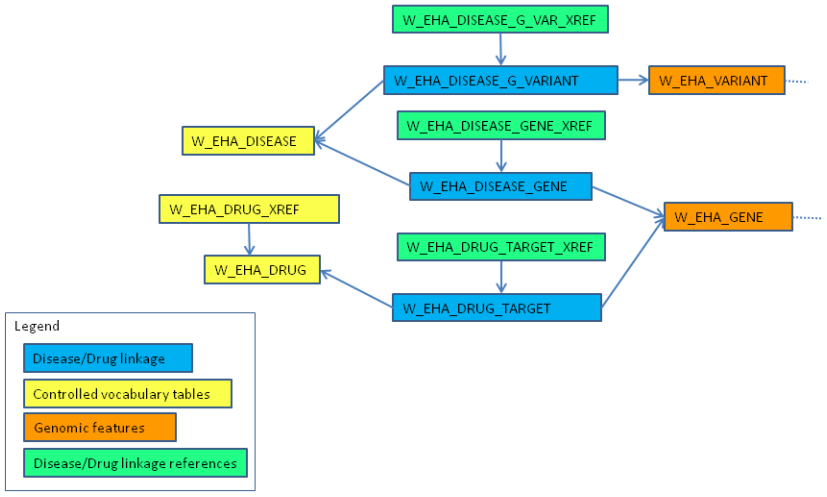
W_EHA_DRUG

This table stores DrugBank-derived drug names.

W_EHA_DRUG_TARGET

This stores drug and gene associations linking DrugBank-derived drug names to their therapeutic targets.

Figure 1–4 ODB Disease and Drug Linkage



W_EHA_ADF

This table stores the configuration information for the Array Data Format (ADF) files, which contain the annotation data required by Two-Channel gene expression result datasets. An ADF dataset loads into the W_EHA_ADF_* reference tables described below. Each file load creates an ADF record and the records inserted into W_EHA_ADF_COMPOSITE and W_EHA_ADF_REPORTER will have an 'ADF_WID' foreign key column to this table.

W_EHA_ADF_COMPOSITE

This table stores the gene composite elements associated with the 2-channel result data present in W_EHA_RSLT_2CHANNEL_GXP table. The composite element coordinates are input from the array design file (ADF) for the AgilentG4502A_07 platform. An additional table W_EHA_ADF_COMPOSITE_XREF is kept to store any external cross reference data for composite elements.

W_EHA_ADF_REPORTER

This table stores the probe (reporter) elements associated to the composite gene element present in W_EHA_ADF_COMPOSITE table. The reporter identifiers are input from the array design file (ADF) for the AgilentG4502A_07 platform. An additional table W_EHA_ADF_REPORTER_XREF is kept to store any external cross reference data for Reporters.

W_EHA_ADF_REPORTER_COORD

This table stores the probe (reporter) elements genomic coordinates associated with the reporter indentifiers present in the W_EHA_ADF_REPORTER table. The reporter genomic coordinates are input from the array design file (adf) for the AgilentG4502A_07 platform.

W_EHA_PROBE

This table holds 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 the same gene. In the rare instance where more than one gene matches a probe, the model has a W_EHA_PROBE_ALT_LINK that should 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_CANCER_S_VARIANT

This table stores the link between gene names, somatic variants reported to COSMIC, both non-coding and coding, its sample annotation, the sample's reference keys to Histology and Anatomical site data, and mutation related annotations. The table has a reference key to W_EHA_VARIANT table as a main link to other reference source tables.

W_EHA_CANCER_S_VARIANT_XREF

This table contains cross reference data to CANCER S(SUBJECT) VARIANT bridge table. The primary cross reference data stored in this table are PubMed publication IDs of associated somatic mutations.

W_EHA_CANCER_S_GENE_XREF

This table contains cross Reference data to CANCER S(SUBJECT)_GENE bridge table. The primary cross reference data stored in this table are PubMed publication IDs of associated somatic mutations.

W_EHA_CANCER_S_GENE

This table links gene annotation including Refseq and/or Genbank, and Ensembl gene IDs to somatic mutation cancer sample data submitted to Cosmic. The table stores a count of variants and linked gene mutations that is found for every gene symbol (hugo name) reported for each reference sample ID given by COSMIC. Each sample links to its histology and anatomical site references.

W_EHA_REFERENCE_SAMPLE

The table contains information on Reference samples studied for observed somatic mutations, which were submitted to COSMIC. The table provides sample annotation including cosmic tumor and sample IDs, sample source, tumor origin along with sample name. The table also references Histology and Anatomical sites associated to Samples.

W_EHA_ANAT_PRIMARY_SITE

This table stores the primary descriptive name of an anatomical site of a linked reference sample.

W_EHA_ANAT_SUBTYPE_SITE

This table stores the descriptive name of an anatomical site subtype of a linked reference sample.

W_EHA_HISTOLOGY_PRIMARY

This table stores the primary Histology description of a linked reference sample.

W_EHA_HISTOLOGY_SUBTYPE

This table stores the descriptive Histology subtype name of a linked reference sample.

W_EHA_GEN_CODE

This table stores the parent record to name a set of genetic codon translations to amino acids.

W_EHA_GEN_CODE_TABLE

This table stores the link that each codon has with its corresponding Amino Acid.

W_EHA_GEN_CODE_USAGE

This table maps how GEN_CODE tables are used with each species and chromosome.

1.4 Result Data Tables

The model currently supports the following two major categories of results:

- Sequencing
- Gene Expression

Types of sequencing results include simple variants, copy number variation, and no-call. Gene expression results comprise of regular microarray gene expression or RNA-seq results. Overall, results are populated into the following major tables:

- W_EHA_RSLT_SEQUENCING
- W_EHA_RSLT_GENE_EXP
- W_EHA_RSLT_NOCALL
- W_EHA_RSLT_RNA_SEQ
- W_EHA_RSLT_2CHANNEL_GXP
- W_EHA_RSLT_COPY_NBR_VAR
- W_EHA_RSLT_NON_VARIANT
- W_EHA_RSLT_CONFLICT
- W_EHA_RSLT_SV_BREAKEND

All the major result tables listed above contain foreign keys to the following tables:

- W_EHA_FILE_LOAD
- W_EHA_RSLT_STUDY
- W_EHA_RSLT_SPECIMEN
- W_EHA_RSLT_TYPE
- W_EHA_GENE
- W_EHA_VERSION

Each record in these tables is linked to a specific reference 'DNA' source Version Label in W_EHA_VERSION table by the VERSION_WID foreign key. Additionally, where possible, each record is linked to a specific W_EHA_GENE record to allow for a gene based partitioning of these result tables. Those records that cannot be linked to a gene record have a GENE_WID value of '0'.

Chromosome Partitioned Tables

For Exadata, some of the result tables listed above have counterpart tables created dynamically during install, having the same columns as existing result tables. The new tables added are named using the same name as the parent result table with a *_CHR* suffix. These tables include:

- W_EHA_RSLT_CONFLICT_CHR
- W_EHA_RSLT_COPY_NBR_VAR_CHR
- W_EHA_RSLT_NOCALL_CHR
- W_EHA_RSLT_NON_VARIANT_CHR
- W_EHA_RSLT_RNA_SEQ_CHR
- W_EHA_RSLT_SEQUENCING_CHR
- W_EHA_RSLT_SV_BREAKEND_CHR

For details on chromosome partitioned tables, see Section 7.11, "Chromosome Partitioned Tables" on page 1-18.

W_EHA_RSLT_SEQUENCING

This table contains sequencing results, more specifically variant information. It has a foreign key to the W_EHA_VARIANT table. The records in this table are linked to the record in the variant reference table. Information such as insertion, deletion, or substitution is recorded in this table along with any quality metrics on this information. The result sequencing table also stores large structural variants.

> **Note:** A record in the W_EHA_RSLT_SEQUENCING table may come from any of the four result file types, namely gVCF, VCF, MAF, or Complete Genomics masterVar file.

W_EHA_RSLT_NOCALL

This table contains results coming from VCF, gVCF and Complete Genomics sequencing files. Only CGI masterVar 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_NON_VARIANT

This table contains the non-variant information belonging to a specimen coming from VCF and gVCF files. The VCF loader populates this table when used in either 'GVCF' mode or 'NON-VAR' mode, provided the file contains non-variant information.

W_EHA_RSLT_CONFLICT

This table contains the low quality score variants which conflict with an existing high quality score variant reported in W_EHA_RSLT_SEQUENCING table for the same specimen. This table is populated through the VCF loader with data coming from gVCF files.

W_EHA_RSLT_SV_BREAKEND

This table contains the structural rearrangement data coming from VCF files.

W_EHA_RSLT_COPY_NBR_VAR

This table contains copy number variation results coming from the Complete Genomics platform and data from Affymetrix Genome-Wide Human SNP Array 6.0 along with any relevant quality or count metrics. This table is populated through the newly provided CNV loader, which can load the .SEG format file, and has been verified to load data from TCGA belonging to Affymetrix Genome-Wide Human SNP Array 6.0. No loader exists to load data from CGI format, however, the tables are designed to accommodate any attributes specific to CGI CNV data.

W_EHA_RSLT_CNV_X

This 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_RSLT_GENE_EXP

This table is loaded from gene expression results and permits storing gene intensity measurements and 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 to establish a foreign key relationship.

W_EHA_RSLT_RNA_SEQ

This table is loaded from TCGA RNA SEQ file format for RPKM expression information of exons. The supplied loader only supports the loading of the exon version of the data files. TCGA has 3 different types of files: exon, gene, and splice junctions. Only the exon files are measured by exact chromosome locations. The other two file types are calculated estimations based upon gene locations using the exon data file.

W_EHA_RSLT_2CHANNEL_GXP

This table is loaded from TCGA – Level 3, AgilentG4502A_07 platform specific, microarray dual channel gene expression files. For each gene record, the loader resolves two values it loads to the result table:

- ADF_COMPOSITE_WID - by looking up the gene composite name in w_eha_adf_composite table.
- GENE_WID - by looking up gene segment table for all complete and partial overlaps of gene segments in reference to gene composites genomic coordinates.

The Array Design file information associated with this data is loaded in the ADF Composite and Reporter tables in ODB, described in the Reference tables section of this chapter.

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 Section A, "Additional Result Tables". Each record in the W_EHA_RSLT table supports a single specific result type.

W_EHA_STUDY

This table permits each result to be linked to a study if 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, called the Cohort Explorer Data Model study table, and only holds the 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 enables partitioning the results into groups based on a study for which the results have been collected. Depending on the customer data and query requirements, by-study partitioning can be used as an alternative partitioning scheme to by-gene partitioning. Currently, partitioning by gene is the default.

W_EHA_CHROMOSOME

This table holds all the chromosome names and is pre-seeded with names for all Human chromosomes. Refer to Section 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. As with the W_EHA_STUDY table, this table is used to partition results for improved query performance.

W_EHA_CHROM_MAPPING

This table stores a list of aliases to chromosomes present in the W_EHA_CHROMOSOME table. Initially this table is seeded with a common chromosome alias list.

W_EHA_RSLT_SPECIMEN

The W_EHA_RSLT_SPECIMEN table is linked to all 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. The W_EHA_RSLT_SPECIMEN table in turn, links to the 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 used to uniquely identify and pull more metadata about a given specimen from other source systems. This information is not stored in the ODB. If the specimen database is CDM, two more tables are used: W_EHA_SPEC_PATIENT and W_EHA_SPEC_SUBJECT, linking a W_EHA_RSLT_SPECIMEN record with a CDM pationet/subject.

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 to be used with Cohort Explorer data model, this table should be seeded with one source of specimen samples, which is the Cohort Data Model.

W_EHA_FILE_TYPE

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

W_EHA_FILE_LOAD_QLFR

These tables contain the header information from VCF and gVCF files. Any line starting with '##' in the VCF and gVCF file is stored in the File Load Qlfr table. This is also where the alternative file location is stored, if the user optionally chooses to specify `-alt_file_loc` in the argument list when loading results.

W_EHA_RSLT_FILE_SPEC

This table stores the foreign key to W_EHA_FILE and W_EHA_RSLT_SPECIMEN and basically links a specific file which is loaded to ODB by a loader to one or more specimen.

1.4.1 Result Tables for Qualifier Metadata

The ODB data model has new tables for Qualifier Metadata attributes. These tables include:

W_EHA_QUALIFIER

This table describes qualifiers - flexible attributes used in _QLFR tables. Qualifiers extend the concept of name/value pair attributes: they could be assigned to one of three data types - CHARACTER, NUMERIC and DATE and are grouped into functional categories. Units of measure can be specified for numeric qualifiers.

W_EHA_QLFR_CATEGORY

Qualifiers can be grouped based on functional categories. For example, a user might want to create qualifier categories such as DNA Sequencing, Gene Expression, RNA Sequencing, and CNV.

W_EHA_QLFR_TABLE

This table lists all qualifier tags applicable to a specific table.

W_EHA_UNIT_OF_MEASURE

This table holds names of a unit of measure.

W_EHA_QLFR_TRANSLATION

This table stores translation rules to convert values from one unit of measure into another.

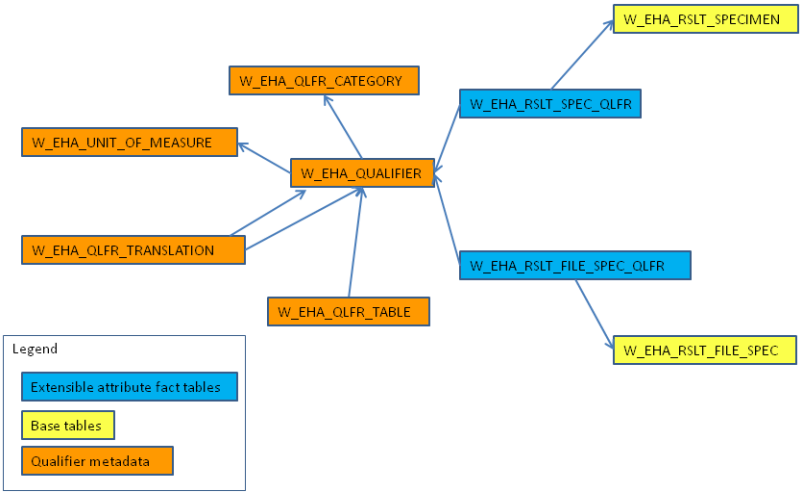
W_EHA_RSLT_FILE_SPEC_QLFR

This is the fact table that stores key or value pairs for flexible attributes associated with a particular result file or specimen combination.

Each _QLFR table has a QLFR_WID attribute that points to a QUALIFIER_TAG record in the W_EHA_QUALIFIER table. It also has a foreign key attribute that references a record from a base table. For W_EHA_RSLT_FILE_SPEC_QLFR table the base table is W_EHA_RSLT_FILE_SPEC.

Three separate attributes are used to store character, numeric and date values. If a qualifier data type is NUMBER the QLFR_NUMB_VALUE attribute is populated, else it is empty. Similarly QLFR_DATE_VALUE field is populated for the DATE data type qualifiers. The QLFR_CHAR_VALUE attribute holds a reported value and is populated for any qualifier data type.

Figure 1–5 Qualifier Metadata Tables



1.4.2 Result Tables for Differential Expression

Newly enhanced, the ODB data model hosts tables for differential gene expression analysis results. However, there is no loader provided to populate result set to these tables. These tables include:

W_EHA_RSLT_DIFF_EXP

The main table used to store results from differential expression files.

W_EHA_RSLT_DXP_ANLYS

Stores a listing of differential analysis result-sets loaded.

W_EHA_RSLT_DXP_ANLYS_MD

Stores metadata for differential expression analysis result-sets.

W_EHA_RSLT_DXP_GRP

Stores and describes a list of differential expression groups of specimens.

W_EHA_RSLT_DXP_GRP_SPEC

Links specimens to differential expression groups.

1.5 Table for Logging

All loaders and procedures created in the ODB output will log records to a single table.

W_EHA_RSLT_LOG

This table stores logging information from all loaders and jobs. The column RESULT_TYPE_NAME specifies the loader populating a given logging record. Each record stores species, specimen, datasource, OS user, host, and etl_proc_wid details taken from a loader run. The LOG_LEVEL column stores the logging level pertaining to a specific record. The log record includes details of record that caused a log entry, error info or trace fields, and a log summary field.

1.6 Aggregate Tables for Gene Expression

In queries for Gene Expression (single-channel) intensities are often compared not to a set value, but to the minimum (maximum, average, and so on) of all intensities for a hybridization or a probe. The queries, performing aggregation, are rather inefficient. To make them more efficient, the following two aggregate tables have been added for Gene Expression:

- W_EHA_RSLT_GXP_HYBRID_AGG
- W_EHA_RSLT_GXP_PROBE_AGG

The first table aggregates over a hybridization, while the second aggregates over a probe. They have almost the same set of aggregated columns, such as MEDIAN_INTENSITY, AVG_INTENSITY, MIN_INTENSITY, MAX_INTENSITY, and so on. Each of them also has columns, by which aggregation is done on the W_EHA_RSLT_GENE_EXP table (FILE_LOAD_WID and PROBE_WID in W_EHA_RSLT_GXP_PROBE_AGG, and FILE_LOAD_WID and HYBRIDIZATION_NAME for W_EHA_RSLT_GXP_HYBRID_AGG).

These tables are updated automatically by the Single Channel Results Loader every time a new Single-Channel Gene Expression file load is performed, and are automatically used by the TRC UI 3.0, when building Gene Expression queries.

1.7 File and File Load Tables

All file and file load tables are populated by the result loaders, the gvf and adf reference loaders where there is a requirement to store file metadata references for records inserted.

W_EHA_FILE

This table stores the name of the input file used to load a result dataset. The file contains, along with the file name used, a unique global identifier (URI) for the file. This identifier is generated, if not provided by the user, on result load.

W_EHA_FILE_LOAD

This table stores a record for each file load for a file. The table contains the field file_load_seg_numb that counts the number of times the file with the same URI is loaded into ODB. The table is referenced by all main result data tables and the view W_EHA_FILE_LAST_LOAD_V that displays the last file load record for a file with the same FILE_URI value.

W_EHA_FILE_LOAD_QLFR

The table is used to store metadata information for each file load.

2

Prerequisites for Loading Data

This chapter contains the following topics:

2.1 Setting Up a Directory Object

All loaders (except for EMBL and SwissProt) use external tables to access data in files. This requires an Oracle directory object to be created. Oracle directory objects require the database Operating System (OS) user account to have access to this directory. Therefore, the directory must be mounted and all permissions granted before the database server is started. For more information about creating directory objects, see *Oracle Database SQL Language Reference 11g Release 2*.

The directory object name is used as a parameter to all the loaders. The ODB schema user must have the CREATE ANY DIRECTORY and CREATE ANY TABLE privilege. The Oracle database OS account must have permissions to access the directory specified in the Oracle directory object.

An example of a command to create oracle directory objects is:

```
>create directory TRC_ODB as /home/oracle/perm_loc;
```

Note: The directory used must reflect the path requirements of the operating system that the database server is installed on. Windows database servers have different naming conventions 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, grant READ and WRITE privileges on the directory to other users as follows:

```
GRANT READ on DIRECTORY <<DIR_NAME>> TO <<ODB USER NAME>>

GRANT WRITE on DIRECTORY <<DIR_NAME>> TO <<ODB USER NAME>>
```

Here *DIR_NAME* is the name of an Oracle directory where all the result files are kept and *ODB USER NAME* is the database user executing the loaders. The database user should have both READ and WRITE grants on Oracle directory to process loaders.

2.2 Setting Up an Oracle Wallet

An Oracle Wallet must be set up with the credentials used to connect to the schema where ODB is installed.

Perform the following steps to set up the 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 to point to your database installation.

2. The Oracle wallet can be created on the client or 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 odb

Your secret/Password is missing in the command line

Enter your secret/Password: <enter password for odb 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 odb.

4. Configure SQLNET to look for the 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 using sqlplus. Enter the following on any command prompt terminal:

```
>sqlplus /@DB001_Wallet
```

You will get the following result:

```
SQL*Plus: Release 11.2.0.1.0 Production on Fri June 7 15:54:35 2013

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 Setting Up User Privileges for Querying or Loading Data

Perform the following steps to set up user privileges for querying or loading data:

1. Create a database user having the following privileges:
   - CREATE SYNONYM
   - CREATE SESSION
2. Assign the user an appropriate role from the following three roles:

- OmicsDatamartUser

  Queries ODB schema. Can only perform queries on the model, cannot write to the model. This role will be typically given to named UI users whose credentials are to be passed to the database layer for querying only.

- OmicsDatamartAdmin

  Loads data into the reference side of ODB, refreshes all reference data into the reference side of the schema including W_EHA_VARIANT table. This user cannot create new data definitions for objects such as tables and views.

- OmicsDatamartContributor

  Can load result data into the Omics Data Bank through provided result loaders. This role enables a named user to write to the ODB result side of the model.

3. Create a local synonym for the database user. The create_synonym_for_user.sql script is available in the *master_install* folder in the TRC Software package.
   a. Connect to the database instance with the above created database user.
   b. Execute the following command by replacing cdm_Schema_name with the name of cohort explorer schema name, odb_schema_name with the actual name of your Omics Data Bank schema, apps_schema_name with the application schema name, and job_schema_name with job engine schema name:

```
@create_synonym_for_user.sql <<cdm_schema_name>> <<odb_schema_
name>> <<apps_schema_name>> <<job_schema_name>>
```

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

Integration with other data models is done through the specimen record. Each genomic data result file must be accompanied by SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER information, and SPECIMEN_DATASOURCE. These entities should match a record in the specimen datasource schema. OHSCE Data Model is the default datasource for specimen in the current release. ODB contains three tables, W_EHA_SPEC_PATIENT, W_EHA_SPEC_SUBJECT, and W_EHA_SPEC_EXTERNAL, each having a foreign key, RSLT_SPECIMEN_WID, to the W_EHA_RSLT_SPECIMEN table. All three tables have a column to store the external table ID value to register a specific result specimen. For W_EHA_SPEC_PATIENT in ODB, this value is SPEC_PATIENT_WID, which should match ROW_WID in the W_EHA_SPECIMEN PATIENT_H table in the OHSCE Data Model. Similarly, OHSCE's subject specific content can be linked through W_EHA_SPEC_SUBJECT.SPEC_SUBJECT_WID matched to SUBJECT specific table W_EHA_SPECIMEN_SUBJECT_H. The table W_EHA_SPEC_EXTERNAL is used to register external non-OHSCE specimen sources, where the external specimen ID is matched to EXT_SPECIMEN_WID.

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 call a specific stored procedure to validate that the SPECIMEN exists.

It is assumed that any database used to provide the SPECIMEN information has a single numeric field to identify the correct SPECIMEN. This numeric value is stored in each result record (without direct FK definition). Three stored function calls are provided as options to the user through a stored procedure in a package ODB_UTIL.

The procedure is not wrapped so that additional external databases can be supported. Currently, the function calls are implemented to support validating SPECIMEN records stored in the Cohort Data Model. The three function calls are:

- ODB_UTIL.validate_cdm_patient_spec: Used to validate against specimens in CDM's W_EHA_SPECIMEN_PATIENT_H table.
- ODB_UTIL.validate_cdm_subject_spec: Used to validate against specimens in CDM's W_EHA_SPECIMEN_SUBJECT_H table.
- ODB_UTIL.validate_cdm_both_spec: Used to validate against specimens to both CDM tables.

This stored procedure can be expanded to support other schemas that are intended to provide specimen data.

Following is the structure of ODB_UTIL.GET_SPECIMEN_WID stored procedure that contains the above functions:

```
function get_specimen_wid
(
i_datasource_id in number
, i_specimen_number in varchar2
, i_specimen_vendor in varchar2
, i_study_id in number
, i_etl_proc_id in number
, i_enterprise_id in number
, i_file_wid in number
, i_logger in result_logger default null
, i_lock_table in number default 1
) return number;
```

This stored procedure has seven mandatory and two optional parameters:

- DATASOURCE_ID: W_EHA_DATASOURCE table is used to configure each external database to provide SPECIMEN data. The VALIDATION_PROC column of this table should be populated with the ODB_UTIL.VALIDATE_CDM_ SPECIMEN value. This procedure validates the specimen in CDM (or external data source) schema.
- SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER: These two VARCHAR2 fields are used to identify a unique specimen.
- ETL PROC WID and ENTERPRISE WID are sent by the loader as an input parameter to procedure.
- FILE WID is sent by loader. It is the ROW_WID of W_EHA_FILE table that creates the records for the result file used for processing the loader. This is used by the stored procedure to create a record in W_EHA_RSLT_FILE_SPEC table.

The stored procedure looks up the W_EHA_DATASOURCE record and compares the name field. Currently, there is a check for a CDM name and code for searching specimen data in CDM. If additional database schemas should be used to provide specimen information, you must first add a record to W_EHA_DATASOURCE with a unique name.

The stored procedure has to specifically handle that data source name and validate the specimen number and specimen vendor number passed. Most data files support a specimen number and the loaders currently have a specimen vendor number passed as a parameter.

If the specimen exists in the CDM schema (W_EHA_SPECIMEN_H table) and I_FILE_ WID parameter value is not null, then the stored procedure calls the local procedure

add_file_spec(), which inserts a record into W_EHA_RSLT_FILE_SPEC table. This establishes the link between result file and the specimen used for loading result data.

If the get_specimen_wid stored procedure raises the NO_DATA_FOUND exception (if specimen does not exist in W_EHA_RSLT_SPECIMEN_H table), then the code retrieves the validation procedure name from W_EHA_DATASOURCE table (ODB_UTIL.VALIDATE_CDM_SPECIMEN). This stored procedure retrieves the SPECIMEN_WID from the W_EHA_SPECIMEN_H table against the specimen number and specimen vendor number which might be used as an external data source.

For an external data source, mention the DB link (W_EHA_DATASOURCE). If the procedure does not exist in the CDM schema, then the function logs the error 'Could not find specimen number' in W_EHA_RSLT_LOG table, else it creates a record in both W_EHA_RSLT_SPECIMEN_H and in W_EHA_RSLT_FILE_SPEC table, and returns SPECIMEN_WID to the loader.

This procedure also logs relevant warning, error, and other information.

2.4.1 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, and gene expression must be present in the OHSCE data model. If a file with multiple specimens is loaded, such as a VCF or MAF file, and one of the specimens is not found in the Cohort Explorer datamart schema, then the loader skips that row and loads the rest of the data into the target tables.

2.5 Migrating W_EHA_RSLT_STUDY, W_EHA_SPEC_PATIENT and W_EHA_SPEC_SUBJECT Tables

If the CDM schema data is refreshed using a full ETL load and result files are loaded in ODB, then the ODB CDM reference is broken and the implicit foreign keys in the ODB schema should be updated to point to new CDM record values.

Prior to ODB specimen records being updated, the function indexes that link ODB schema to Patient and Subject records should be rebuilt. Log in as ODB schema and use the following SQL to rebuild these indexes:

```
SQL> alter index W_EHA_SPEC_PATIENT_M1 rebuild;
SQL> alter index W_EHA_SPEC_SUBJECT_M1 rebuild;
```

Following are the SQL statements to update CDM-related tables:

W_EHA_RSLT_STUDY:

```
update w_eha_rslt_study rs set external_study_wid = (
select row_wid from cdm.w_eha_study_d sd
where rs.result_study_name = sd. study_name
);
```

W_EHA_SPEC_PATIENT:

```
update w_eha_spec_patient sp
set
spec_patient_wid = (select sph.row_wid
from
CDM.w_eha_specimen_patient_h_v sph,
w_eha_rslt_specimen rs
where
rs.row_wid = sp.rslt_specimen_wid
and rs.specimen_number = sph.specimen_number
```

```
and nvl(rs.specimen_vendor_number, 'n0Ne' ) = nvl(sph.specimen_vendor_number,
'n0Ne'));
```

W_EHA_SPEC_SUBJECT:

```
update w_eha_spec_subject ss
set
spec_subject_wid = (select sph.row_wid
from
CDM.w_eha_specimen_subject_h_v sph,
w_eha_rslt_specimen rs
where
rs.row_wid = ss.rslt_specimen_wid
and rs.specimen_number = sph.specimen_number
and nvl(rs.specimen_vendor_number, 'n0Ne' ) = nvl(sph.specimen_vendor_number,
'n0Ne'));
```

Also, if the CDM schema is refreshed, ODB aggregates must be rebuilt using the following procedure:

```
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name        => 'RebuildAggregates',
    job_type        => 'PLSQL_BLOCK',
    job_action      => 'BEGIN ODB_VARIANT_AGG_UTIL.rebuild_aggregates ; END;',
    enabled         => true,
    comments        => 'To rebuild aggregates from scratch');
END;
/
```

Note: Since this procedure may take some time to execute, Oracle recommends that you run it as a scheduled job.

2.6 Reference Version Compatibility

Before loading the ENSEMBL version to ODB, ensure its compatibility with other reference and results data.

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

1. GVF data files should belong to versions of Ensembl existing in ODB. For example, if ODB is loaded with Ensembl 66 version, then Oracle recommends that the GVF file to be loaded with the same Version Label should also belong to Ensembl 66. However, a GVF data file belonging to the same genomic alignment can be loaded for any DNA version with the same genomic alignment without losing validity.
2. Variation data files, which include VCF, gVCF, MAF, and CGI masterVar should be based on the same reference genome that was used by the Ensembl version loaded with the same Version Label. For example, if the loaded Ensembl 66 version is using GRCh 37 reference genome, then the results to be loaded should also be mapped based on GRCh 37 version.

Note: If the same reference DNA version source file is loaded more than once, then for each variant, one record is created in the variant table for every source record matched.

3. Copy Number Variation result data should also be checked for reference genome version compatibility with Ensembl version as specified in point 2.
4. TCGA RNASeq exon data should similarly be matched to the correct reference version. TCGA provides a *description.txt* file along with other mage-tab analysis files for every RNAseq dataset. Mapping reference version details are available in this file.

2.7 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 the file to the DOS format.

2.8 Periodically Purge the Recycle Bin

All loaders that use dynamic tables to load data also contain code to drop any temporary tables as well as external tables. These dropped tables are accumulated in Oracle database's recycle bin. Periodically execute the following command to remove these unused objects and free up space:

```
SQL>purge recyclebin;
```

3

Loaders for Reference Data

This chapter contains the following topics:

> **Note:** As the Reference part of the ODB model changes, the Reference Data Loaders have to adapt to these changes. Therefore, both should be updated together.

3.1 Ensembl and SwissProt Loaders

3.1.1 Installing the Loaders

Following are the prerequisites for installing the Ensembl and SwissProt reference loaders:

1. You must have an Oracle database instance with ODB installed in a schema, where the name and password are known. Running the loaders can be simplified by creating an Oracle Wallet with these credentials. Take into consideration that using an Oracle Wallet in Java programs is different from when starting SQL scripts: you require 2 arguments for it—both being OS paths—one to the Oracle Home (the directory where the tnsnames.ora file resides) and the other to the directory where the wallet is created.

> **Note:** Ensembl and SwissProt loaders are written in Java.

> **Note:** Since version 10.2, the JDBC Thin driver has the ability to read a *tnsnames.ora* file using the *oracle.net.tns_admin* property. The JDBC Thin driver, however, cannot access additional TNS content pointed to by an IFILE clause residing inside *tnsnames.ora*. This is a JDBC Thin driver-specific issue and can occur on any platform.
>
> This limitation applies to JDBC 10.2.0.4 and later (Release 10.2 and later).

2. Java Runtime 1.7 or higher must be installed and should be the default on the machine (this can be verified using the `java-version` command from the command prompt).
3. Copy the Reference Loader folder into a directory of your choice. The program should be run from the directory it is installed in. If you are using shell scripts on Unix/Linux, they must be made executable (for example, `chmod +x SwissProt.sh`).

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-74/embl/homo_sapiens`.

   The link above may not reflect the most recent version of the multi-gene files available. Oracle recommends that you use the latest release available from Ensembl. It can be navigated to from the following web page: `http://www.ensembl.org/info/data/ftp/index.html` (select the EMBL format)

   The files are organized by chromosome. There are also some configuration and patch files. At the very least, you must load all chromosome files to cover the entire Human genome. The files are gzipped and can be loaded without being extracting.

   Prior to version 68, Ensembl EMBL files were organized differently. The data was divided into segments, each approximately 100,000 base pairs in length. Starting with version 68, the sequence of an entire chromosome comes in one section. This has presented memory problems for the EMBL loader, which were largely overcome in ODB 3.0, facilitating loading the Human EMBL files using the standard Java Virtual Machine maximum heap allocation of 1 GB. However, this is not guaranteed for other species. So if an out-of-memory error occurs when loading an EMBL file, the load should be repeated with a higher JVM maximum heap allocation, as described below, and may require a computer with Linux or a 64-bit Windows operating system and at least 4 GB of RAM (Oracle recommends 8GB).
2. The SwissProt file (a single file) can be downloaded from `http://www.ebi.ac.uk/uniprot/database/download.html`.
3. Get the (UniProtKB/SwissProt) Flat File.
4. Both EMBL and SwissProt files can be loaded in the compressed form, without extracting the contents, just as they are downloaded. The EMBL and SwissProt Loader can handle GZIP archives (identified by the .gzip extension) as well as some ZIP archives (a ZIP archive must contain only one file to be handled correctly, and is identified by the .zip extension).

3.1.3 Loading the Data

Before you begin:

1. As the Loader runs, it logs some information in the gdm.log file. The file is always appended to and keeps growing. You may occasionally 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. The EMBL/SwissProt loader also logs into the W_EHA_RSLT_LOG table, like the ODB SQL loaders. However, it logs into the database only while connected to it, that is, a connection failure is not logged in the database.

2. The order of loading EMBL and SwissProt files is not important. The scenario outlined below is just an example. Both DNA and Protein reference data are now versioned. Versions stored in the W_EHA_VERSION table are used. For DNA sources and all reference data that are linked to them (genes and so on) the version must be of type DNA. For Protein Info records and all the reference data linked to them the version must be of type PROTEIN. Starting from version 3.0, a Protein record is loaded only if there is no identical record already loaded with another version. Each part of a SwissProt file is date-stamped with a release date, and this is used to determine if a record in a new file has changed from an already loaded version or not.

   The EMBL Loader accepts a version argument, which must match an existing version of type DNA. If the version is not found, there is a prompt that lets you use the provided version label. On confirmation, the version is created. If no version argument is provided, the loader enters the full interactive mode, letting you select an existing version from a list or creating a new one. The same is true for the SwissProt loader, except that the version is of type PROTEIN.

   Version labels are not case-sensitive and stored in uppercase.

   Note: If the version label is not provided or the provided version does not exist, the EMBL or SwissProt loader prompts for input and does not continue until you respond. Therefore, to run the loader in the non-interactive mode, it is necessary to create the version of the correct type beforehand and ensure it is passed to the loader correctly. It is also possible to prevent the loader from waiting for the user input in the case of an incorrect version being passed to it, by appending `>outlog <empty.txt` to the command line and creating an empty text file empty.txt. In this case the loader fails if the version does not exist.

Perform the following steps to load files using Linux or Windows shell scripts. There is also a native JAVA command-line interface available described in Section 3.1.4, "Running the Embl/Swissprot Loader with Named Command-Line Arguments".

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 (or SwissProt.sh on Linux). When SwissProt.bat is executed, you can optionally specify the Species List file. The purpose of the Species List file is to permit only loading protein information for the organism(s) you want.

   The format of the file is simple - type in the species primary (Latin) name(s), one species per line. A file for just the human genome is included in the distribution — 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 are loaded (which takes much longer).

If an Oracle Wallet is set up, SwissProt.bat/.sh can use the credentials stored in the Wallet to connect to the schema else it prompts for a password. If an Oracle Wallet is set up, pass the following parameters to run SwissProt.bat/.sh:

a. Username - when an 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. Directory location of the tnsnames.ora file

e. Path to Wallet

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

g. Complete path and name of the data file

h. Reference version of type 'PROTEIN' to use (omit to enter the interactive mode):

Following is an example of how the SwissProt.bat is to be run if an Oracle Wallet is set up and a Species list file is not present or not used:

```
C:\>swissProt.bat "" DB001Wallet trc_gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets ""
SwissProt.dat "VERSIONP1"
```

Example using SwissProt.sh

```
> sh SwissProt.sh "" DB001Wallet trc_gdm
"/app/oracle/product/11.2.0.2.0/network/admin" "/app/wallet/" ""
SwissProt.dat "VERSIONP1"
```

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 "VERSIONP1"
```

Example using SwissProt.sh

```
> sh SwissProt.sh "" DB001Wallet trc_gdm
"/app/oracle/product/11.2.0.2.0/network/admin" "/app/wallet/"
Species.dat SwissProt.dat "VERSIONP1"
```

If no Oracle Wallet is set up, you 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, or scan-server-name:port:SID for a multiple node DB). For example, Localhost:1613:devdb1

c. Schema name

d. Directory location of the tnsnames.ora file. 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 the species list file. This is an optional parameter. If you do not have this list, enter ""

g. Complete path and name of the data file

h. Optional version of type 'PROTEIN' to use (omit to enter the interactive mode)

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 "VERSIONP1"
```

Example using SwissProt.sh

```
>sh SwissProt.sh trc_gdm localhost:1613:devdb1 trc_gdm "" "" ""
SwissProt.dat "VERSIONP1"
```

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

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

Example using SwissProt.sh

```
>sh SwissProt.sh trc_gdm localhost:1613:devdb1 trc_gdm "" ""
Species.dat SwissProt.dat "VERSIONP1"
```

The SwissProt Loader can also be run with named command-line arguments.

2. The Ensembl EMBL files can be loaded next. Multiple EMBL files can 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). Multiple EMBL files can be loaded concurrently (for example, in separate terminal windows). However, to create a new DNA reference version using the interactive mode, one file should be loaded first (creating the new version in the process) and only then can multiple files be loaded concurrently.

If an Oracle Wallet is set up, embl.bat/.sh can use the credentials stored in the Wallet to connect to the schema else it prompts for a password. If Oracle Wallet is set up, pass the following parameters when Embl.bat is run:

a. Username — when an 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. Directory location of the tnsnames.ora file

e. Path to Wallet

f. Complete path and name of the data file

g. Depending on the file being loaded and on the operating system, a seventh, optional, argument may need to be used: the Java Virtual Machine heap size, in MB.

By default, embl.bat (and the corresponding UNIX shell script, embl.sh) specifies 1 GB of Java Virtual Machine (JVM) maximum heap space (using the –Xmx1024M option). This is known to be sufficient for loading all Human Ensembl files, version 68, and works on most Linux or Windows systems

(Oracle recommends 8 GB of RAM or more). Hence, you do not need to provide the heap size argument.

However, some Ensembl EMBL files for other species, version 68 or higher, may require more heap space. The seventh argument should then be provided as 2048 (that is 2048 MB). These files can then be loaded successfully on Linux or 64-bit Windows with enough RAM. If there is an `Out-of-memory` error while loading a file, use a larger maximum heap size and use the same option for all subsequent Ensembl files for the same organism.

If the specific computer or operating system cannot handle the specified JVM heap size, a *Java Virtual Machine creation failed* error occurs. You will then have to use another machine.

h. The version of type DNA can be passed as the eighth argument (omit to enter the interactive mode).

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
2560
```

Example for the embl.sh shell script for Linux:

```
>sh embl.sh "" DB001Wallet trc_gdm
/app/ora11g/product/11.2.0/dbhome_2/NETWORK/ADMIN /app/wallets
embl.dat 2048 "GRCH37.P8"
```

If no Oracle Wallet is set up, 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. Directory location of the tnsnames.ora file. 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

g. Optional maximum heap size (in MB), or ""

h. Optional version of type 'DNA' (omit to enter the interactive mode to select or create a version)

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
2048 "GRCH37.P8"
```

Example for embl.sh:

```
>sh embl.sh trc_gdm localhost:1613:devdb1 trc_gdm "" "" embl.dat
2048 "GRCH37.P8"
```

The EMBL Loader can also be run with named command-line arguments.

3.1.4 Running the Embl/Swissprot Loader with Named Command-Line Arguments

The EMBL/SwissProt Loader is a single Java application, packaged into a JAR archive GDM.jar. It can be used without any shell or Windows script, using the arguments

described in this section. Running the application using these arguments provides some additional capabilities, not supported by the shell or Windows scripts installed with it.

Java Runtime 1.7 is required to run the EMBL/SwissProt Loader. It has to be either in the PATH environment variable, or the full path to the Java executable with version 1.7 should be specified.

If the Java 1.7 executable is in the PATH environment variable, the EMBL/SwissProt loader is run as follows:

```
java -Xmx2048m -jar gdm.jar <argument 1>…<argument N>
```

The *-XmxNNNNm* Java option is optional. In the 3.0 (or later) version of the EMBL/SwissProt Loader, it is usually not required because the memory usage has been optimized to handle chromosome-wide Ensembl files. Use it if you encounter an `Out-of-memory` error for any Ensembl EMBL file (provided that the machine has enough memory and the operating system is Linux or 64-bit Windows).

The arguments (except for the path or name of the data file to load, which must always be the last argument) are not positional, and can be used in any order. The key or option arguments begin with the "-" character. Some of them require a value as the following argument, others are stand-alone. For the up-to-date list of all available arguments, execute the following command:

```
java -jar gdm.jar -help
```

Following is more detailed information about the arguments and their usage:

`-url <url>` - specifies the URL of the database to be connected. <*url*> must be in the form - *host*:*port*:*instance*. This argument is mandatory.

`-schema <schema name>` - specifies the ODB schema name. This argument is mandatory.

`-user <user>` - specifies the user name used to log into the database (needed only when not using an Oracle Wallet).

`-wallet <wallet>` - specifies the directory where the Oracle Wallet is set up.

`-orahome <oracle home directory>` - specifies the Oracle home directory, when a Wallet is used (this is the directory where the tnsnames.ora file resides).

`-sprot` - this key is used to load a SwissProt file. If it is absent, the data file will be loaded as an EMBL file.

`-protFile <species file>` - specifies the file path (optional) and name of a Species List file, used to optionally filter the contents of a SwissProt file by species (used when loading SwissProt only).

`-version <version label>` - specifies the version label of the DNA or Protein reference version. If this argument is present, it must match an existing version of appropriate type (DNA for EMBL, PROTEIN for SwissProt). If it is omitted, the Loader will start in the interactive mode, prompting you to select an existing version, or create a new one.

`-verbose` - if this argument is present, additional information will be printed on the screen and logged.

`-updateDB` - this argument is necessary if you want to actually load the contents of the file into the database. If it is omitted, the contents of the data file are parsed, but nothing is inserted into the database tables (except the version, if the Loader is started in the interactive mode and you choose to create a new version). Omitting this option is useful for verifying that the file parses without errors.

-print_summary - if this key is present, summary info will be printed on the console at the end of the run.

Examples:

Loading an EMBL file without a Wallet, Windows:

```
java -jar GDM.jar -url localhost:1521:b41804x1 -schema odb -user odb
-version V1 -updateDB EMBLFile.dat
```

Verifying (without loading) the same file:

```
java -jar GDM.jar -url localhost:1521:b41804x1 -schema odb -user odb
-version V1 EMBLFile.dat
```

Loading a SwissProt file with a Wallet, Windows:

```
java -jar GDM.jar -url localhost:1521:b41804x1 -schema odb -orahome
C:\ora11g\product\11.2.0\dbhome\NETWORK\ADMIN -wallet C:\Wallets -version
P1 -updateDB -sprot -protFile species.dat SPFile.dat
```

Loading an EBML file with a Wallet, Linux, interactive mode for version:

```
java -jar GDM.jar -url localhost:1521:b41804x1 -schema odb -orahome
/home/apps/ora11g/product/11.2.0/dbhome/NETWORK/ADMIN -wallet
/home/Wallets -updateDB EMBLFile.dat
```

3.1.5 Index-Organized Tables Loader

The Java (EMBL/SwissProt) loader uses staging tables to prevent blocking when multiple sessions are loading concurrently. EMBL and Swissprot linked XREF and QLFR data tables are index referenced. A stored procedure, ODB_REFERENCE_UTIL.create_referecne_iot is called after all Ensembl and SwissProt data has been loaded (this builds the XREF and QLFR tables more efficiently).

After loading the EMBL and SwissProt reference files, call the shell script IOT_loader.sh, which in-turn executes load_from_stage_to_iot.sql, which calls the stored procedure.

Following is the command line options for the loader script:

```
Sh IOT_loader.sh <options>
```

Options

(*) required

-db_wallet* <VARCHAR2> (required, unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

-db_conn* <VARCHAR2> (required if -db_wallet is not provided)

SID, or the Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_
DATA=(SID=XE)))"
```

-db_user* <VARCHAR2> (required if -db_conn is provided)

ODB user name for the Database connection.

-check_version <NUMBER>

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

Example

```
sh ./IOT_loader.sh -db_wallet TRCQC
```

3.1.6 Gathering Optimizer Statistics

Oracle recommends gathering table and index statistics after completing an Ensembl and (or) SwissProt data load. Missing or stale statistics can profoundly deteriorate query performance. Oracle statistics is a collection of data about database objects such as tables and indexes. Oracle optimizer requires you to estimate the most efficient query execution plan.

To collect statistics, connect to a database as ODB_SCHEMA owner using SqlPlus and execute the following command:

```
exec dbms_stats.gather_schema_stats ('ODB_', cascade=>true,estimate_
percent=>dbms_stats.auto_sample_size);
```

3.2 HUGO Loader

3.2.1 Description and Files to Load

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 of the complete HGNC dataset, which can be retrieved from their Statistics and Downloads Webpage here http://www.genenames.org/cgi-bin/hgnc_stats. You must specifically download the complete HGNC dataset by clicking the hyperlink in the sentence *Click here for the complete HGNC dataset* provided in the above webpage. The data downloaded is a large file with tabular text and tab-delimited values, given with column headers.

Important: The format of the complete HGNC files has changed as of May 2013. The ODB HUGO Loader 3.0 only supports the new file format, while the previous versions of the loader (ODB 2.0.2.1 and prior) only support the old format.

A batch file for Windows and an alternative shell script for Linux-bash, have been provided for loading the data.

3.2.2 Running the Loader

The loader is found bundled in the latest ODB build in the /ODB_Loaders/Reference_Loader/Hugo_loader directory. This folder contains 8 files:

- hugo_loader.bat
- hugo_loader.sh
- hugo_script.sql
- several common sh, bat, and SQL scripts for reference version checking

To run the loader, perform the following:

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 hugo_loader.bat and (or) hugo_loader.sh file resides.
3. If working on Linux, ensure the scripts are executable (you may need to run `chmod u+x *.sh`)
4. The hugo_loader.sh/.bat scripts use the credentials stored in an Oracle Wallet to connect to the schema that has the ODB. Pass the following parameters when the hugo_loader.bat is run:
   a. The Hugo data file.
   b. Oracle Directory Object
   c. Wallet name
   d. Operation without an Oracle Wallet (with user name and database connection arguments) is only supported on Linux.
5. Execute hugo_loader.sh/.bat, with appropriate arguments, to load the data.

3.2.3 Command-Line Argument List

Synopsis

hugo_loader.sh -help

hugo_loader.sh <...options>

Description

Description:

Validates input options and calls the loader script hugo_script.sql#hugo_loader.load_hugo

Options

(*) required

`-db_wallet* <VARCHAR2>` (required, unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

SID, or the Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"
```

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version* <NUMBER>`

Run check version (1=yes | 0=no)

`-check_version_non_i* <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no)

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table location. Refer to the programmer's guide on how to retrieve this file from HGNC's web portal.

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table directory. For details, see Section 2.1, "Setting Up a Directory Object"

`-reference_version <VARCHAR2>`

The reference version label of the Hugo file being loaded. A "HUGO" reference version label is defined in W_EHA_VERSION.VERSION_LABEL. If the version label is not present in the W_EHA_VERSION table, the loader prompts the user to confirm whether to continue with the version label provided. If yes, the loader inserts the new record in the version table with the given Version_label and proceeds with the load.

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
$ sh hugo_loader.sh -db_wallet odb_user -check_version 1 -check_version_
non_i 0 -data_file "genefam_list.pl" -data_directory "ODB_LOAD"
-reference_version "DLD_DT_01062013" -read_size ""
```

Windows

```
C:\> hugo_loader.bat -db_wallet odb_user -check_version 1 -check_version_
non_i 0 -data_file "genefam_list.pl" -data_directory "ODB_LOAD"
-reference_version "DLD_DT_01062013" -read_size ""
```

Once loading is complete, log into SQL developer, or SQL*Plus, with ODB Schema and verify that 35000 or more records are populated in W_EHA_HUGO_INFO table. The execution information is logged in the W_EHA_RSLT_LOG table. If run with the `-print_summary 1` option, the loader will also print information about the execution on the console, including the count of inserted records and errors, if any.

3.3 GVF Ensembl Loader

3.3.1 Description and Files to Load

The GVF Ensembl loader is responsible for the input of known variants for any given species for which DNA source records are present. It loads 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). Therefore, you must ensure that 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. Therefore, the W_EHA_SPECIES table should 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, GVF input files can be downloaded from the Ensembl FTP website `ftp://ftp.ensembl.org/pub/release-65/variation/gvf/homo_sapiens/`.

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

Note: Ensembl now keeps Germ-line mutations and Somatic mutations in separate gvf files. To have both datasets, Oracle recommends loading both the Homo_sapiens.gvf.gz and the Homo_sapiens_somatic.gvf.gz files present in the link.

3.3.2 Running the Loader

The .bat and .sh files for GVF loaders require the same set of named command line arguments, except that the .bat script only supports using an Oracle Wallet connection (like for all other SQL loaders).

3.3.3 Command-Line Argument List

Name

GVF_loader.sh - load records

Synopsis

GVF_loader.sh -help

GVF_loader.sh <...options>

Description

Validates input options and calls the loader script load_gvf.sql#odb_ref_gvf_util.process_gvf

Options

(*) required

`-db_wallet* <VARCHAR2>` (required, unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle SID, or the Oracle connection string that is,

"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"

-db_user* <VARCHAR2> (required if -db_conn is provided)

ODB user name for the Database connection.

-check_version <NUMBER>

Run check version (1=yes|0=no) [default: 0]

-check_version_non_i <NUMBER>

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

-log_level <VARCHAR2>

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

-print_summary <NUMBER>

Print summary (1=yes|0=no) [default: 0]

-data_file* <VARCHAR2>

Data file name - Oracle external table LOCATION

-data_directory* <VARCHAR2>

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

-species_name* <VARCHAR2>

Species name defined in W_EHA_SPECIES that is, for humans "Homo sapiens"

-reference_version <VARCHAR2>

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

-preprocess_dir <VARCHAR2>

Preprocess directory - Oracle external table PREPROCESSOR

-preprocess_file <VARCHAR2>

Preprocess file - Oracle external table PREPROCESSOR

-read_size <NUMBER>

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
$ sh GVF_loader.sh -db_wallet odb_user -data_file "som_variants.gvf"
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -reference_version
"GRCh37.p8" -preprocess_dir "" -preprocess_file "" -read_size ""
```

Windows

```
C:\> GVF_loader.bat -db_wallet odb_user -data_file "som_variants.gvf"
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -reference_version
"GRCh37.p8" -preprocess_dir "" -preprocess_file "" -read_size ""
```

3.3.4 Gathering Optimizer Statistics

Oracle recommends gathering table and index statistics after running the GVF loader. Oracle statistics is a collection of data about database objects such as tables and indexes. It is required by Oracle optimizer to estimate the most efficient query execution plan. Missing or stale statistics can profoundly deteriorate query performance.

To collect statistics, connect to a database as ODB_SCHEMA owner using sqlplus and execute the following command:

```
exec dbms_stats.gather_schema_stats ('ODB_', cascade=>true,estimate_
percent=>dbms_stats.auto_sample_size);
```

3.4 Pathway Loader

3.4.1 Description and Files to Load

Pathway_loader is a script for extracting, transforming, and loading GSEA standard file formats.

The data used can be downloaded from http://www.pathwaycommons.org/archives/PC1/last_release-2011/gsea/by_species/homo-sapiens-9606-gene-symbol.gmt.zip.

Currently, this is the last supported pathway file format. More recent versions exist but should be reformatted to the above supported version.

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

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

3.4.2 Running the Loader

The loader is made up of 10 files:

- pathway_loader.bat
- pathway_loader.sh
- pathway_script.sql
- several common sh, bat, and SQL scripts for reference version checking

The execution call of the stored procedure load_pathway() is designed in one of the script files (pathway_script.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTORY OBJECT, SPECIES_NAME, PATHWAY REFERENCE VERSION and READ_SIZE as input parameters.

It creates an external table dynamically and uploads data from the source file into it. External tables let Oracle query data that is stored outside the database in flat files. The ORACLE_LOADER driver accesses data stored in any format that can be loaded by the SQL*Loader.

The stored procedure dynamically creates PATH_DATA_!!SEQ!! as an external table. This external table stores the complete pathway data. This table maps all the fields existing in the pathway file.

> **Note:** In the above external table, the "!!SEQ!!" string is replaced by ETL_PROC_ID at run time.

There are two bulk insert statements executed dynamically. One SQL inserts the record into the W_EHA_PATHWAY table and the other inserts the record into the W_EHA_PATHWAY_PROTEIN table.

Multiple versions of pathway data can be loaded into the table. The VERSION_WID column saves the version ID of the particular version loaded, which is retrieved from the W_EHA_VERSION table.

The pathway_loader.bat file requires the logon credentials to be stored in an Oracle Wallet, while the pathway_loader.sh script can be run with or without an Oracle Wallet.

The load_pathway procedure supports error logging associated with Pathway Reference, species_name, sql_err and Species lookup failure. These errors are logged into the W_EHA_RSLT_LOG table.

3.4.3 Command-Line Argument List

Name

pathway_loader.sh - load records

Synopsis

pathway_loader.sh -help

pathway_loader.sh <...options>

Description

Validates input options and calls the loader script pathway_script.sql#pathway_loader.load_pathway

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

SID or Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection. `-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-reference_version <VARCHAR2>`

"PATHWAY" reference version label defined in W_EHA_VERSION.VERSION_LABEL. If the version label is not present in the W_EHA_VERSION table, the loader interactively prompts for confirmation to continue with the version label provided. If yes, the loader inserts the new record in the version table with the given Version_label and proceeds with the load.

`-read_size <NUMBER>`

Read size in bytes. Oracle external table READSIZE

Examples

UNIX

```
$ sh pathway_loader.sh -db_wallet odb_user -data_file
"homo-sapiens-9606-gene-symbol.gmt" -data_directory "ODB_LOAD" -species_
name "Homo sapiens" -reference_version "feb-2011" -read_size ""
```

Windows

```
C:\> pathway_loader.bat -db_wallet odb_user -data_file
"homo-sapiens-9606-gene-symbol.gmt" -data_directory "ODB_LOAD" -species_
name "Homo sapiens" -reference_version "feb-2011" -read_size ""
```

3.5 Prediction Score (PolyPhen, SIFT) Loader

3.5.1 Description and Files to Load

In Ensembl, human mutations affecting the amino acid substitutions are further analyzed for the effect of this substitution on protein function. This is done using SIFT and PolyPhen predictive algorithms. The source files from running either SIFT or PolyPhen contain prediction and score which is stored in the target tables. The model supports multiple versions of SIFT and PolyPhen data. The versions are recorded in the W_EHA_VERSION and W_EHA_FILE_TYPE tables.

Downloading Data from Ensembl BioMart

The source data is downloaded from the Ensembl BioMart tool. Download SIFT and POLYPHEN data separately and load them in separate loader runs.

The data can be downloaded from one of the following links:

http://uswest.ensembl.org/biomart/martview/

http://asia.ensembl.org/biomart/martview/

1. Select Database as Ensembl Variation <*ver*>.
2. Select Dataset as either Homo sapiens Somatic Short Variants (COSMIC sourced) or Homo Sapiens Short Variants (dbSNP sourced).
3. From Filters: If you want to download data for a specific region of the chromosome, then use this option. Otherwise you can retain the default filter options.
4. From Attributes: Select following options in the *specific order defined below*:
   - From SEQUENCE VARIATION:
     * Variation Name
     * Chromosome Name
     * Position on Chromosome (bp)
     * Strand
     * Variant Allele
   - From GENE ASSOCIATED INFORMATION:
     * Consequence specific allele
     * Ensembl Transcript ID
     * Polyphen prediction or SIFT prediction
     * Polyphen score or SIFT score.
5. Click **Result** at the top of the screen.
6. Export all results to select following:
   1. Select File
   2. Select TSV
   3. Select Unique results only.
7. Click **Go** to download the file.

Select the attributes in the specified order. Also, ensure that you select PolyPhen prediction and PolyPhen score when downloading PolyPhen data, and SIFT prediction and SIFT score when downloading SIFT data.

For downloading large SIFT and PolyPhen datasets, you can alternatively use the Ensembl Perl APIs. For details, refer the following links:

http://www.ensembl.org/info/docs/index.html

http://www.ensembl.org/info/docs/api/variation/index.html#api

The execution call of the stored procedure `odb_result_util.process_variant_prediction ()` is designed in one of the script files (load_prediction_score.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTORY OBJECT, FILE TYPE

(being either SIFT or Polyphen), FILE VERSION and DNA REFERENCE VERSION as an input parameters.

It creates an external table dynamically and uploads data from the source file into it. External tables let Oracle query data that is stored outside the database in flat files. The ORACLE_LOADER driver accesses 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 stored procedure dynamically creates PREDICTION_DATA_!!SEQ!! as an external table. This external table stores the complete result data. This table maps all the fields existing in the input file.

Note: In the above external table, the "!!SEQ!!" string is replaced by ETL_PROC_ID at run time.

There are two multi-table insert statements executed dynamically. One inserts records into the w_eha_variant_prediction table and another inserts records into the w_eha_rslt_log table.

A `select` statement that parses the data from the external table uses an inline query, which gets the dataset of variant and transcript records. An inline query uses a partition (analytical function) function to avoid duplicate records for different variation names with the same variant record (VARIANT_WID). The dataset of this inline query will then be joined with the W_EHA_VARIANT_PREDICTION table to look up VARIANT_WID and STRUCTURE_WID, and will insert a record into either the W_EHA_VARIANT_PREDICTION table or the W_EHA_RSLT_LOG table.

If you upload the same file (that is, a file with the same name as the one previously loaded) with a different version, the loader considers this file as a new file and uploads the record into the target (W_EHA_VARIANT_PREDICTION) table with a different file version.

3.5.2 Running the Loader

A record is inserted in the W_EHA_VARIANT_PREDICTION table for a variant belonging to a specific transcript; hence reference_id and transcript_id is looked up before inserting a record in this table.

Before inserting a record in this table, the loader also checks if a record already exists in the W_EHA_VARIANT_PREDICTION table for a specific reference_id and transcript_id and for the version of SIFT or PolyPhen data.

The following operation is performed based on the above condition:

- If reference_id (variant_wid) and transcript_id (structure_wid) exist in the target table but SIFT or polyphen version is different, a new record is inserted in this table.
- If reference_id (variant_wid) and transcript_id (structure_wid) exist in the target table and SIFT or polyphen version are also the same, the score is compared.
- If the score is same, the variant record is not updated.
- If the score is not same, the existing record is not updated but the reference_id along with transcript_id is reported to the W_EHA_RSLT_LOG table stating that the score was different.

- If reference_id (variant_wid) and transcript_id (structure_wid) do not exist in the table, a new record is inserted.

> **Note:** The batch file requires an Oracle Wallet to be set up to run correctly. However, the shell script can be run with or without an Oracle Wallet.

SIFT or PolyPhen Reference Version

The Program Version refers to the SIFT or PolyPhen program version used to generate the data. You can check the version from http://asia.ensembl.org/info/genome/variation/predicted_data.html

Table 3–1 Mapping of Polyphen or SIFT File

Column Name in Result File	Table and Column Name in ODB	Description
Variation Name	REFERENCE_ID in W_EHA_VARIANT_XREF table	This is a FK to W_EHA_VARIANT table. It is extracted by the loader using a lookup of Variation Name from the source file against the REFERENCE_ID in the W_EHA_VARIANT_XREF table.
Chromosome Name	W_EHA_CHROMOSOME .CHROMOSOME	Chromosome Name
Position on Chromosome (bp)	W_EHA_RSLT_SEQUENCING.START_POSITION	Stores the start position on chromosome.
Strand	N/A	N/A
Variant Alleles	N/A	N/A
Ensembl Transcript ID	W_EHA_GENE_STRUCTURE. TRANSCRIPT_ID	This is a FK to W_EHA_GENE_STRUCTURE table. It is extracted by the loader using a lookup of Ensembl Transcript ID from the source file against the 'TRANSCRIPT_ID' in W_EHA_GENE_STRUCTURE table.
PolyPhen /SIFT prediction	W_EHA_PREDICTION_CODE .CODE	This is a FK to W_EHA_CODE table, which stores all possible predictions for SIFT and PolyPhen data. The FK corresponding to the prediction of this record is available in the source file.
PolyPhen or SIFT score	W_EHA_VARIANT_PREDICTION. PREDICTION_SCORE	Stores the prediction score value from the source file for a particular variant.

3.5.3 Command-Line Argument List

Name

prediction_score_loader.sh - load records

Synopsis

prediction_score_loader.sh -help

prediction_score_loader.sh <...options>

Description

Validates input options and calls the loader script load_prediction_score.sql#odb_ref_prediction_util.process_variant_prediction

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-prediction_version_type* <VARCHAR2>`

Prediction reference version type (SIFT | POLYPHEN)

`-prediction_version_label* <VARCHAR2>`

"SIFT" | "Polyphen" reference version label defined in W_EHA_VERSION.VERSION_LABEL. If the version label is not present in the W_EHA_VERSION table, the loader

prompts for confirmation to continue with the version label provided. If yes, the loader inserts the new record in the version table with the given Version_label and proceeds with the load.

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
$ sh prediction_score_loader.sh -db_wallet odb_user -data_file "ut_
variant_pred1.txt" -data_directory "ODB_LOAD" -species_name "Homo sapiens"
-prediction_version_type "Polyphen" -prediction_version_label "5.0"
-reference_version "GRCh37.p8" -preprocess_dir "" -preprocess_file ""
-read_size ""
```

Windows

```
C:\> prediction_score_loader.bat -db_wallet odb_user -data_file "ut_
variant_pred1.txt" -data_directory "ODB_LOAD" -species_name "Homo sapiens"
-prediction_version_type "Polyphen" -prediction_version_label "5.0"
-reference_version "GRCh37.p8" -preprocess_dir "" -preprocess_file ""
-read_size ""
```

Typical Errors Associated with prediction_score Loader

- *Record not inserted* is logged if the same version and same file type is loaded but the input record inserted has a score different than the existing record.
- Other possible errors are:
  - *Generating etl process id*
  - *Generating enterprise id*
  - *Verifying species name*
  - *Verifying reference version*
  - *Verifying prediction version*
  - *Inserting file type*
  - *Verifying file type*
  - *Processing result records*

3.6 Probe Loader

3.6.1 Description and Files to Load

The probe loader populates 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 mostly a reference loader, but it varies with vendors, for example, Affymetrix, Illumina.

Following are the assumptions for the data file for the Probe Loader:

- The file is tab-delimited.
- The first row is always the header.

Mappings for Probe Loader

Table 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

3.6.2 Running the Loader

The execution call of the stored procedure PROBE_LOADER() is in the script file probe_script.sql. This stored procedure accepts FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, DNA VERSION LABEL, PROBE VERSION LABEL and READ_SIZE as input parameters.

It creates an external table dynamically and uploads data from the source file into it. External tables let Oracle query data that is stored outside the database in flat files. The ORACLE_LOADER driver accesses data stored in any format that can be loaded by the SQL*Loader.

The stored procedure dynamically creates PROBE_DATA_!!SEQ!! as an external table. This external table stores the complete probe data and maps all the fields existing in the probe file.

There is a `merge` statement that dynamically either inserts or updates the existing probe record in w_eha_probe table. While updating a record, it updates all the columns including the row_wid with a new row_wid for a particular probe name.

Old records can be referenced using the Flashback Data Archive (FDA) approach, which is used for securely tracking the contextual history of all data. FDA makes it possible to automatically and transparently track all changes to tables in the database and easily query data in those tables as of any point in time or over any interval within the specified retention period, with minimal performance impact.

To load gene expression result files for a particular older reference version, run the result loader *BEFORE* re-running the probe loader, updating the probes in the table to a new reference version. Then load the next set of gene expression result files pointing to the new version. The gene expression records will lose their probe reference keys as probe row_wids are updated.

The .bat file requires an Oracle Wallet to be set up before it can run successfully.

For Shell scripts, if an Oracle Wallet is set up, the shell script uses those credentials to run the Sqlplus. If an Oracle Wallet is not set up, the script prompts for a password and connects to Sqlplus.

3.6.3 Command-Line Argument List

Name

probe_loader.sh - load records

Synopsis

probe_loader.sh -help

probe_loader.sh <...options>

Description

Validates input options and calls the loader script probe_script.sql#probe_loader.load_probe

Options

(*) required

`-db_wallet* <VARCHAR2>` (required, unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string, that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-dna_version_label* <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-probe_version_label* <VARCHAR2>`

"PROBE" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
$ sh probe_loader.sh -db_wallet odb_user -data_file "dummy_probeset_
annotation_summary.txt" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -dna_version_label "GRCh37.p8" -probe_version_label "PROBE_VER_1"
-read_size ""
```

Windows

```
C:\> probe_loader.bat -db_wallet odb_user -data_file "dummy_probeset_
annotation_summary.txt" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -dna_version_label "GRCh37.p8" -probe_version_label "PROBE_VER_1"
-read_size ""
```

3.7 ADF Data Loader

3.7.1 Description and Files to Load

An Array Description Format (ADF) file, or an array design file, is a microarray platform-specific, tab-delimited file that describes the design of an array. For a particular array platform, this file lists out the features (spots) found on an array, along with its location and associated annotation information including the Reporters (oligo probes) found at that feature and the Composite Elements (genomic features such as genes) represented by it.

The ADF Data Loader loads the ADF file for AgilentG402A_07_1 (Agilent 244K Custom Gene Expression G4502A-07-1) platform, which contains annotation data for all Gene Composite elements found in AgilentG402A_07 Level-3 data files present in TCGA and are loaded using the Dual Channel Loader into ODB.

TCGA provides all available ADF files on its Platform Design page:
https://tcga-data.nci.nih.gov/tcga/tcgaPlatformDesign.jsp

The TCGA ADF file for AgilentG4502A_07_1 can be retrieved from the following link:
http://tcga-data.nci.nih.gov/docs/integration/adfs/tcga/AgilentG4502A_07_01.tcga.adf.zip.

3.7.2 Running the Loader

The execution call of the stored procedure odb_result_util.process_adf() is designed in the script file load_adf.sql. This stored procedure accepts FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES, USER LABEL, Reference Version, File Flag, Preprocess directory, Preprocess File, Data File Path, DBFS Store, Alternate file location (ftp location or http location), Read Size as input parameters.

It creates an external table dynamically and uploads data from the source file into it. External tables let Oracle query data that is stored outside the database in flat files. The ORACLE_LOADER driver accesses 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 stored procedure dynamically creates ADF_DATA_!!SEQ!! as an external table. This external table stores the complete result data. This table maps all the fields existing in the result file.

The procedure first loads a row into the w_eha_adf table. A simple `merge` command is written to lookup against user label, file WID, version WID, etl proc WID and enterprise WID dataset. All the values are passed as input parameters. If the dataset matches any of the w_eha_adf records, then the loader updates all the columns (except w_update_dt) of w_eha_adf table for corresponding user_label of the table.

The three multi-table `insert` statements executed dynamically insert records into the w_eha_adf_composite table, the w_eha_adf_reporter, and the w_eha_adf_reporter_coord tables.

The deleted records can be referenced using the Flashback Data Archive (FDA) approach, which is used for securely tracking the contextual history of all data. FDA makes it possible to automatically and transparently track all of the changes to the tables in the database, and to easily query data in those tables as of any point in time or over any interval within the specified retention period, with minimal performance impact

Note: The batch file requires an Oracle Wallet to be set up to run correctly. The shell script can be run with or without an Oracle Wallet.

3.7.3 Command-Line Argument List

Name

ADF_loader.sh - load records

Synopsis

ADF_loader.sh -help

ADF_loader.sh <...options>

Description

Validates input options and calls the loader script load_adf.sql, which calls odb_ref_adf_util.process_adf

Options

(*) required

`-db_wallet* <VARCHAR2>` (required, unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string, that is,

"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"

-db_user* <VARCHAR2> (required if -db_conn is provided)

ODB username for the database connection.

-check_version <NUMBER>

Run check version (1=yes | 0=no) [default: 0]

-check_version_non_i <NUMBER>

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

-log_level <VARCHAR2>

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

-print_summary <NUMBER>

Print summary (1=yes | 0=no) [default: 0]

-data_file* <VARCHAR2>

Data file name - Oracle external table LOCATION

-data_directory* <VARCHAR2>

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

-species_name* <VARCHAR2>

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

-user_label* <VARCHAR2>

User label used to identify a composite record's source ADF dataset that is, AgilentG4502A_07_1

-reference_version <VARCHAR2>

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

-file_flg* <CHAR>

File flag (E=external | S=copy to secure data file directory) [default: E]

-preprocess_dir <VARCHAR2>

Preprocess directory - Oracle external table PREPROCESSOR

-preprocess_file <VARCHAR2>

Preprocess file - Oracle external table PREPROCESSOR

-data_file_path <VARCHAR2> (required, if -file_flg is "S")

File system path to secure data file directory

-dbfs_store <VARCHAR2> (required, if -file_flg is "S")

Database file system store

-alt_file_loc <VARCHAR2>

Alternate file location link that is, ftp:location, http:location

-read_size <NUMBER>

Read size in bytes - Oracle external table READSIZE

```
-data_file_dir <VARCHAR2>
```

File system path to Oracle directory object

Examples

UNIX

```
$ sh ADF_loader.sh -db_wallet odb_user -data_file "adf_summary.adf" -data_
directory "ODB_LOAD" -species_name "Homo sapiens" -user_label
"AgilentG4502A_07_01" -reference_version "GRCh37.p8" -file_flg "E"
-preprocess_dir "" -preprocess_file "" -data_file_path "" -dbfs_store ""
-alt_file_loc "" -read_size ""
```

Windows

```
C:\> ADF_loader.bat -db_wallet odb_user -data_file "adf_summary.adf"
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -user_label
"AgilentG4502A_07_01" -reference_version "GRCh37.p8" -file_flg "E"
-preprocess_dir "" -preprocess_file "" -data_file_path "" -dbfs_store ""
-alt_file_loc "" -read_size ""
```

3.8 HGMD (BioBase) Loader

3.8.1 Description and Files to Load

The HGMD loader is used to load mutations and associated disease, drug and other annotations from GFF formatted source files that can be obtained from BioBase Biological Databases as a part of the Genome Trax™ data set. Files can be loaded from the BioBase site at the following location:

https://portal.biobase-international.com/download/genometrax/

A valid Genome Trax license is required for the download. Download only the gff archive for the desired reference genome assembly. The loader currently loads 3 data source file types:

- hgmd_hg*.gff
- hgmd_disease_hg*.gff
- drug_hg*.gff

> **Note:** BioBase provides scheduled updates as a full set of curated data assembled against two most recent reference genome builds.

The BioBase release version of these data sets is not available in data files and has to be provided as a command-line argument (see -hgmd_version_label below). Currently the HGMD datasets are built against Human Genome 18 (HG18) and Human Genome 19 (HG19) in UCSC notation (or Build 36 and Build 37 in NCBI notation).

You can load both HG18 and HG19 curated data, but they have to be linked to the correct reference genome versions that exist in ODB. Loading a new HGMD release version will overwrite all the linkage data previously loaded for the same reference genome version. The overwritten records can be referenced using the Flashback Data Archive (FDA) approach.

The file processing starts with the hgmd_hg*.gff file representing the inherited mutations track. All variants listed in the source file are identified and the novel ones are added to the W_EHA_VARIANT table. A reference to the HGMD accession for all variants from the source file is inserted into the W_EHA_VARIANT_XREF table. New disease names are added to the W_EHA_DISEASE table. Finally, all curated associations are loaded into W_EHA_DISEASE_G_VARIANT table and literature links are added to the W_EHA_DISEASE_G_VAR_XREF table.

The second source file hgmd_disease_hg*.gff that represents gene or disease linkage is loaded into w_eha_disease_gene and w_eha_disease_gene_xref tables. The latter table stores literature links.

Lastly, the drug_hg*.gff file is processed to populate W_EHA_DRUG, W_EHA_DRUG_XREF, W_EHA_DRUG_TARGET and W_EHA_DRUG_TARGET_XREF tables. The first two tables store drug records and drug references respectively. Currently drugs are referenced by the DruBank IDs (http://www.drugbank.ca/). Two other tables store gene or drug linkage and references to supporting research findings.

3.8.2 Running the Loader

The loader is implemented as a PL/SQL stored procedure that can be invoked from a provided shell or batch file. The procedure accepts an Oracle Directory Object, file suffix, species name, reference genome version, BioBase release version and read size as input parameters.

The file suffix—the hg* portion of the input file name—reflects the version of the reference genome assembly (HG18 or HG19) that is used to prepare the source reference files. The loader parameter called *reference genome version* specifies which reference genome release loaded into ODB schema should be used to link HGMD annotations. Therefore the file suffix and the target reference genome version should be within the same major release.

For example, HG19 files can be linked with GRCH37.P8 or GRCH37.P9 reference genome but not Build 36.

The loader does not require specific filenames to process. Given the file suffix, it loads all currently supported files.

Note: The batch file requires an Oracle Wallet to be set up to run correctly. The shell script can be run with or without an Oracle Wallet.

3.8.3 Command-Line Argument List

Name

HGMD_loader.sh - load records

Synopsis

HGMD_loader.sh -help

HGMD_loader.sh <...options>

Description

Validates input options and calls the loader script load_hgmd.sql#odb_ref_hgmd_util.process_hgmd

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_
DATA=(SID=XE)))"
```

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_directory* <VARCHAR2>`

Data file suffix name - Oracle external table LOCATION

`-data_file_suffix* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-dna_version_label* <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-hgmd_version_label* <VARCHAR2>`

"HGMD" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
$ sh HGMD_loader.sh -db_wallet odb_user -data_file_suffix "hg19_12" -data_
directory "ODB_LOAD" -species_name "Homo sapiens" -dna_version_label
"GRCh37.p8" -hgmd_version_label "2012.4" -read_size ""
```

Windows

```
C:\> HGMD_loader.bat -db_wallet odb_user -data_file_suffix "hg19_12"
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -dna_version_label
"GRCh37.p8" -hgmd_version_label "2012.4" -read_size ""
```

3.9 COSMIC Loader

3.9.1 Description and Files to Load

The COSMIC loader is used to load information on cancer causing somatic mutations such as sample data, histology, anatomical site observed, the genomic location and the publication IDs from where the data is curated. The dataset is available at the following official Sanger website:

http://cancer.sanger.ac.uk/cancergenome/projects/cosmic/.

Cosmic has changed its license model and the versions after v71 are not free to download. Oracle supports loading COSMIC data for v71 versions or before.

For details on downloading the COSMIC dataset, visit the following location

http://cancer.sanger.ac.uk/cosmic/download

The loader takes 4 files of two types as input, that must be retrieved from the Sanger's ftp file download repository from separate file paths.

- Two coding and non-coding annotation data files with the following file name patterns:
  - **CosmicCompleteExport_vxx_xxxxxx.tsv.gz**: contains annotations on somatic variants the affect coding regions of the genome.
  - **CosmicNCV_vxx_xxxxxx.csv.gz**: contains available annotations on variants found in non-coding regions of the genome.
- Two VCF format variant genomic feature files with the following file name patterns:
  - **CosmicCodingMuts_vxx_xxxxxx_noLimit.vcf.gz**: contains genomic feature information for coding variants.
  - **CosmicNonCodingVariants_vxx_xxxxxx_noLimit.vcf.gz**: contains genomic feature information for non coding variants.

You must decompress all files before running the loader.

3.9.2 Running the Loader

The loader is implemented as a PL/SQL stored procedure that could be invoked from a provided shell or batch file. The procedure accepts all four of the input file Oracle Directory Object, species name, reference genome version, COSMIC release version and read size as input parameters.

The VCF files from COSMIC that map Mutation identifiers with the genomic position or sequence variation list the reference genome version in the header portion. For example, `##reference=GRCh37`.

When you load VCF files from COSMIC—(the COSMIC loader invokes the VCF loader but you must provide a reference genome version). You can link them to any reference genome patch for this reference genome version.

All Comic versions should be linked to GRCH37 based DNA versions such as GRCH37.P8 or GRCH37.P7. If your analysis involves both p8 and p7, run the loader twice and link the same files to both patches.

The cosmic loader first calls process_gvcf to process and load reference variants found in the 2 vcf format files into the variant tables. Thus, any new variants are added to ODB. After the variant load, the data files are processed by the loader procedure which loads sample, histology, anatomical site to ODB and bridges the data to variant and gene information.

Note: The batch file requires an Oracle Wallet to be set up to run correctly. The shell script can be run with or without an Oracle Wallet.

3.9.3 Command-Line Argument List

Name

COSMIC_loader.sh - load records from COSMIC files

Synopsis

COSMIC_loader.sh -help

COSMIC_loader.sh <...options>

Description

Validates input options and calls the loader script load_cosmic.sql#odb_ref_cosmic_util.process_cosmic

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

`-data_directory* <VARCHAR2>`

Data file suffix name - Oracle external table LOCATION

`-data_file_coding* <VARCHAR2>`

Coding Mutations Data file name - Oracle external table LOCATION

`-data_file_noncoding* <VARCHAR2>`

Non Coding Mutations Data file name - Oracle external table LOCATION

`-vcf_file_coding* <VARCHAR2>`

Coding region Varaint's VCF file name - Oracle external table LOCATION

`-vcf_file_noncoding* <VARCHAR2>`

Non Coding region Variant's VCF file name - Oracle external table LOCATION

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, for humans "Homo sapiens"

`-dna_version_label* <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-cosmic_version_label* <VARCHAR2>`

"COSMIC" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

Examples

UNIX

```
sh COSMIC_loader.sh -db_wallet slc04lx3 -data_file_coding "CosmicCompleteExport_
v67_241013.tsv" -data_file_noncoding "CosmicNCV_v67_241013.tsv" -vcf_file_coding
"CosmicCodingMuts_v67_20131024.vcf" -vcf_file_noncoding "CosmicNonCodingVariants_
v67_20131024.vcf" -data_directory "ODB_LOAD" -species_name "Homo sapiens" -dna_
version_label "GRCH37.P8" -cosmic_version_label "COSMIC.V67"
```

Windows

```
C:\> COSMIC_loader.bat -db_wallet slc04lx3 -data_file_coding
"CosmicCompleteExport_v67_241013.tsv" -data_file_noncoding "CosmicNCV_v67_
241013.tsv" -vcf_file_coding "CosmicCodingMuts_v67_20131024.vcf" -vcf_file_
noncoding "CosmicNonCodingVariants_v67_20131024.vcf" -data_directory "ODB_LOAD"
-species_name "Homo sapiens" -dna_version_label "GRCH37.P8" -cosmic_version_label
"COSMIC.V67"
```

3.10 Variant Effect Job

The variant effect job is responsible for loading the impact of the presence of a genomic variant on any feature transcript it falls on or is close to. The variant effect links a variant to a gene transcript component, such as a coding exon segment, the protein coded and the genomic source sequence and then calculates the following:

- NET EFFECT caused by the presence of the mutation on the target transcript.
- The gene level GENE EFFECT on the base nucleotides.
- The CODING Region EFFECT, which stores the specific region and the nucleotide change occurrence, as calculated from the start site of the transcript.

- PROTEIN_EFFECT that notes the location on and change to the Amino Acid translation sequence of the coding transcript.

By default, the variant effect job is disabled, and when enabled, is set to run once every 24 hours by the DBMS Scheduler. When the job completes a run, the procedure called by the job, odb_var_effect_util.process_var_effect, will load impact data for any new variants added to W_EHA_VARIANT.

The procedure can also be run once and immediately, by adding a new job to the scheduler without a time specification.

Execute the following command creates a single run job:

```
BEGIN
  -- Job defined entirely by the CREATE JOB procedure.
  DBMS_SCHEDULER.create_job (
    job_name        => 'VARIANT_EFFECT_JOB3',
    job_type        => 'PLSQL_BLOCK',
    job_action      => 'BEGIN odb_var_effect_util.process_var_effect; END;',
                         enabled         => true,
    comments        => 'Test Job.');
END;
/
```

3.11 Typical Errors Associated with Reference Loaders

3.11.1 Loader Runtime Error: ORA-01460 Unimplemented or Unreasonable Conversion Requested

Errors have been observed while running various ODB loaders. The loader run aborts prematurely with the following error message: *ORA-01460 unimplemented or unreasonable conversion requested*.

Oracle recommends applying an RDBMS patch to the TRC database that fixes this bug. See Oracle Support Bug 13099577 (ORA-1460 WHEN PARALLEL QUERY SERVERS ARE USED) available here https://mosemp.us.oracle.com/epmos/faces/BugDisplay?id=13099577 for details.

4

Loaders for Result Data

This chapter includes the following topics:

4.1 Prerequisites

Before using the result loaders, ensure that at least one version of reference Ensembl files has been loaded using the Java loader.

> **Note:** The reference loaded has to match the reference used for alignment of result files.

Also, ensure that Oracle optimizer statistics were gathered after the reference data was loaded.

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

1. Create a record in the W_EHA_RSLT_STUDY table. There is a sequence (W_EHA_RSLT_STUDY_S) associated with this table that lets you create as many study records as required. The result data can be partitioned by study or by gene, depending on how the ODB schema was created. Add the required number of studies to this table. The result loaders use the value for RESULT_STUDY_NAME to look up the corresponding study primary key.

2. Verify that the W_EHA_DATASOURCE record, which identifies the CDM schema to validate specimen numbers, is correct. Each result record that is to be loaded must reference a specimen that exists in the CDM schema. Patient related specimens should be in the W_EHA_SPECIMEN_PATIENT_H bridge table and subject related ones in the W_EHA_SPECIMEN_SUBJECT_H table. These CDM schema bridge tables have a SPECIMEN_VENDOR_NUMBER field to be used for vendor specific information and a SPECIMEN_NUMBER field to a universal specimen identifier. The W_EHA_DATASOURCE table has a DB_LINK_NAME field that ensures that a database link can be used if the CDM schema is in another instance. The table stores the CDM-lookup procedure function call in the field VALIDATION_PROC. Four seed data entries are provided in this table, each with the following DATASOURCE_CD that the result loaders can choose from as an input parameter:
   - 'CDM': which is the default datasource with the VALIDATION_PROC value as 'ODB_UTIL.VALIDATE_CDM_PATIENT_SPEC', that does a patient specific CDM specimen look up.
   - 'CDM_PATIENT': has the VALIDATION_PROC value as 'ODB_UTIL.VALIDATE_CDM_PATIENT_SPEC', that does a patient specific CDM specimen look up.
   - 'CDM_SUBJECT': has the VALIDATION_PROC value as 'ODB_UTIL.VALIDATE_CDM_SUBJECT_SPEC', that does a subject specific CDM specimen look up.
   - 'CDM_BOTH': has the VALIDATION_PROC value as 'ODB_UTIL.VALIDATE_CDM_BOTH_SPEC', that internally calls both ODB_UTIL.VALIDATE_CDM_PATIENT_SPEC and ODB_UTIL.VALIDATE_CDM_SUBJECT_SPEC functions, to lookup and link to both subject and patient specimens.
3. Ensure that the ODB schema has SELECT privileges on the W_EHA_SPECIMEN_PATIENT_H_V and W_EHA_SPECIMEN_SUBJECT_H_V views in the CDM schema.
4. All the specimens required for the example files should be added into the W_EHA_SPECIMEN_PATIENT_H table for patient related specimens and in W_EHA_SPECIMEN_SUBJECT_H for subject related specimens in the CDM schema.

To install the loaders, copy the Result_Loader folder into a directory. Run a loader from the directory it is installed in (on Linux this requires an execute permission for all SH scripts, which should be granted after the files are copied (for example, using the Linux `chmod` command).

4.1.1 Setting Default Cache Sizes for Result Loading

Each of the result tables have a corresponding sequence, named similar to the table with a suffix *_S*. Each of these result table sequences has a default cache size that reflects an average number of records that might be inserted for any load of result files.

Most of the sequences have a default cache size of 6000, whereas all of the sequencing related result table sequences have a cache size of 15000. There may be a need to load much larger files (that is, TCGA VCF data can have 4.5 million rows). For larger files, Oracle recommends that a DBA adjusts all the corresponding sequence cache sizes to at least 100,000 or larger.

Lower sequence cache sizes can result in delays for each parallel process trying to get the next cache of sequences. The actual decision to increase sequence cache size should be based on the number of rows estimated to be inserted during any load.

An example of the SQL to alter a sequence is:

```
alter sequence w_eha_rslt_gene_exp_s cache 100000;
```

The current list of sequences used by relevant result loaders are as follows:

1. CGI loader
   - ODB_RSLT_CGI_UTIL (default cache 15000)
     - W_EHA_RSLT_SEQUENCING1_S
     - W_EHA_RSLT_NOCALL1_S
     - W_EHA_RSLT_NON_VARIANT_S
2. VCF loader
   - ODB_RSLT_GVCF_UTIL (default cache 15000)
     - W_EHA_RSLT_CONFLICT_S
     - W_EHA_RSLT_NOCALL1_S
     - W_EHA_RSLT_NON_VARIANT_S
     - W_EHA_RSLT_NOCALL1_S
     - W_EHA_RSLT_SEQUENCING1_S
     - W_EHA_RSLT_STRUCT_VAR_S
     - W_EHA_RSLT_SV_BREAKEND_S
3. MAF Loader
   - ODB_RSLT_MAF_UTIL (default cache 15000)
     - W_EHA_RSLT_SEQUENCING1_S
4. RNA-seq loader
   - ODB_RSLT_RNA_SEQ_UTIL (default cache 6000)
     - W_EHA_RSLT_RNA_SEQ_S
5. CNV loader
   - ODB_RSLT_CNV_UTIL (default cache 6000)
     - W_EHA_RSLT_COPY_NBR_VAR_S
6. Single channel loader
   - ODB_RSLT_SINGLE_CHANNEL_UTIL (default cache 6000)
     - W_EHA_RSLT_GENE_EXP_S
7. Dual channel loader
   - ODB_RSLT_DUAL_CHANNEL_UTIL (default cache 6000)
     - W_EHA_RSLT_2CHANNEL_GXP_S
8. QC metadata loader
   - ODB_RSLT_METADATA_UTIL (default cache 6000)
     - W_EHA_RSLT_SPEC_QLFR_S

4.2 Overview of Result Loaders

Following are the result loaders provided, one for each file type:

- Gene Expression - Single channel
- Gene Expression - Dual channel
- Copy Number Variation - SEG format
- MAF - MAF format
- VCF - VCF and gVCF formats
- CGI - CGI MasterVar format
- RNA-seq - TCGA RNA-seq exon files

Additionally, there are:

- Probe loader - a prerequisite to load probes before single channel gene expression loader runs.
- ADF loader - a prerequisite to load gene composite and reporter annotation before dual channel gene expression loader runs.
- QC metadata - loads Quality Control metadata files for specimens in ODB.
- File Specimen Loader - links specimen in ODB to files, including existing input files, derived files or any other file related to result data. Also, enables loading file details into ODB if they are not already present.
- File Lineage Linker- Lets the user link any 2 files in ODB to each other, thus enabling building a file lineage between source files and their derivatives.

All loaders can be run using .bat files in Windows or shell scripts in Linux.

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

- single_channel_gene_expr_loader.bat
- dual_channel_gene_expr_loader.bat
- MAF_loader.bat
- VCF_loader.bat
- CGI_masterVar_loader.bat
- CNV_loader.bat
- TCGA_RNA_SEQ_loader.bat
- METADATA_loader.bat
- File_specimen_linker.bat
- File_lineage_linker.bat

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

If an Oracle Wallet is set up, a shell script can use the wallet credentials to run Sqlplus.

If Oracle Wallet is not set up, the shell script prompts for a password and connects to Sqlplus.

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

- single_channel_gene_expr_loader.sh
- dual_channel_gene_expr_loader.sh
- MAF_loader.sh
- VCF_loader.sh
- CGI_masterVar_loader.sh
- TCGA_RNA_SEQ_loader.sh
- CNV_loader.sh
- METADATA_loader.sh
- File_specimen_linker.sh
- File_lineage_linker.sh

4.3 Version Information Utility

The version information utility checks for all available versions in the database instance for any of version types allowed in W_EHA_VERSION table. It is a command line API which:

- lists specific versions loaded for a single version type when a version type is specified through -list_ver, OR
- lists all versions present, grouped by version type, when version type is not specified.

4.3.1 Functional Description

The version info utility is a standalone script version_info.sh (version_info.bat for Windows**)** with an optional named argument **'-list_ver'**, to which a reference version type argument is passed. The loader calls the **odb_reference_util.list_version_info** function, accepting the version type argument as a parameter. The function queries W_EHA_VERSION table for the VERSION_LABEL values where column VERSION_TYPE is the value of input parameter accepted by the function. These version labels are listed to the user on the command line. As mentioned in the previous section, if the user does not pass the parameter, the function displays the list of all versions for each version type.

4.3.2 Running the Version Check Utility

Name

version_info.sh - Lists Version Labels

Synopsis

version_info.sh -help

version_info.sh <...options>

Description

Validates input options and calls the loader script list_version.sql# odb_reference_util.list_version_info

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_
DATA=(SID=XE)))"
```

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

-list_ver <VARCHAR2>

A Reference Version type allowed for W_EHA_VERSION.VERSION_TYPE column. Known Version types are:
'DNA','PROTEIN','HUGO','PATHWAY','SIFT','POLYPHEN','PROBE', 'GENETIC_CODE', 'COSMIC', and 'BIOBASE'

4.4 CGI masterVar Data Loader

The CGI masterVar file is an integrated report of variant calls and annotations with each file representing variants per sample. The current CGI masterVar loader supports 4 versions of CGI Format which includes 2.0, 2.2, 2.4 and 2.5.

Since there is a large difference of number of columns between the CGI format versions, the loader is built to handle this change and load the data accordingly.

The CGI 2.0 file format is described here:

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

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

- 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 while 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 mentioned while executing the batch file. The loader also parses the #FORMAT information, which contains the CGI format version details, and maps it to CGI file type in W_EHA_FILE_TYPE table.

4.4.1 Functional Description of CGI Loader

The CGI loader currently loads variant records as well as the wild type (WT) information. For example,

- If one of the alleles is WT while the other is variant, then the loader only records the variant allele.

- If both the alleles are WT then the loader creates a single record in W_EHA_RSLT_NON_VARIANT table.
- If both alleles are variants and homozygous, then it stores only one record. In such cases it stores the least score value in SCORE_VAF and SCORE_EAF columns.
- If both alleles are variant and heterozygous, then it stores data as two separate records.
- If a record is nocall and both alleles have the same value then a single record is created in W_EHA_RSLT_NOCALL table.
- If a record has one allele as variant and another as a nocall, then the loader report creates one record in variant and one record in nocall table.
- A haploid record contains only one allele information and if it is a variant then it is reported as a variant record.
- If a haploid record is wildtype, then it is reported in W_EHA_RSLT_NON_VARIANT table.

The loader takes the chromosome and position details of a record from CGI masterVar file and checks if the corresponding region exists in W_EHA_DNA_SOURCE table. If it is present, the loader maps this record of W_EHA_DNA_SOURCE table as W_EHA_VARIANT.SOURCE_WID. If the region does not exist, the loader ignores that record and logs information in the W_EHA_RSLT_LOG table indicating that some records were not loaded. The loader does not validate for invalid chromosome number or positions details. If it encounters such invalid data, the loader ignores that record and does not log it to W_EHA_RSLT_ERR_LOG table.

The loader does not validate the accuracy of the reference nucleotides in the database. It assumes that the same version of reference mapped CGI masterVar data is loaded in to ODB. Ensure that the reference version matches the results file being loaded and the reference data available in the ODB.

4.4.2 Files to Load

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

4.4.3 Data Load

The loader creates 2 external tables. One table stores the metadata and the other stores the actual result data.

The loader first creates an external table to store the metadata of the result file. This metadata resides in the header part of the result file, which starts with ## string. This external table then populates the W_EHA_RSLT_FILE_QLF table. The W_EHA_RSLT_FILE_QLF table is simply a name value pair table.

Most of the file metadata is in XML format where an identifier or tag is followed by an attribute value or XML definition. The metadata load will set the QUALIFIER_TAG to the identifier before the = character (that is, FORMAT, INFO, FILTER) and everything after the = is copied to the QUALIFIER_VALUE column of W_EHA_RSLT_FILE_QLF table.

The SQL to create the data external table can exceed 32K, so a cursor is used to create the external table. The constant string used is broken up into 5 separate strings. This

allows for more than 32K statement to create the external table. The structure of the external table depends on the version of file type.

The loader first processes the reference data. A `select` statement which inserts data into W_EHA_VARIANT_STG, computes the overlap value comparing reference with the allele sequence. Using this overlap value, the reference sequence and the allele sequence is shortened and the start position and end position are incremented. Also, this overlap value creates a replace tag with shortened reference and allele sequence.

After inserting the record into the W_EHA_VARIANT_STG table, a PROCESS_ VARIANT() procedure is called that populates the W_EHA_VARIANT table. This procedure also populates the W_EHA_VARIANT_LINK_STG table if it exists, else it creates and populates a new _link table. Another new feature is added to this procedure to populate the new table W_EHA_VARIANT_GENE_MAP. This table stores the link between variant and gene_wid. To populates this table, a `select` query is written which maps the gene_wid of W_EHA_GENE_SEGMENT with variant record using the BEGIN and END position of the result file data.

The loader then process to parse the first set of result data which does not use W_ EHA_ VARIANT foreign key. This includes W_EHA_RSLT_NOCALL, W_EHA_RSLT_ NON_ VARIANT tables.

The loader then process to parse to link all records to W_EHA_VARIANT table. This includes W_EHA_RSLT_SEQUENCING.

Specimen identifiers are stored in the CGI data file header. There are several values used to obtain 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. The loader will verify the tag `#SAMPLE` in the header part of the result file. This is usually 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. There is also a specimen vendor number used to look up the correct SPECIMEN record. The last value required is used to specify if multiple sources are used to provide SPECIMEN records. Each result table stores a FK to the correct datasource for the SPECIMEN and the Primary Key value for the SPECIMEN record. The loading code must use all three fields to find the correct values for the result records.

The alleles identified as no-call create a W_EHA_RSLT_NOCALL record. All other non-ref alleles create a W_EHA_RSLT_SEQUENCING. The query used to create sequencing records maps with w_eha_variant_gene_map table to compute the gene_ wid for sequencing record against variant wid of both w_eha_rslt_sequencing and w_ eha_variant_gene_m ap table. All wildtype records with zygosity of *hom* or *hap* and varitype of *ref* are loaded in the W_EHA_RSLT_NON_VARIANT table.

To compute the gene_wid for nocall and non-variant records, a temporary intermediate table is created, which calculates the relative position of the records. This table is then used to map against start and end position of w_eha_nocall or w_eha_ non_variant table for nocall and non-variant records.

Note:

- The batch file requires Oracle Wallet to be set up to run correctly.
- In Linux, you do not need to use Homo sapiens in "". This requirement is only for Windows.

Table 4–1 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.CHROMOSOME_WID W_EHA_VARIANT.CHROMOSOME W_EHA_RSLT_NON_ VARIANT.CHROMOSOME_WID	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, 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 the notation combining reference and allele sequences. For NOVEL variants, a new record is created in the 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 W_EHA_RSLT_NON_ VARIANT.START_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 in the ABSOLUTE_POSITION column in the W_ EHA_VARIANT table. For W_EHA_VARIANT.START_POSITION, Start_Position is relative to the value in the W_EHA_DNA_SOURCE.START_POSITION table using the begin value.
End	W_EHA_RSLT_NOCALL.END_ POSITION W_EHA_VARIANT_STG.END_ POSITION W_EHA_RSLT_NON_VARIANT. END_POSITION	This value is used for no-call results and stored in the END_POSITION field. This value calculates 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 W_EHA_RSLT_NON_VARIANT. 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 4–1 (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 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 are incremented. This overlap value creates 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. In some cases, this field is empty for insertions.
allele1Seq	W_EHA_VARIANT.REPLACE_TAG W_EHA_VARIANT_X.ALLELE	For sequencing results, this value constructs the replace tag. In some cases, this field is empty for deletions.
allele2Seq	W_EHA_VARIANT.REPLACE_TAG W_EHA_VARIANT_X.ALLELE	For sequencing results, this value constructs the replace tag. In some cases, this field is empty for deletions.
allele1VarScoreVAF	W_EHA_RSLT_ SEQUENCING.SCORE_VAF W_EHA_RSLT_NON_VARIANT. SCORE_VAF	Used for sequencing results and stored in the SCORE_VAF field. If the variant is homozygous, as only single allele record is stored, the least value out of two scores is stored.
Allele2VarScoreVAF	W_EHA_RSLT_ SEQUENCING.SCORE_VAF W_EHA_RSLT_NON_VARIANT. SCORE_VAF	Used for sequencing results and stored in the SCORE_VAF field. If the variant is homozygous, as only single allele record is stored, the least 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, as only single allele record is stored, the least 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, as only single allele record is stored, 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.

Table 4–1 (Cont.) Mapping of CGI Result File

Column Name	Table and Column Name in ODB	Description
allele1ReadCount	W_EHA_RSLT_SEQUENCING. ALLELE_READ_COUNT	Used for sequencing results and stored in the ALLELE_READ_COUNT field.
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 W_EHA_RSLT_NON_VARIANT. REFERENCE_READ_COUNT	Used for sequencing results and stored in the REFERENCE_READ_COUNT field.
totalReadCount	W_EHA_RSLT_SEQUENCING. TOTAL_READ_COUNT W_EHA_RSLT_NON_VARIANT. TOTAL_READ_COUNT	Used for sequencing and non-variant results and stored in the TOTAL_READ_COUNT field.

4.4.4 Running the CGI Loader with Named Command-Line Arguments

Name

CGI_masterVar_loader.sh - load records

Synopsis

CGI_masterVar_loader.sh -help

CGI_masterVar_loader.sh <...options>

Description

Validates input options and calls the loader script load_cgi.sql#odb_rslt_cgi_util.process_cgi

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_
DATA=(SID=XE)))"
```

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, for humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-alt_file_loc <VARCHAR2>`

Alternate file location link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

`-parallel_degree <NUMBER>`

The degree of parallelism, or number of parallel execution servers associated with the load operation.

4.4.5 Examples

UNIX:

```
$ CGI_masterVar_loader.sh -db_wallet odb_user s04jsnx1 -check_version 0
-check_version_non_i 1 -data_file 'summary_masterVarBeta_2.4_format.tsv'
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -study_name
"STUDY1" -datasource_name "CDM_PATIENT" -specimen_vendor "vendor2"
-reference_version "GRCh37.p8" -file_flg "E" -parallel_degree 8
```

Windows:

```
C:\> CGI_masterVar_loader.bat -db_wallet odb_user s04jsnx1 -check_version
0 -check_version_non_i 1 -data_file 'summary_masterVarBeta_2.4_format.tsv'
-data_directory "ODB_LOAD" -species_name "Homo sapiens" -study_name
"STUDY1" -datasource_name "CDM_PATIENT" -specimen_vendor "vendor2"
-reference_version "GRCh37.p8" -file_flg "E" -parallel_degree 8
```

4.5 VCF Sequence Data Loader

4.5.1 Functional Description

The VCF loader procures the chromosome, position and reference version details of a record from a VCF file and checks if the corresponding region of that chromosome exists in W_EHA_DNA_SOURCE table for the specific reference version given as input to the loader. If present, it maps this record of W_EHA_DNA_SOURCE table as W_EHA_VARIANT.SOURCE_WID. If the region is not found (for example, if the chromosome name and (or) the position details are invalid), the loader ignores the record and does not log into W_EHA_RSLT_LOG table.

The loader supports two types of chromosome representation in the VCF file, like chr10 and also 10 would be loaded without any error. For mitochrondrial chromosome the loader can read chrM, chrMT, M and MT from the file.

A typical VCF file contains data for multiple specimens. If one or more specimen values does not exist in the CDM schema, then the data for these particular specimens is not loaded but is logged in W_EHA_RSLT_LOG.

The loader does not validate the accuracy of the reference nucleotides in the database. It assumes that the same version of reference mapped VCF data is loaded to ODB. Therefore, ensure that the reference version of the results file being loaded matches that of the reference data available in the ODB. ODB now supports multiple reference versions, so the VCF loader must be instructed as to which reference data it has to map the results. You must provide the version information, present in the VERSION_ LABEL column of the W_EHA_VERSION table, to the VCF loader as a parameter. Refer the loader parameter list for more details.

The VCF loader has been extended from the existing support of the 1000 Genomes VCF 4.1 format which includes mutations such as SNV, small indel to large structural variants and structural re-arrangements from the 1000 genomes VCF 4.1. The updated VCF loader also supports the gVCF (genome VCF) data from Illumina, version 20120906a.

Details of the 1000 Genomes VCF 4.1 specification can be found at the following link:

http://www.1000genomes.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-41

Details of the gVCF specification can be found at the following link:

https://sites.google.com/site/gvcftools/home/about-gvcf

Following is a brief description of each format supported by the VCF loader.

4.5.1.1 1000 genomes VCF4.1 Version

The 1000 genomes VCF 4.1 format can be broadly classified into 3 categories based on the type of variants given below:

1. **SNV and small indel**: These mutations are loaded in W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_NOCALL and W_EHA_VARIANT table. The W_EHA_VARIANT table is only populated for novel variants not already present in that table.

   The loader populates W_EHA_RSLT_NOCALL table based on the GT values having './.'. Consecutive records with this type of nocall genotype will be collapsed while loading to this table. For example, if there are three records with POS as 1001, 1002 and 1003 with nocall genotype, then only one record is created in W_EHA_RSLT_NOCALL table with the START_POSITION 1001 and the END_POSITION 1003.

2. **Large Structural Variation**: These large structural changes in the genome are recorded in the W_EHA_RSLT_SEQUENCING, W_EHA_VARIANT and W_EHA_VARIANT_X tables. The W_EHA_VARIANT and W_EHA_VARIANT_X tables are only populated for novel variants, not present in that table. W_EHA_VARIANT_X table is used to populate only the ALT column value from the VCF file in ALLELE clob column of this table.

3. **Structural rearrangement**: Currently, 1000 genomes data for structural rearrangements is not yet released. There is neither detailed documentation nor proper examples in the 1000 genomes manual, which would cover all scenarios for this data set. In view of this, the loader is built on the following assumptions, considered from the small amount of information available in the 1000 genomes VCF 4.1 manual.

   a. The loader assumes that there is GT information in the form of either '0' or '1'.

   b. The loader identifies structural rearrangements from the tag 'SVTYPE=BND' present in the INFO column.

   c. Allele depth (AD) and total depth (DP) are expected to be a single value by the loader.

   These genomic re-arrangements are stored in W_EHA_RSLT_SV_BREAKEND table.

   Since a VCF 4.1 file can contain all the above three types of mutations in the same file, the VCF loader automatically distinguishes these three types of data using the INFO column and records are created in their respective result tables as described above.

   - If the INFO column of the VCF file does not have an 'SVTYPE' tag or has 'SVTYPE' other than 'BND' (for example, 'SVTYPE=DEL'), then this mutation is considered as either SNV, small indel or large SV and data is loaded to the W_EHA_RSLT_SEQUENCING table.
   - If the INFO column has 'SVTYPE=BND', then this mutation is considered as a structural rearrangement and records are loaded to the W_EHA_RSLT_SV_BREAKEND table.

4.5.1.2 Genome Variant Call Format (gVCF)

gVCF is designed to store both variant and non-variant information related to single sample only. gVCF follows the 1000 genomes VCF 4.1 conventions, with additional features such as siteConflicts. The VCF loader processes gVCF data into W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_NON_VARIANT, W_EHA_RSLT_CONFLICT, W_EHA_RSLT_NOCALL and W_EHA_VARIANT. The W_EHA_VARIANT table is only populated for novel variants not present in that table. The mutations are stored in W_EHA_RSLT_SEQUENCNG, the non-variant information is stored in W_EHA_RSLT_NON_VARIANT table, the conflict variants are stored in W_EHA_RSLT_CONFLICT, and nocall information is stored in the W_EHA_RSLT_NOCALL table. Nocall data for the consecutive positions is not collapsed in gVCF load because it is already compressed in the gVCF file based on similar quality scores and other parameters defined while creating the gVCF file.

The following logic is used while populating gVCF records in target tables:

- Any gVCF file records with GT values 0/1 or 1/0 or 1/2 or n/n, where n is not zero are stored in the W_EHA_RSLT_SEQUENCING table.
- Any records with GT values 0/0 or just 0 are stored in the W_EHA_RSLT_NON_VARIANT table.
- Any records with GT values '.' or './.' and ALT value '.' are stored in the W_EHA_RSLT_NOCALL table.
- Any records with GT values '.' and ALT value not '.' are stored in the W_EHA_RSLT_ CONFLICT table.

4.5.1.3 FILE_TYPE_CODE and LOAD_MODE of VCF Loader

As mentioned earlier, the VCF loader can be used to load all types of VCF data, that is, SNP and small indel, large structural variation, structural rearrangement, and gVCF data. Since there is just one loader for loading all the data types, it has to be provided with some information to identify the file type. There is one additional parameter which can be used to load data in a specific mode as described below.

There are mainly two parameters required by the VCF loader which determine the input file type and the mode in which to load this data.

These parameters are:

- **FILE_TYPE_CODE**: The 'FILE_TYPE_CODE' parameter is used to provide the file type information. While loading a VCF file containing either SNP and small indel or large SV or SV-rearrangements, give FILE_TYPE_CODE as 'VCF'. While loading a gVCF file, FILE_TYPE_CODE should be given as 'GVCF'.
- **LOAD_MODE**: The VCF loader has options to load VCF and gVCF data in three different modes using the parameter 'load_mode'. Following are the three types of modes identified by the loader:
  - **'VCF' mode**: This mode can be used for both VCF and gVCF file types. This mode loads mutations and nocall data and only populates the W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_NOCALL tables. If FILE_TYPE_CODE is GVCF and LOAD_MODE is VCF, then all non-variant and conflict records from the gVCF file are skipped and only variants and nocalls are loaded.
  - **'GVCF' mode**: This mode loads data to all tables made for gVCF, that is, W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_NON_VARIANT, W_EHA_RSLT_NOCALL and W_EHA_RSLT_CONFLICT tables. Since the gVCF file has all the information about a genome such as variants, non-variants, nocalls and conflicts, this mode is best suited for the gVCF file type. However, this

mode can also be used to load data from a VCF file with all non-variant information for a specific genome. Usually, a VCF file doesn't contain all the non-variant information of a genome and only shows few records as non-variant for a specimen when there is a mutation at that position for a different specimen. It is advisable not to load such files using GVCF mode because incomplete non-variant information will be loaded for that specimen.

– **'NON-VAR' mode**: This mode loads data only into the W_EHA_RSLT_NON_VARIANT and W_EHA_RSLT_NOCALL tables. It is designed for scenarios where mutations have already been loaded from a VCF file and only non-variant information is to be loaded. Like the GVCF mode, this mode is also mostly suited for the gVCF file type as the gVCF file contains all non-variant information. It is not advisable to load a VCF file which does not contain all the non-variant information for a specimen in NON-VAR mode, as incomplete non-variant information will be stored in the database.

4.5.2 Custom Format Specification in VCF

The VCF loader also supports loading custom data types from the FORMAT column of a VCF file. Following are the details on loading custom formats to ODB.

The custom format option helps load certain VCF FORMAT column fields which are currently not mapped in ODB. Before executing the loader, you must manually create a column in W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SV_BREAKEND tables. For Exadata, the staging tables like W_EHA_STG_SEQUENCING and W_EHA_STG_SV_BREAKEND should also be appended with the additional column in the same order as defined in the main result tables.

The column names should follow a specific naming convention. To map a 'PL' data type from the FORMAT column in the input file, create a column with the field name 'CUST_PL'. Then provide the mapping details to the loader under 'custom_format' parameter for the loader as "PL=CUST_PL". To load multiple custom format columns, provide the values as a comma separated string, for example, "PL=CUST_PL,GL=CUST_GL".

Although there is now no limitation on the number of custom formats supported by the loader, it can read only 32 format data types at a time. So if a custom format data type is beyond the 32 data type then it will not be loaded. The order of the custom columns created should be same in the main tables and in the corresponding staging tables, otherwise there could be a mismatch in the data loaded. It is recommended to add a custom column with a VARCHAR2(%n) data type as there could be comma separated values and other alphanumeric characters in the field. The %n should be defined based on the string requirement of the FORMAT data type for which the column is created.

Alternatively, set these custom format field mappings as a global variable in VCF_FORMAT column of the W_EHA_PRODUCT_PROFILE table. This has to be set manually by the user in this table. The `-custom_format` input parameter of the VCF_Loader, which is a temporary variable has precedence over the globally defined variable in W_EHA_PRODUCT_PROFILE.VCF_FORMAT column while loading VCF data. The format specification in VCF_FORMAT column is the same as used for -custom_format input parameter.

The VCF export functionality of the Cohort Explorer UI also requires that the custom format details be defined in the global variable, that is, W_EHA_PRODUCT_PROFILE.VCF_FORMAT column. The exported VCF file contains the custom format field mapping data only if the VCF_FORMAT global variable is set. Otherwise only the

default format mapping data is exported. While updating VCF_FORMAT column ensure that only new custom format fields are appended and not existing ones.

4.5.2.1 Debugging Inconsistent Datatypes for FORMAT Field in VCF File

The VCF files can be generated by various in-house and open source tools, so there is a possibility of using non-standard datatypes for some of the VCF mapped columns in ODB. In such cases, an *Invalid number* error is generated by the loader without providing information about the line having a mismatch in datatype. To know the exact location of datatype mismatch, there is an additional parameter called '-validate_numbs Y' used by the VCF loader. The default value of this parameter is 'N'. Pass 'Y' as the parameter value to validate the numeric values in the VCF file. This parameter determines if there is a character value instead of the expected numeric value and displays the line number and sample order number which has the issue. For example, the error appears as follows:

- `Detail: Error in number conversion of row number 16 and specimen column S1`
- `Description: ORA-06502: PL/SQL: numeric or value error: character to number conversion error`

The error indicates that there is a data format issue for the first sample in line number 16 of the VCF file.

4.5.3 Data Load

The loader code uses an external table through a cursor to parse the header rows to store metadata records and validate specimens. Most of the file metadata is in XML format where there is an identifier or tag followed by an attribute value or XML definition. The metadata load sets the QUALIFIER_TAG to the identifier before the "=" character (that is, FORMAT, INFO, FILTER), and everything after the "=" character is copied to the QUALIFIER_VALUE column of the W_EHA_FILE_LOAD_QLFR table.

A separate external table is then used to read it in a single sequential (non-parallel) pass. This promotes accurate line number values and efficient loading from various storage locations such as DBFS. All other DML statements employ the temporary table used to load this data.

The loader then utilizes an `insert` statement to create all W_EHA_VARIANT records as long as the "i_load_mode" parameter is not set to "NON-VAR". This process creates the W_EHA_VARIANT_X records, when there is a large structural variant or the replace tag for non-deletion variants is larger than 1000 characters. The variant record is created by eliminating any overlapping DNA sequences and then adjusting the start position. Checksums and length checks are performed to avoid duplicate records in the W_EHA_VARIANT_X table. If the "i_xref_db" parameter is specified, all rows that have XREF information also create the W_EHA_VARIANT_XREF records.

> **Note:** The current loader can be used to load VCF files that do not have any specimens. This is convenient for loading information from dbSNP and Cosmic.

If the "i_validate_numbs" parameter is set to 'Y", then all fields that are expected to be numeric, are evaluated and any rows that have invalid numbers are stored in the log file. No further processing is performed. This is useful for debugging files that do not create any result records.

The loader then uses separate insert statements to create the following records:

- W_EHA_RSLT_SEQUENCING
- W_EHA_RSLT_NOCALL
- W_EHA_RSLT_NON_VARIANT
- W_EHA_RSLT_SV_BREAKEND
- W_EHA_RSLT_CONFLICT

The following table indicates the tables inserted based on LOAD_MODE:

Table 4–2 Tables inserted based on LOAD_MODE:

Name of the Record	VCF	GVCF	NON-VAR
W_EHA_RSLT_SEQUENCING	X	X	-
W_EHA_RSLT_NOCALL	X	X	X
W_EHA_RSLT_NON_VARIANT	-	X	X
W_EHA_RSLT_SV_BREAKEND	X	X	
W_EHA_RSLT_CONFLICT	-	X	-

4.5.3.1 Data Files

Two kinds of VCF files are available at 1000 Genomes - sites and genotypes. A sites file does not contain genotypic data and sample details whereas a genotype VCF file contains individual genotypic data along with sample information. The current loader supports only VCF files with sample and genotype data. The sample information is present in the header row of the VCF data following the FORMAT column. Each row represents one sample.

The data type representation format and its order for each sample are specified in the FORMAT column. All alleles for all samples are stored in the ALT column. To obtain 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.

Following is the list of passes that are used to process each VCF file:

1. The file is parsed for header columns that are indicated by "##" and the results are stored in the W_EHA_FILE_LOAD_QLFR table.
2. The file is parsed to create all referenced W_EHA_VARIANT records.

   Note: This pass does not require "GT" format field to facilitate loading reference VCF files.

3. The file is parsed to add records that do not use W_EHA_VARIANT foreign keys. This includes W_EHA_RSLT_NOCALL, W_EHA_RSLT_NON_VARIANT, W_EHA_RSLT_SV_BREAKEND.
4. The file is parsed to link all records to W_EHA_VARIANT. This includes W_EHA_RSLT_SEQUENCING.
5. In GVCF mode, the file is parsed to add W_EHA_RSLT_CONFLICT records.

6. The loader can read only 32 data types from the FORMAT column in the VCF file. Any data type, either supported or custom, not present in the first 32 data types of FORMAT column will not be loaded.

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

Following is the table mapping of VCF Result File (snps, indels, large SVs, and rearrangements) and gVCF:

Column Name in Result File	Table and Column Name in ODB	Description
CHROM	W_EHA_VARIANT.CHROMOSOME_WID W_EHA_RSLT_SEQUENCING.CHROMOSOME_WID W_EHA_RSLT_NON_VARIANT.CHROMOSOME_WID W_EHA_RSLT_NOCALL.CHROMOSOME_WID W_EHA_RSLT_CONFLICT.CHROMOSOME_WID W_EHA_RSLT_SV_BREAKEND.REF_CHROMOSOME_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. Three values are required 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 the 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 W_EHA_RSLT_SEQUENCING.START_POSITION W_EHA_RSLT_NON_VARIANT.START_POSITION W_EHA_RSLT_NOCALL.START_POSITION W_EHA_RSLT_CONFLICT.START_POSITION W_EHA_RSLT_SV_BREAKEND.REF_START_POSITION	This field is used as described above in the Chromosome column. For novel variants, POS is stored in the ABSOLUTE_POSITION column in the W_EHA_VARIANT table. For W_EHA_VARIANT.START_POSITION, Start_Position is relative to the value in the W_EHA_DNA_SOURCE.START_POSITION table using the pos value. Since, VCF does not have the end position information, while inserting novel variants in the VARIANT table, END_POSITION need to be calculated based on POS information and number of bases in REF.
ID	W_EHA_RSLT_SV_BREAKEND.BREAKEND_ID	This value is only stored for structural re-arrangement data where 'SVTYPE=BND' present in INFO column. This value in ID column is stored in BREAKEND_ID column.

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. 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 is implemented in the procedure which can be called with any of the above formats.
ALT	W_EHA_RSLT_SEQUENCING.ALLELE W_EHA_RSLT_CONFLICT.ALLELE W_EHA_RSLT_CONFLICT.ALLELE_CLOB W_EHA_RSLT_CONFLICT.ALLELE_CLOB W_EHA_VARIANT_X.ALLELE W_EHA_RSLT_SV_BREAKEND.ALLELE W_EHA_RSLT_SV_BREAKEND.ALLELE_CLOB	For sequencing results, this value constructs the replace tag and stores the value in the results table as per rules.
INFO.SVTYPE	If the INFO column of the VCF file does not have 'SVTYPE' tag, then this mutation is considered as either SNV or small indel. If the INFO column has 'SVTYPE=BND', then this mutation is considered as structural re-arrangement. If the INFO column has 'SVTYPE' other than 'BND', for example, 'SVTYPE=DEL', then these mutations are considered as large structural variants.	-
INFO.END	W_EHA_RSLT_NON_VARIANT.END_POSITION W_EHA_RSLT_NOCALL.END_POSITION W_EHA_RSLT_CONFLICT.END_POSITION	For gVCF and large SV data, END_POSITION value using 'END=' tag present in this INFO column.

Column Name in Result File	Table and Column Name in ODB	Description
INFO.CIPOS	W_EHA_RSLT_ SEQUENCING.CIPOS_START W_EHA_RSLT_ SEQUENCING.CIPOS_END	For large SV, the INFO column contains tag 'CIPOS' which contains two values. The first value is stored in CIPOS_START and second in CIPOS_END.
INFO.CIEND	W_EHA_RSLT_ SEQUENCING.CIEND_START W_EHA_RSLT_ SEQUENCING.CIEND_END	For large SV, the INFO column contains tag 'CIEND' which contains two values. The first value is stored in CIEND_START and the second n CIEND_END.
INFO.HOMLEN	W_EHA_RSLT_ SEQUENCING.HOMLEN	For large SV, the value for the tag 'HOMLEN' present in INFO column is stored here.
INFO.HOMSEQ	W_EHA_RSLT_ SEQUENCING.HOMSEQ	For large SV, the value for the tag 'HOMSEQ' present in INFO column is stored here.
INFO.MEINFO	W_EHA_RSLT_ SEQUENCING.MEINFO	For large SV, the value for the tag 'MEINFO' present in INFO column is stored here.
INFO.MATE_ID	W_EHA_RSLT_SV_ BREAKEND.MATE_ID	For structural rearrangement data, MATE_ID tag value present in INFO column is stored here.
INFO.EVENT_ID	For structural re-arrangement data, EVENT_ID tag value present in INFO column is stored here.	-
INFO.PRECISION	W_EHA_VARIANT.PERCISION W_EHA_RSLT_SV_ BREAKEND.PRECISION	For either Large SV or SV rearrangement data, if the 'IMPRECISE' tag is present in the INFO column, then 'IMPRECISE' value is populated in these columns, otherwise 'PRECISE' is populated.
FORMAT.GT	Gets the allele information for each sample. It is represented as '<allele1_num>/<allele2_num>'. In some cases instead of "/" there could be "|". ■ For diploid: 0|0 represents both the alleles from REF. ■ 0|1 represents one allele from REF and other from ALT allele. ■ 0/2 represents one allele from REF and other from ALT allele 2. ■ 1/3 represents one allele from first ALT value and 2nd allele from 3rd ALT value. ■ For haploid: only one allele number is represented. ■ '.' or './.'is specified if a call cannot be made for a sample at that locus.	-

Column Name in Result File	Table and Column Name in ODB	Description
FORMAT.FT	W_EHA_RSLT_SEQUENCING.GENOTYPE_FILTER W_EHA_RSLT_NON_VARIANT.GENOTYPE_FILTER W_EHA_RSLT_NOCALL.GENOTYPE_FILTER W_EHA_RSLT_CONFLICT.GENOTYPE_FILTER W_EHA_RSLT_SV_BREAKEND.GENOTYPE_FILTER	Sample genotype filter indicating if this genotype is called (the concept is similar to the FILTER field). PASS indicates that all filters have been passed. A semi-colon separated list of codes for filters that fail, or "." indicates that filters have not been applied. These values should be described in the meta-information in the same way as FILTERs (For string, white-space or semi-colons are not permitted).
FORMAT.GQ	W_EHA_RSLT_SEQUENCING.SCORE_VAF W_EHA_RSLT_NON_VARIANT.SCORE_VAF W_EHA_RSLT_NOCALL.SCORE_VAF W_EHA_RSLT_CONFLICT.SCORE_VAF W_EHA_RSLT_SV_BREAKEND.SCORE_VAF	This is mapped to W_EHA_RSLT_SEQUENCING.SCORE_VAF
QUAL	W_EHA_RSLT_SEQUENCING.QUAL W_EHA_RSLT_NON_VARIANT.QUAL W_EHA_RSLT_CONFLICT.QUAL W_EHA_RSLT_SV_BREAKEND.QUAL	Quality of the allele sequence. Mapped to QUAL column of VCF file.
FILTER	W_EHA_RSLT_SEQUENCING.FILTER W_EHA_RSLT_NON_VARIANT.FILTER W_EHA_RSLT_NOCALL.FILTER W_EHA_RSLT_CONFLICT.FILTER W_EHA_RSLT_SV_BREAKEND.FILTER	Filter applied to the particular record. Mapped to FILTER column in the VCF file.

Column Name in Result File	Table and Column Name in ODB	Description
FORMAT.DP	W_EHA_RSLT_SEQUENCING.TOTAL_READ_COUNT W_EHA_RSLT_NON_VARIANT.TOTAL_READ_COUNT W_EHA_RSLT_NOCALL.TOTAL_READ_COUNT W_EHA_RSLT_CONFLICT.TOTAL_READ_COUNT W_EHA_RSLT_SV_BREAKEND.TOTAL_READ_COUNT	Total number of reads that mapped to the defined allele sequence
FORMAT.AD	W_EHA_RSLT_SEQUENCING.ALLELE_READ_COUNT W_EHA_RSLT_SEQUENCING.REFERENCE_READ_COUNT W_EHA_RSLT_NON_VARIANT.REFERENCE_READ_COUNT W_EHA_RSLT_NOCALL.ALLELE_READ_COUNT W_EHA_RSLT_NOCALL.REFERENCE_READ_COUNT W_EHA_RSLT_CONFLICT.ALLELE_READ_COUNT W_EHA_RSLT_CONFLICT.REFERENCE_READ_COUNT W_EHA_RSLT_SV_BREAKEND.ALLELE_READ_COUNT	The first value is mapped to REFERENCE_READ_COUNT and consecutive values are mapped to ALLELE_READ_COUNT
FORMAT.BQ	W_EHA_RSLT_SEQUENCING.RMS_BASE_QUAL W_EHA_RSLT_NON_VARIANT.RMS_BASE_QUAL W_EHA_RSLT_NOCALL.RMS_BASE_QUAL W_EHA_RSLT_CONFLICT.RMS_BASE_QUAL W_EHA_RSLT_SV_BREAKEND.RMS_BASE_QUAL	RMS base quality at this position.

Column Name in Result File	Table and Column Name in ODB	Description
FORMAT.MQ	W_EHA_RSLT_SEQUENCING.RMS_MAPPING_QUAL W_EHA_RSLT_NON_VARIANT.RMS_MAPPING_QUAL W_EHA_RSLT_NOCALL.RMS_MAPPING_QUAL W_EHA_RSLT_CONFLICT.RMS_MAPPING_QUAL W_EHA_RSLT_SV_BREAKEND.RMS_MAPPING_QUAL	RMS mapping quality at this position.
FORMAT.GQX	W_EHA_RSLT_SEQUENCING.GENOTYPE_QUAL_X W_EHA_RSLT_NON_VARIANT.GENOTYPE_QUAL_X W_EHA_RSLT_NOCALL.GENOTYPE_QUAL_X W_EHA_RSLT_CONFLICT.GENOTYPE_QUAL_X W_EHA_RSLT_SV_BREAKEND.GENOTYPE_QUAL_X	GQX - Minimum of {Genotype quality assuming variant position, Genotype quality assuming non-variant position}
FORMAT.SS	W_EHA_RSLT_SEQUENCING.SOMATIC_STATUS_WID	W_EHA_SOMATIC_STATUS table pre-seeded with all values while installation. For more information, see Table A–4. The loader maps the SS value from the VCF file to the SOMATIC_STATUS_CODE column of the W_EHA_SOMATIC_STATUS table and a foreign key is created in W_EHA_RSLT_SEQUENCING with SOMATIC_STATUS_WID column.
FORMAT.SSC	W_EHA_RSLT_SEQUENCING.SOMATIC_SCORE	Somatic score of the variant

4.5.4 Command-Line Argument List

Name

VCF_loader.sh - load records

Synopsis

VCF_loader.sh -help

VCF_loader.sh <...options>

Description

Validates input options and calls the loader script for VCF and GVCF mode load_vcf.sql#odb_rslt_gvcf_util.process_gvcf and for NON-VAR mode odb_nonvar_gvcf_util.process_nonvar_gvcf

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external|S=copy to secure data file directory) [default: E]

`-custom_format <VARCHAR2>`

Custom format comma delimited (`format_name=column(,format_name=column)*`)

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-file_type_code* <VARCHAR2>`

File type code (GVCF|VCF) [default: VCF]

`-load_mode <VARCHAR2>`

Load mode (VCF|GVCF|NON-VAR)[default: VCF]

`i_validate_numbs <CHAR>`

Validate all number fields before insert (Y|N)[default: N)

`-alt_file_loc <VARCHAR2>`

Alternate file location link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

4.5.5 Examples

UNIX: with 'GVCF' file_type_code and 'GVCF' load_mode

```
$ sh VCF_loader.sh -db_wallet odb_user -data_file "YRI.trio.2010_
03.snps.genotypes_NEW.vcf" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_vendor
"vendor1" -reference_version "GRCh37.p8" -file_flg "E" -custom_format ""
-preprocess_dir "" -preprocess_file "" -data_file_path "" -dbfs_store ""
-file_type_code "GVCF" -load_mode "GVCF" -alt_file_loc "" -read_size ""
```

Windows: with 'GVCF' file_type_code and 'GVCF' load_mode

```
C:\> VCF_loader.bat -db_wallet odb_user -data_file "YRI.trio.2010_
03.snps.genotypes_NEW.vcf" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_vendor
"vendor1" -reference_version "GRCh37.p8" -file_flg "E" -custom_format ""
-preprocess_dir "" -preprocess_file "" -data_file_path "" -dbfs_store ""
-file_type_code "GVCF" -load_mode "GVCF" -alt_file_loc "" -read_size ""
```

When the file type is VCF and load mode is VCF, the same command will be used. Pass the -file_type_code as VCF an d -load_mode as VCF as a parameter with SH and BAT command.

4.6 MAF Sequence Data Loader

The 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 importing MAF versions 2.0 - 2.2.

4.6.1 Functional Description

The MAF loader currently loads only the variant records and not wild type (WT) information. For example:

- If one of the alleles is WT while the other is variant, then the loader only records the variant allele.
- If both the alleles are WT, then the loader does not load any of these alleles.
- If both alleles are variants and homozygous, then it stores only one record.
- If both alleles are MT and heterozygous, then it stores them as two separate records.

The loader obtains the chromosome and position details of a record from the MAF file and checks if the corresponding region of that chromosome exists in W_EHA_DNA_ SOURCE table for the reference version specified as input. If it is present, it maps this record to the W_EHA_DNA_SOURCE table as W_EHA_ VARIANT.SOURCE_WID. If the region is not found (for example, the chromosome and (or) position information is invalid), the loader ignores that record and does not log in to W_EHA_RSLT_LOG table.

The loader does not validate the accuracy of the reference nucleotides in the database. It assumes that the same version of reference mapped MAF data is loaded in to ODB. Ensure that the reference version of the results file being loaded matches that of the reference data available in ODB.

A single record in a MAF file contains data for both normal and tumor samples. The loader loads data for both these samples.

A typical MAF file contains information about multiple specimens. If one or more of the specimens do not exist in the CDM schema, then the loaders skips that row and logs an error with details in W_EHA_RSLT_LOG.

4.6.2 Data Load

The execution call of the stored procedure `ODB_RSLT_MAF_UTIL.process_maf()` is designed in one of the script files (load_maf.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, DATA SOURCE, SPECIMEN VENDOR, Reference Version, File Flag, Preprocess directory, Preprocess File, Data File Path, DBFS Store, Alternate file location (ftp location/http location), Read Size as input parameters.

It creates an external table dynamically and uploads data from the source file into it. External tables let Oracle 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, W_EHA_MAF_SPECIMEN, is created explicitly to store the Normal and Tumor sample barcodes. There are two passes through the MAF file. One creates a W_EHA_VARIANT_STG record and collects each unique specimen number into the global temporary table. A bulk collect then calls Odb_util.GET_SPECIMEN_WID for all the specimen numbers in one statement.

Another bulk `insert` statement inserts the data into W_EHA_RSLT_SEQUENCING.

A `select` statement parses data from the external table and employs a join with w_eha_variant and w_eha_dna_source table to get the dataset. This is then used to compute the gene_wid from w_eha_gene_segment table. The dataset returned with variant and DNA source table examines whether the start position of variant record is less than or equal to the end position of a gene segment. It also checks if the End position variant record is greater than or equal to the start position of the gene segment and populates the GENE_WID in W_EHA_STG_SEQUENCING table.

After inserting the record into the W_EHA_VARIANT_STG table, the PROCESS_VARIANT() procedure is called, which populates the W_EHA_VARIANT table.

4.6.2.1 Data files

Each row of a MAF file has two allele information for two sample types – a tumor sample and a normal one. These two sample IDs are specified in each row of the file. Sample ID of the tumor sample is specified in Tumor_Sample_Barcode and that of the normal one is specified in Matched_Norm_Sample_Barcode column. For a single file row, a maximum of eight records can be created in ODB depending on the heterozygosity and resemblance to the reference sequence - four for the tumor sample and four for the normal one.

Allele sequences for a deletion represent a variant and for an insertion represent a wild-type allele. If an allele sequence is the same as the Reference_Allele sequence, its information is not stored in the data bank. There is no information on NOCALL in MAF data, hence all the data is populated in the W_EHA_VARIANT and W_EHA_RSLT_SEQUENCING tables.

The loader validates the reference version, which is passed as an input parameter, against the W_EHA_VERSION table and populates the VERSION_WID in the corresponding result table.

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

Following is the table mapping of the 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 and 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 the W_EHA_VARIANT table with the 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 in the ABSOLUTE_POSITION column in the W_EHA_VARIANT table. For W_EHA_VARIANT.START_POSITION, Start_Position is relative to the value in the W_EHA_DNA_SOURCE.START_POSITION table.
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 also calculates the relative end position based on W_EHA_DNA_SOURCE.START_POSITION for END_POSITION in W_EHA_VARIANT.END_POSITION for novel variants.
Strand	W_EHA_VARIANT.STRAND W_EHA_VARIANT_STG.STRAND	This value indicates 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 the W_EHA_RSLT_SEQUENCING table.

Column Name in Result File	Table and Column Name in ODB	Description
Reference_Allele	W_EHA_VARIANT.REPLACE_TAG	This value is used for REPLACE_TAG, REPLACE_TAG and is used twice, one for the tumor and the other for normal sample. It 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. 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_VARIANT.REPLACE_TAG	For sequencing results this value constructs the replace tag. '-' value represents a deletion.
Tumor_Seq_Allele2	W_EHA_VARIANT.REPLACE_TAG	For sequencing results, this value constructs 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.
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_VARIANT.REPLACE_TAG	For sequencing results this value constructs the replace tag. '-' value represents a deletion.
Match_Norm_Seq_Allele2	W_EHA_RSLT_SEQUENCING.ALLELE W_EHA_VARIANT.REPLACE_TAG	For sequencing results this value constructs the replace tag. '-' value represents a deletion.
Score	-	This column is not functional in the MAF files and is currently not mapped.

4.6.3 Command-Line Argument List

Name

MAF_loader.sh - load records

Synopsis

MAF_loader.sh -help

MAF_loader.sh <...options>

Description

Validates input options and calls the loader script load_maf.sql#odb_rslt_maf_util.process_maf

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-alt_file_loc <VARCHAR2>`

Alternate file location, link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE]

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

4.6.4 Examples

UNIX

```
$ sh MAF_loader.sh -db_wallet odb_user -data_file "ut_maf.txt" -data_
directory "ODB_LOAD" -species_name "Homo sapiens" -study_name "STUDY1"
-datasource_name "CDM" -specimen_vendor "vendor3" -reference_version
"GRCh37.p8" -file_flg "E" -preprocess_dir "" -preprocess_file "" -data_
file_path "" -dbfs_store "" -alt_file_loc "" -read_size ""
```

Windows

```
C:\> MAF_loader.bat -db_wallet odb_user -data_file "ut_maf.txt" -data_
directory "ODB_LOAD" -species_name "Homo sapiens" -study_name "STUDY1"
-datasource_name "CDM" -specimen_vendor "vendor3" -reference_version
"GRCh37.p8" -file_flg "E" -preprocess_dir "" -preprocess_file "" -data_
file_path "" -dbfs_store "" -alt_file_loc "" -read_size ""
```

4.7 RNA-Seq Loader

4.7.1 Functional Description

The TCGA RNA SEQ data file format specifications are described here:

https://wiki.nci.nih.gov/display/TCGA/RNASeq+Data+Format+Specification#RNASeqDataFormatSpecification-Datafiles

TCGA has three different types of files: exon, gene, and splice junctions. Only the exon files are measured by exact chromosome locations. The other two files are calculated estimations based upon gene locations using this exon data file. Currently, support is provided for loading the exon version data files.

Once an exon data file is loaded you can select genes, which can map to specific chromosome regions. RNA sequencing based data has a data type alias of Quantification-Exon.

A RNASeq exon quantification file is a tabular, text-based, tab-separated dataset, with a single header row stating the column names. The file consists of the following columns:

- barcode: Identifies the sample. This column may or may not be used by the loader. For ODB v3.0, only files with this column can be loaded.
- exon: Provides standard chromosome token: chr1-chr22, chrX, chrY, chrM, followed by a coordinate pair, strand indicated with +/-, for example, chr1:12227:-,chr1:12595:+
- raw_counts: Stores raw read counts in positive floating point values or a zero, if unavailable.
- median_length_normalized: A normalized region length calculation in positive float or zero.
- RPKM: (Reads Per Kilobaseq exon Model per million mapped reads) Calculated expression intensity values in positive float or zero.

4.7.2 Data Load

The RNASeq Loader inserts data from an exon quantification file into the W_EHA_RSLT_RNA_SEQ table.

The last four columns of a TCGA exon file are populated into the table mentioned above. The EXON type file specifies a chromosome and range. This field will be parsed to find the corresponding chromosome record, strand, and separate the start and end positions. The actual value is stored in RESULT_EXON_NAME in W_EHA_RSLT_RNA_SEQ for reference.

The table stores an additional FK value for SPECIES, DNA reference version to which the results are mapped, and W_EHA_GENE table for each gene the reference mapping associates to a record. If no such gene is found in the current reference, a '0' value is added into the column. If the RPKM value for an input row is a null value (a blank, or has the text 'null'), then it is skipped by the loader.

The execution call of the stored procedure `ODB_RSLT_RNA_SEQ_UTIL.process_tcga_rna_seq()` is designed in one of the script files (load_tcga_rna_seq.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, Reference Version as mandatory input parameters, and DATA SOURCE, SPECIMEN VENDOR, File Flag, Preprocess directory, Preprocess File, Data File Path, DBFS Store, Alternate file location (ftp location or http location), Read Size as optional input parameters.

This stored procedure creates an external table dynamically and uploads data from the source file into it. External tables let Oracle query data that is stored outside the database in flat files.

Only one external table is created dynamically and holds the complete result data. There is one bulk `insert` statement which inserts data into the W_EHA_RSLT_RNA_SEQ table. The query uses the two inline views, one of which computes the gene WID and the other computes the start position, end position, chromosome WID, strand, row_count, median_length columns. These two inline views are then joined to populate the W_EHA_RSLT_RNA_SEQ table.

Note: The result types, and mainly the identifiers used in the first column are different for TCGA-Exon and TCGA-Gene. Additional columns specified to hold gene result record identifiers have been created, which remain empty as these are not filled by the exon loader. This is because gene record identifiers have a different querying requirement and are therefore separated in the table from the columns that are populated with exon result-identifiers. The remaining column fields (RPKM, median length, raw count) are common for both formats.

4.7.2.1 Data File

The Windows batch file of the RNA-seq loader requires an Oracle Wallet to be set up to run correctly.

Specimen number -

A valid specimen number that is either provided by the user or retrieved from the data file (the "barcode" column). It links the result records by using the specimen to the associated external datasource given in the previous parameter. If the Datasource is CDM or CDM_PATIENT, then this value should be present for a record in W_EHA_SPECIMEN_PATIENT_H table under SPECIMEN_NUMBER. If the datasource is CDM_SUBJECT, then this value should be present for a record in W_EHA_SPECIMEN_SUBJECT_H table.

Note: In Linux, it is not required to use Homo sapiens within "". That requirement is only for Windows.

Following is the table mapping of RNASeq exon result file:

Column Name in Result File	Table and Column Name in ODB	Description
exon	W_EHA_RSLT_RNA_SEQ.CHROMOSOME_WID	The column contains values in the following format:
	W_EHA_RSLT_RNA_SEQ.START_POSITION	'*<chromosome>*:*<absolute start position>*-*<absolute end position>*:*<strand>*'
	W_EHA_RSLT_RNA_SEQ.END_POSITION	The loader parses each value and populates data in the respective fields. The chromosome value is looked up in W_EHA_CHROMOSOME for its ROW_WID value to populate CHROMOSOME_WID. The entire value is place in RESULT_EXON_NAME.
	W_EHA_RSLT_RNA_SEQ.STRAND	
	W_EHA_RSLT_RNA_SEQ.RESULT_EXON_NAME	
raw_counts	W_EHA_RSLT_RNA_SEQ. RAW_COUNTS	Raw Read counts gives a positive floating point or zero.

Column Name in Result File	Table and Column Name in ODB	Description
median_length_ normalized	W_EHA_RSLT_RNA_SEQ. MEDIAN_LENGTH	Calculated average normalized median length of the exon region for which an RPKM count if generated. Stores a positive float or zero.
RPKM	W_EHA_RSLT_RNA_SEQ.RPKM	Reads Per Kilobaseq exon Model per million mapped reads. Stores a positive float or zero.

The barcode column in the file is optionally used to identify the sample when the sample number is not passed to the loader as an argument. The value from the second row (first after the header) is then taken as the specimen number.

4.7.3 Command-Line Argument List

Name

TCGA_RNA_SEQ_loader.sh - load records

Synopsis

TCGA_RNA_SEQ_loader.sh -help

TCGA_RNA_SEQ_loader.sh <...options>

Description

Validates input options and calls the loader script load_tcga_rna_seq.sql#odb_rslt_ rna_seq_util.process_tcga_rna_seq

Options

(*) required

`-db_wallet* <VARCHAR2> (Required unless the -db_conn/-db_user combination is used to log into the database)`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_ DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 1]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES, that is, for humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_number <VARCHAR2>`

Specimen number - Identification number of the specimen for which the genomic result file is being loaded. If CDM is referenced, this value should be defined in W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_NUMBER. If the specimen number argument is missing or is null (""), the value from the second row (first after the header) in the "barcode" column of the data file is used as the specimen number.

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of the specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR [default: NULL]

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR [default: NULL]

`-data_file_path <VARCHAR2>`

File system path to secure data file directory [default: NULL]

`-dbfs_store <VARCHAR2>`

Database file system store [default: NULL]

`-alt_file_loc <VARCHAR2>`

Alternate file location, link that is, ftp:location, http:location [default: NULL]

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE [default: NULL]

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object [default: NULL]

`-File_version (varchar2) [default: NULL]`

File Version of a result file.

4.7.4 Examples

UNIX

```
$ sh TCGA_RNA_SEQ_loader.sh -db_wallet odb_user -data_file "summary_
TCGA-AB-2803-03A-01T-0734-13.exon.quantification.txt" -data_directory
"ODB_LOAD" -species_name "Homo sapiens" -study_name "STUDY1" -datasource_
name "CDM" -specimen_number "RNA01" -specimen_vendor "vendor1" -reference_
version "GRCh37.p8" -file_flg "E" -read_size ''-file_version '3.1.4.0'
```

Windows

```
C:\> TCGA_RNA_SEQ_loader.bat -db_wallet odb_user -data_file "summary_
TCGA-AB-2803-03A-01T-0734-13.exon.quantification.txt" -data_directory
"ODB_LOAD" -species_name "Homo sapiens" -study_name "STUDY1" -datasource_
name "CDM" -specimen_number "RNA01" -specimen_vendor "vendor1" -reference_
version "GRCh37.p8" -file_flg "E" -file_version '3.1.4.0'
```

4.8 File Specimen Loader and File Lineage Linker

File Lineage Linker facilitates associating file records with each other (each association, a File Link, is directional with one file being a parent and the other a child. The user decides which file is the parent and which the child when running the File Lineage Linker). Another loader, File-Specimen Loader, is used to create file records for low-level (or other) files which have no real loader provided in ODB. These files can then be linked to other files or to each other. This loader also permits associating files with specimens.

4.8.1 File-Specimen Loader

The File-Specimen Loader creates records in the W_EHA_FILE table, representing files not loaded by any of the loaders provided in ODB. This permits creating records for files of unsupported types, such as BAM and other low level files. These records can be then linked with other file records (such as VCF files), to provide lineage associations. The File-Specimen Loader can also associate new or existing file records with specimen records.

> **Note:** The File Specimen Loader can create a file record for any File Type defined in the W_EHA_FILE_TYPE table, including file types supported by "real" loaders, such as VCF, CNV, and so on. It will not, however, load the file's contents into the database. Therefore, it should not be used for files supported by the provided loaders.

The loader can be run in two different modes. In one mode, it creates a new File record and associates it with at least one specimen. In this case, provide a file name and data directory. In the other mode, an existing File record is associated with one or more specimen. In this mode, the only required arguments (besides the connection arguments) are the File URI and the Specimen Number(s) and Vendor. The second mode is distinguished by using the `-append_specimen 1` argument.

In both modes, at least one specimen must be specified. Specimen numbers are available as a delimited string (the default delimiter is comma, but there is an argument that lets you specify a different delimiter), and there can be any number of items in it. However, only one Specimen Vendor number can be supplied, so that all specimen associated with a file in one run of this Loader should share the same Specimen Vendor Number. Subsequent runs in the Append Specimen mode can be used to associate specimen with other Specimen Vendor Number(s) with the same file.

The command-line arguments for the File-Specimen Loader are as follows:

Name

File_specimen_linker.sh

Synopsis

File_specimen_linker.sh -help

File_specimen_linker.sh <...options>

Description

Validates input options and calls the loader script

load_file_spec.sql#odb_rslt_file_spec_util.process_file_spec

Options

(*) denotes that it is required

- Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

  `-db_wallet* <VARCHAR2>`
  Required, unless the -db_conn/-db_user combination is used to log into the database

- Oracle connection string that is,

  "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CO NNECT_DATA=(SID=XE)))"

  `-db_conn* <VARCHAR2>`
  Required if -db_wallet is not provided

- ODB user name for the Database connection

  `-db_user* <VARCHAR2>`
  Required if -db_conn is provided

- Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

  `-log_level <VARCHAR2>`

- Print summary (1=yes|0=no) [default: 0]

  `-print_summary <NUMBER>`

- Data file name - Oracle external table LOCATION

  `-data_file* <VARCHAR2>`
  Required if loading a new file

- Oracle directory object: Oracle external table DIRECTORY. For information, see Section 2.1, "Setting Up a Directory Object".

  `-data_directory* <VARCHAR2>`
  required if loading a new file

- Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

  `-datasource_name* <VARCHAR2>`
- Specimen vendor: Sample vendor number of specimens with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

  `-specimen_vendor* <VARCHAR2>`
- Specimen numbers: A delimited list of identification numbers of specimens to be associated with the newly or previously loaded file. The default delimiter is comma, if this is not acceptable (for example, if one or more Specimen Numbers contain comma(s)), an alternative single-character delimiter can be defined using the -delimiter_char <CHAR> argument.

  `-specimen_numbers* <VARCHAR2>`
- File flag (E=external | S=copy to secure data file directory) [default: E]

  `-file_flg <CHAR>`
- File system path to secure data file directory

  `-data_file_path <VARCHAR2>`
- Database file system store

  `-dbfs_store <VARCHAR2>`
- Alternate file location link that is, ftp:location, http:location

  `-alt_file_loc <VARCHAR2>`
- File system path to Oracle directory object

  `-data_file_dir <VARCHAR2>`
- URI (unique resource identifier) for the file

  `-file_uri* <VARCHAR2>`
  Required, if appending associated specimen to an already loaded file. Optional, if a new file is being loaded
- The file type of the new file being loaded (the type must exist in the W_EHA_FILE_TYPE table)

  `-file_type* <VARCHAR2>`
  Mandatory, if loading a new file
- The file type version of the new file being loaded (the version must exist in the W_EHA_FILE_TYPE table, and the file type of the record must match the value of the -file_type argument)

  `-file_version <VARCHAR2>`
- Sets the loader mode (1=append specimen associations to an already loaded files | 0=load a new file) [default: 0]

  `-append_specimen <NUMBER>`
- Character to use as a delimiter when parsing the -specimen_numbers value [default: comma]

  `-delimiter_char <CHAR>`

Examples

- UNIX: Loading a new file

```
$ sh File_specimen_linker.sh -db_wallet odb_user -data_file "File020113.bam"
-file_uri "MYFILE1" -data_directory "ODB_LOAD"
-datasource_name "CDM" -specimen_numbers "Spec 1,Spec 2,Spec 3"
-specimen_vendor "vendor1" -file_flg "E" -file_type "BAM" -file_version "1.4"
```

- Windows: Loading a new file

```
C:\> File_specimen_linker.bat -db_wallet odb_user -data_file "File020113.bam"
-file_uri "MYFILE1" -data_directory "ODB_LOAD"
-datasource_name "CDM" -specimen_numbers "Spec 1,Spec 2,Spec 3"
-specimen_vendor "vendor1" -file_flg "E" -file_type "BAM" -file_version "1.4"
```

- UNIX: Appending specimen to an already loaded file

```
$ sh File_specimen_linker.sh -db_wallet odb_user -file_uri "MYFILE1" -append_
specimen 1 -datasource_name "CDM" -specimen_numbers "Spec 4,Spec 5" -specimen_
vendor "vendor1"
```

- Windows: Appending specimen to an already loaded file

```
C:\> File_specimen_linker.bat -db_wallet odb_user -file_uri "MYFILE1" -append_
specimen 1 -datasource_name "CDM" -specimen_numbers "Spec 4,Spec 5" -specimen_
vendor "vendor1"
```

4.8.2 File Lineage Linker

The File Lineage Linker creates a directional association between two files (W_EHA_FILE records), in which one file is a parent and the other a child. The user decides which file should be the child, and which the parent. The association record is created (unless it already exists) in the W_EHA_FILE_LINK table.

Normally, the files have to be associated with at least one common specimen to be linked. The File Lineage Linker verifies this before creating a link. However, this requirement can be overridden by using the `-force_link 1` argument.

The File Lineage Linker requires that the parent and child files are identified by their file URIs. This ensures that not more than one file link is created every time the Linker runs.

The command-line arguments for the File-Lineage Linker are as follows:

Name

File_lineage_linker.sh

Synopsis

File_lineage_linker.sh -help

File_lineage_linker.sh <...options>

Description

Validates input options and calls the loader script load_link.sql#

odb_rslt_link_util.process_link

Options

(*) denotes that it is required

- Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

  `-db_wallet* <VARCHAR2>`
  Required, unless the -db_conn/-db_user combination is used to log into the database

- Oracle connection string that is,

  "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CO NNECT_DATA=(SID=XE)))"

`-db_conn* <VARCHAR2>`
Required if -db_wallet is not provided

- ODB user name for the Database connection

  `-db_user* <VARCHAR2>`
  Required if -db_conn is provided

- Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

  `-log_level <VARCHAR2>`
- Print summary (1=yes | 0=no) [default: 0]

  `-print_summary <NUMBER>`
- The File URI of the child file in the link

  `-child_file_uri* <VARCHAR2>`
- The File URI of the parent file in the link

  `-parent_file_uri* <VARCHAR2>`
- Allow the files to be linked even if they have no associated specimen in common (1=yes | 0=no) [default: 0]

  `-force_link <NUMBER>`

Examples

- UNIX

```
$ sh File_lineage_linker.sh -db_wallet odb_user -parent_file_uri "MYFILE1"
-child_file_uri "MYFILE2" -force_link 1
```

- Windows

```
C:\> File_lineage_linker.bat -db_wallet odb_user -parent_file_uri "MYFILE1"
-child_file_uri "MYFILE2" -force_link 1
```

4.9 Copy Number Variation Loader

4.9.1 Functional Description

The Copy Number Variation (CNV) loader loads data from TCGA belonging to Affymetrix Genome-Wide Human SNP Array 6.0 platform. The input file should contain columns in the following order for the loader to perform correctly:

1. Sample
2. Chromosome
3. Start
4. End
5. Num_Probes
6. Segment_Mean

An extract of the input CNV file is shown in the table:

Sample	Chromosome	Start	End	Num_Probes	Segment_Mean
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	51598	219036	22	0.8546
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	219482	1176387	120	0.0513
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	1176449	1243413	47	0.623
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	1243440	5290540	2132	0.0167
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	5291209	5308749	6	0.6214
JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020	1	5308775	9230624	2368	-0.0261

4.9.2 Data Load

The execution call of the stored procedure `odb_rslt_cnv_util. process_cnv_nbr_var()` is designed in one of the script files (load_cnv.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTORY OBJECT, SPECIES NAME,STUDY, DATASOUCRE NAME, SPECIMEN VENDOR, FILE FLAG (External or Secured), DBFS_STORE, DNA_VERSION, and a few other input parameters.

It creates an external table and uploads data from the source file into it. It creates cnv_data_!!SEQ!! as an external table, which stores the complete result data. This table maps all the fields existing in the result file.

> **Note:** In the above external table, the !!SEQ!! string is replaced by ETL_PROC_ID at run time.

There is a single bulk `insert` statement which inserts records into the W_EHA_STG_COPY_NBR_VAR table.

A `select` statement, which parses the data from the external table, utilizes an inline query which gets the dataset of gene segment records. The query looks up the dataset returned from the inline query and checks if the start position of the result file is less than or equal to the end position of (a gene segment) + (start position of DNA source). It also checks if the End position of result file is greater than or equal to the (start position of the gene segment) + (start position of DNA source). The datasets of both inline queries are then outer joined with the ROW_WID of external table lookup for GENE_WID, and records are inserted into the W_EHA_STG_COPY_NBR_VAR table.

The loader associates this data with the FILE_TYPE_CODE 'Genome_Wide_SNP_6' in W_EHA_FILE_TYPE table to distinguish it.

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

4.9.3 Command-Line Argument List

Name

CNV_loader.sh - load records

Synopsis

CNV_loader.sh -help

CNV_loader.sh <...options>

Description

Validates input options and calls the loader script load_cnv.sql#odb_rslt_cnv_util.process_cnv_nbr_var

Options

(*) required

`-db_wallet* <VARCHAR2> (Required, unless the -db_conn/-db_user combination is used to log into the database)`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES, that is, for humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_number <VARCHAR2>`

Specimen number - Identification number of the specimen for which the genomic result file is being loaded. If CDM is referenced, this value should be defined in W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_NUMBER. If the specimen number is not passed on the command-line, or is passed as "", the value in the second (first after the header) row in the first column ("sample") in the file will be used as the Specimen Number.

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of the specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-alt_file_loc <VARCHAR2>`

Alternate file location link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

4.9.4 Examples

UNIX

```
$ sh CNV_loader.sh -db_wallet odb_user -data_file "JOUAL_p_TCGA_b96_SNP_N_
GenomeWideSNP_6_A01_748020.hg19.seg.txt" -data_directory "ODB_LOAD"
-species_name "Homo sapiens" -study_name "STUDY1" -datasource_name "CDM"
```

```
-specimen_number "JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020"
-specimen_vendor "vendor1" -reference_version "GRCh37.p8" -file_flg "E"
-read_size ''
```

Windows

```
C:\> CNV_loader.bat -db_wallet odb_user -data_file "JOUAL_p_TCGA_b96_SNP_
N_GenomeWideSNP_6_A01_748020.hg19.seg.txt" -data_directory "ODB_LOAD"
-species_name "Homo sapiens" -study_name "STUDY1" -datasource_name "CDM"
-specimen_number "JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020"
-specimen_vendor "vendor1" -reference_version "GRCh37.p8" -file_flg "E"
```

4.10 Single Channel Gene Expression Loader

4.10.1 Functional Description

The single channel gene expression loader loads gene expression data into the W_EHA_RSLT_GENE_EXP table. The loader begins with loading all the hybridization sets (consisting of intensity, call and P-value) from the input file into an intermediary staging table W_EHA_STG_GENE_EXP.

While reading from the file, it assumes that there are a maximum of 15 hybridization sets, which translates to a maximum of 45 columns. If there are fewer columns, the loader uses empty fillers in the staging table for the unavailable hybridization sets. For each record from the staging table, it verifies if the probe name exists in the W_EHA_PROBE table along with the matching version and species ID, which are passed as input parameters.

If a match is found, the loader inserts all the hybridization sets available for that record into W_EHA_RSLT_GENE_EXP table, excluding the empty fillers. If the probe name does not exist in the W_EHA_PROBE table, the loader skips that record. If the probe exists, but with non-matching version and (or) species ID, the loader logs a warning into W_EHA_RSLT_LOG table, with a message that version and (or) species do not match.

The W_EHA_RSLT_LOG table contains error records if records were not loaded successfully into the target tables. If an input row intensity value is a null value (or blank, or has the text *null*), then the loader skips this row.

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

4.10.2 Data Load

The gene expression loader primarily loads into the W_EHA_RSLT_GENE_EXP table. It also updates two aggregate tables, W_EHA_RSLT_GXP_HYBRID_AGG and W_EHA_RSLT_GXP_PROBE_AGG. To run the loader, use 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 (for details, see Section 3.6, "Probe Loader") has already populated W_EHA_PROBE table with probe names corresponding to the genes.

The input file for this loader should contain normalized intensity values, and optional inputs of present or absent calls and P-value (such as the output of Affymetrix's MAS5 algorithm). The input file permits multiple hybridization intensity data in a tabular format.

4.10.2.1 Assumptions for Data File

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

- The file is tab delimited.
- The first row is always the header.
- The first column is named DATA.
- Each hybridization present in the data file should have three columns in the following order:
  - Intensity - The header value should be the Hybridization Name
  - Call
  - P-Value
- The header of the first column for each hybridization should contain only the hybridization name. The values in this column are the hybridization intensity values.
- The total size of the header in the data file should not be greater than 32000 characters.

4.10.2.2 Mappings for Gene Expression Loader

Following are the table 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
HYBRIDIZATION_Call	W_EHA_RSLT_GENE_EXP.CALL
HYBRIDIZATION_P-VALUE	W_EHA_RSLT_GENE_EXP.P_VALUE

4.10.2.3 Aggregate Tables

There are two tables introduced in version 3.0, namely W_EHA_RSLT_GXP_HYBRID_AGG and W_EHA_RSLT_GXP_PROBE_AGG. These tables are populated with calculated aggregate values for the normalized intensity column of result file. The aggregates are: median (MEDIAN), average (AVG), minimum (MIN), maximum (MAX), Standard Deviation (STDDEV), and Variance (VARIANCE). The data in W_EHA_RSLT_GXP_HYBRID_AGG is aggregated over hybridizations, and in W_EHA_RSLT_GXP_PROBE_AGG over probes.

4.10.3 Command-Line Argument List

Name

single_channel_gene_expr_loader.sh - load records

Synopsis

single_channel_gene_expr_loader.sh -help

single_channel_gene_expr_loader.sh <...options>

Description

Validates input options and calls the loader script load_single_channel_gene_expr.sql#odb_rslt_single_channel_util.process_single_channel

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes|0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes|0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes|0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_number* <VARCHAR2>`

Specimen number - Identification number of specimen for which the genomic result file is being loaded. If CDM is referenced, this value should be defined in W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_NUMBER

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-alt_file_loc <VARCHAR2>`

Alternate file location link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

4.10.4 Examples

UNIX

```
$ sh single_channel_gene_expr_loader.sh -db_wallet odb_user -data_file
"mas5_expression_summary_part1.txt" -data_directory "ODB_LOAD" -species_
name "Homo sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_
number "RNA01" -specimen_vendor "vendor1" -reference_version "GRCh37.p8"
-file_flg "E" -preprocess_dir "" -preprocess_file "" -data_file_path ""
-dbfs_store "" -alt_file_loc "" -read_size ""
```

Windows

```
C:\> single_channel_gene_expr_loader.bat -db_wallet odb_user -data_file
"mas5_expression_summary_part1.txt" -data_directory "ODB_LOAD" -species_
name "Homo sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_
number "RNA01" -specimen_vendor "vendor1" -reference_version "GRCh37.p8"
-file_flg "E" -preprocess_dir "" -preprocess_file "" -data_file_path ""
-dbfs_store "" -alt_file_loc "" -read_size ""
```

4.11 Dual Channel Loader

4.11.1 Functional Description

The dual channel loader supports Agilent 244K Custom Gene Expression G4502A-07 platform specific Level-3 (Gene level) input files from TCGA. It inputs Level-3 result data, which contains gene symbols and associated LOWESS log2 transformed ratio gene expression values and loads it into the W_EHA_RSLT_2CHANNEL_GXP result table.

ADF data with a specific user label links the Dual Channel data comprising genes with the specific DNA Reference Version through the GENE_WID foreign key in the W_EHA_RSLT_2CHANNEL_GXP table. Currently, for each Gene Symbol or Ratio input from the file, the loader is set to generate a record for each GENE_WID value taken from W_EHA_GENE_SEGMENT reference table where the genomic coordinates of the corresponding Composite Name at least partially match the genomic coordinates of a Gene Segment in the EMBL reference.

To correctly map the genomic coordinates of the result composite genes (using the ADF file composite annotation loaded through the ADF Data Loader) to the EMBL reference genome, the genomic reference version of the loaded EMBL release must match the reference version of the ADF file. That is, the EMBL data loaded in ODB must be the same genomic release as that given in the ADF file.

The ADF file data must be input into ODB, using the ADF data loader (and with the same ADF User Label) before loading 2channel result data with this loader.

If the Log2 ratio value for an input row is a null value (a blank, or has the text 'null'), then this row is skipped by the loader during the insert to the result table.

4.11.2 Data Load

The execution call of the stored procedure `odb_rslt_dual_channel_util. process_ dual_channel()` is designed in one of the script files (load_dual_channel.sql). This stored procedure accepts FILE NAME, ORACLE DIRECTORY OBJECT, STUDY, DATASOURCE NAME, SPECIMEN NAME, SPECIMEN VENDOR, SPECIES NAME, (ADF) USER LABEL, (DNA) REFERENCE VERSION, FILE FLAG (External or Secure), DBFS_STORE, DNA_VERSION, and a few other input parameters.

It creates an external table and uploads data from the source file into it. The stored procedure creates dual_channel_data_!!SEQ!! as an external table. This external table stores the complete result data. This table maps all the fields existing in the result file.

Note: In the above external table, the !!SEQ!! string is replaced by ETL_PROC_ID at the run time.

There is a single bulk `insert` statement written dynamically. This statement inserts the record into the W_EHA_STG_2CHANNEL_GXP table.

A `select` statement which parses the data from the external table utilizes an inline query which gets the dataset of gene segment records. The query looks up the dataset returned from the inline query and checks whether the start position of the result file is less than or equal to the end position of gene segment + start position of DNA source. It will also check whether the End position of result file greater than or equal to start position of gene segment + start position of DNA source. The dataset of both inline

queries is then the outer join with the ROW_WID of external table to look up the GENE_WID and populates the records in W_EHA_STG_2CHANNEL_GXP table.

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

4.11.3 Command Line Argument List

Name

dual_channel_gene_expr_loader.sh - load records

Synopsis

dual_channel_gene_expr_loader.sh -help

dual_channel_gene_expr_loader.sh <...options>

Description

Validates input options and calls the loader script load_dual_channel_gene_expr.sql#odb_rslt_dual_channel_util.process_dual_channel

Options

(*) required

`-db_wallet* <VARCHAR2>`

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

`"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"`

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file* <VARCHAR2>`

Data file name - Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-species_name* <VARCHAR2>`

Species name defined in W_EHA_SPECIES that is, For humans "Homo sapiens"

`-study_name* <VARCHAR2>`

Study name defined in W_EHA_RSLT_STUDY.RESULT_STUDY_NAME

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_number* <VARCHAR2>`

Specimen number - Identification number of specimen for which the genomic result file is being loaded. If CDM is referenced, this value should be defined in W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_NUMBER

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-control_specimen* <VARCHAR2>`

Control specimen

`-user_label* <VARCHAR2>`

User label (W_EHA_ADF.USER_LABEL) used to identify a composite record's source ADF dataset, that is, AgilentG4502A_07_1

`-reference_version <VARCHAR2>`

"DNA" reference version label defined in W_EHA_VERSION.VERSION_LABEL

`-file_flg* <CHAR>`

File flag (E=external|S=copy to secure data file directory) [default: E]

`-preprocess_dir <VARCHAR2>`

Preprocess directory - Oracle external table PREPROCESSOR

`-preprocess_file <VARCHAR2>`

Preprocess file - Oracle external table PREPROCESSOR

`-data_file_path <VARCHAR2>`

File system path to secure data file directory

`-dbfs_store <VARCHAR2>`

Database file system store

`-alt_file_loc <VARCHAR2>`

Alternate file location link that is, ftp:location, http:location

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object

4.11.4 Examples

UNIX

```
$ sh dual_channel_gene_expr_loader.sh -db_wallet odb_user -data_file
"dual_channel_summary.txt" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_number
"JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020" -specimen_vendor
"vendor1" -control_specimen "Stratagene Univeral Reference" -user_label
"AgilentG4502A_07_01" -reference_version "GRCh37.p8" -file_flg "E"
```

Windows

```
C:\> dual_channel_gene_expr_loader.bat -db_wallet odb_user -data_file
"dual_channel_summary.txt" -data_directory "ODB_LOAD" -species_name "Homo
sapiens" -study_name "STUDY1" -datasource_name "CDM" -specimen_number
"JOUAL_p_TCGA_b96_SNP_N_GenomeWideSNP_6_A01_748020" -specimen_vendor
"vendor1" -control_specimen "Stratagene Univeral Reference" -user_label
"AgilentG4502A_07_01" -reference_version "GRCh37.p8" -file_flg "E"
```

4.12 Quality Control Metadata Loader

The Quality Control Metadata Loader captures Specimen and Analysis metadata and loads them into ODB. Although ODB provides standard %_QLFR suffixed tables that can accommodate extensible metadata attributes as name or value pairs, it does not support range queries. To provide such a feature, a table (W_EHA_QUALIFIER) is created in ODB and two additional columns are appended to this and the QLFR-suffixed tables, that store numeric and date metadata values. These columns are populated only for tags that represent dates or numbers. The metadata input file is a standard CSV file format, created by the user. A detailed description of this file is provided below.

4.12.1 Functional Description

Two new tables provide context and improve qualifier search and display:

- W_EHA_QLFR_CATEGORY: Deals with Qualifier categories group qualifiers into distinct functional areas. Some category examples are:
  - Run
  - Analysis
  - Analysis Component
  - Loader
  - Sample Preparation
  - Specimen
- W_EHA_QUALIFIER: Describes qualifiers used for all new %_QLFR tables. The logical key is a combination of table name, qualifier tag name and qualifier category (numeric, date, or character string). The table contains foreign keys to W_EHA_UNIT_OF_MEASURE, thereby providing the option to name a unit of measure for values, and W_EHA_QLFR_CATEGORY to store user defined categories. W_EHA_QLFR_TABLE links to this table and stores the table names of the %_QLFR suffixed table using a qualifier.

Each record loaded into the QUALIFIER table always has a QLFR_CHAR_VALUE column populated with a reported value. Additionally, all numeric values are loaded into QLFR_NUMB_VALUE column. All values for tag type *date* are loaded into QLFR_

DATE_VALUE column. Both date and numeric values can be queried and sorted by range. DISPLAY_ORDER and DISPLAY_NAME attributes provide additional means to improve metadata reports. This QUALIFIER table also includes fields to support units of measure. PREFERRED_UNIT is a flag that indicates whether a particular unit is a *preferred* unit. The measurements that are not associated with preferred units can be converted into preferred units using translation rules defined in the W_EHA_QLFR_TRANSLATION table.

4.12.2 Data Load

The QC Metadata loader takes one or more CSV files as input and loads them into the following tables:

- w_eha_rslt_file_spec_qlfr
- w_eha_rslt_spec_qlfr
- w_eha_qualifier
- w_eha_qlfr_table
- w_eha_qlfr_category
- w_eha_unit_of_measure

The execution call of the stored procedure `ODB_RSLT_METADATA_UTIL.process_metadata()` is designed in one of the script files (load_metadata.sql). This stored procedure accepts as input DATA FILE LIST, ORACLE DIRECTYORY OBJECT, DATA SOURCE, SPECIMEN VENDOR, FILE FLAG, DATA FILE PATH, DBFS STORE, DATA FORMAT, and READ SIZE as optional input parameters.

It creates an external table dynamically and uploads data from each source file in the file list into it. This lets Oracle query data that is sourced from the flat files outside the database. The procedure processes the external metadata and inserts it into a staging table, W_EHA_QUALIFIER_STG. It then checks for the presence of each qualifier in the W_EHA_QUALIFIER table before inserting new qualifiers into it. A specimen lookup procedure is called and the process, based on the input field TABLE_NAME value, adds records to W_EHA_RSLT_SPEC_QLFR and W_EHA_FILE_SPEC_QLFR, linking the qualifier value to result data and file.

4.12.2.1 Data File

Each metadata file is a comma separated text file with the following fields:

Field Header	Description
TABLE_NAME	Target ODB qualifier table, this can be W_EHA_RSLT_SPEC_QLFR, or W_EHA_RSLT_FILE_SPEC_QLFR
QLFR_CATEGORY_NAME	Qualifier Category type
QLFR_TAG	Qualifier tag type name
DISPLAY_NAME	Display name to type
DISPLAY_ORDER	Order of display
DATA_TYPE	Value data type
UOM_NAME	Unit of measure type
PREFERRED_UNIT	-
SPECIMEN_NUMBER	Specimen ID of qualifier

Field Header	Description
SPECIMEN_VENDOR_NUMBER	Specimen vendor ID
SPECIMEN_DATA_SOURCE	Data source for specimen lookup
FILE_URI	Unique Identifier of specimen result file
QLFR_CHAR_VALUE	Stores Character string based Qualifier values
QLFR_NUMB_VALUE	Stores numerical qualifier values
QLFR_DATE_VALUE	Store data formatted qualifier values

The following table lists the sample content of a metadata file.

Table 4–3 Sample Content

TABLE_NAME	QLFR_CATEGORY_NAME	QLFR_TAG	DISPLAY_NAME	DISPLAY_ORDER	DATA_TYPE	UOM_NAME	PREFERRED_UNIT	SPECIMEN_NUMBER	SPECIMEN_VENDOR_NUMBER	SPECIMEN_DATA_SOURCE	FILE_URI	QLFR_CHAR_VALUE	QLFR_NUMB_VALUE	QLFR_DATE_VALUE
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	COLLECTION_DATE	-	-	DATE	-	-	TCGA-02-0075-10A-01W	vendor3	CDM_PATIENT	-	28-Aug-13	-	8/28/2013 14:22
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	COLLECTION_DATE	-	-	DATE	-	-	TCGA-08-0389-01A-01W	vendor3	CDM_PATIENT	-	28-Aug-13	-	8/28/2013 14:22
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	COLLECTION_DATE	-	-	DATE	-	-	TCGA-08-0389-11A-01W	vendor3	CDM_PATIENT	-	28-Aug-13	-	8/28/2013 14:22
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	DNA_CONC (MG/ML)	DNA Concentration	1	NUMBER	mg/ml	-	TCGA-02-0007-01A-01W	vendor3	CDM_PATIENT	-	.006 mg/ml	0.006	-
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	DNA_CONC (MG/ML)	DNA Concentration	1	NUMBER	mg/ml	-	TCGA-02-0007-10A-01W	vendor3	CDM_PATIENT	-	.007 mg/ml	0.007	-
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	DNA_CONC (MG/ML)	DNA Concentration	1	NUMBER	mg/ml	-	TCGA-02-0028-01A-01W	vendor3	CDM_PATIENT	-	.008 mg/ml	0.008	-
W_EHA_RSLT_SPEC_QLFR	SAMPLE_PREP	DNA_CONC (MG/ML)	DNA Concentration	1	NUMBER	mg/ml	-	TCGA-02-0028-10A-01W	vendor3	CDM_PATIENT	-	.009 mg/ml	0.009	-
W_EHA_RSLT_FILE_SPEC_QLFR	RUN	RUNID	RunID	1	CHARACTER	-	-	TCGA-08-0390-01A-01W	vendor3	CDM_PATIENT	ABCDEFMAF1	120126_SN316_0202_19	-	-

Table 4–3 (Cont.) Sample Content

TABLE_NAME	QLFR_CATEGORY_NAME	QLFR_TAG	DISPLAY_NAME	DISPLAY_ORDER	DATA_TYPE	UOM_NAME	PREFERRED_UNIT	SPECIMEN_NUMBER	SPECIMEN_VENDOR_NUMBER	SPECIMEN_DATA_SOURCE	FILE_URI	QLFR_CHAR_VALUE	QLFR_NUMB_VALUE	QLFR_DATE_VALUE
W_EHA_RSLT_FILE_SPEC_QLFR	RUN	RUNID	RunID	1	CHARACTER	-	-	TCGA-08-0390-11A-01W	vendor3	CDM_PATIENT	ABCDEFMAF1	120126_SN316_0202_20	-	-
W_EHA_RSLT_FILE_SPEC_QLFR	RUN	RUNID	RunID	1	CHARACTER	-	-	TCGA-08-0390-01A-01W	vendor3	CDM_PATIENT	ABCDEFMAF1	120126_SN316_0202_21	-	-
W_EHA_RSLT_FILE_SPEC_QLFR	RUN	RUNID	RunID	1	CHARACTER	-	-	TCGA-08-0390-10A-01W	vendor3	CDM_PATIENT	ABCDEFMAF1	120126_SN316_0202_22	-	-

4.12.3 Command-Line Argument

Name

METADATA_loader.sh - load records

Synopsis

METADATA_loader.sh -help

METADATA_loader.sh <...options>

Description

Validates input options and calls the loader script load_metadata.sql# odb_rslt_metadata_util.process_metadata

Options

(*) required

`-db_wallet* <VARCHAR2>` (required unless the -db_conn/-db_user combination is used to log into the database)

Oracle wallet name, see Section 2.2, "Setting Up an Oracle Wallet"

`-db_conn* <VARCHAR2>` (required if -db_wallet is not provided)

Oracle connection string that is,

```
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521))(CONNECT_DATA=(SID=XE)))"
```

`-db_user* <VARCHAR2>` (required if -db_conn is provided)

ODB user name for the Database connection.

`-check_version <NUMBER>`

Run check version (1=yes | 0=no) [default: 0]

`-check_version_non_i <NUMBER>`

Run check version in non-interactive mode (1=yes | 0=no) [default: 1]

`-log_level <VARCHAR2>`

Set log level TRACE, DEBUG, INFO, WARNING, ERROR [default: INFO]

`-print_summary <NUMBER>`

Print summary (1=yes | 0=no) [default: 0]

`-data_file_list* <VARCHAR2>,<VARCHAR2>…`

A comma separated list of data file names - each an Oracle external table LOCATION

`-data_directory* <VARCHAR2>`

Oracle directory object - Oracle external table DIRECTORY, see Section 2.1, "Setting Up a Directory Object"

`-datasource_name* <VARCHAR2>`

Datasource name defined in W_EHA_DATASOURCE.DATASOURCE_NM [default: CDM]

`-specimen_vendor* <VARCHAR2>`

Specimen vendor - Sample vendor number of specimen with genomic result data. If CDM is referenced, this value should be defined in the W_EHA_SPECIMEN_PATIENT_H.SPECIMEN_VENDOR_NUMBER

`-file_flg* <CHAR>`

File flag (E=external | S=copy to secure data file directory) [default: E]

`-data_file_path <VARCHAR2>`

File system path to secure data file directory [default: NULL]

`-dbfs_store <VARCHAR2>`

Database file system store [default: NULL]

`-date_format <VARCHAR2>`

Format of the date data type file input field [default: 'YYYY-MM-DD HH24:MI:SS']

`-read_size <NUMBER>`

Read size in bytes - Oracle external table READSIZE [default: NULL]

`-data_file_dir <VARCHAR2>`

File system path to Oracle directory object [default: NULL]

4.12.4 Examples

UNIX

```
$ sh METADATA_loader.sh [-db_wallet odb_user -data_file_list "FILE_NOV_28_
2013_04.csv,FILE_AUG_28_2013_04.csv" -data_directory "ODB_LOAD"
-datasource_name "CDM_PATIENT" -specimen_vendor "vendor1" -file_flg "E"
-date_format "mm/dd/yyyy HH24:MI" -data_file_path "" -dbfs_store "" -read_
size ""
```

Windows

```
C:\> METADATA_loader.bat -db_wallet odb_user -data_file_list "FILE_NOV_28_
2013_04.csv,FILE_AUG_28_2013_04.csv" -data_directory "ODB_LOAD"
-datasource_name "CDM" -specimen_vendor "vendor1" -file_flg "E" -date_
format "mm/dd/yyyy HH24:MI" -data_file_path "" -dbfs_store "" -read_size
""
```

4.13 Typical Errors Associated with Result Loaders

Errors have been observed while running various ODB loaders. The loader run aborts prematurely with the following error message: *ORA-01460 unimplemented or unreasonable conversion requested*.

Oracle recommends applying an RDBMS patch to the TRC database that fixes this bug. See Oracle Support Bug 13099577 (ORA-1460 WHEN PARALLEL QUERY SERVERS ARE USED) available here https://mosemp.us.oracle.com/epmos/faces/BugDisplay?id=13099577 for details.

4.13.1 Errors Relevant to Sequencing Loads

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 the W_EHA_RSLT_LOG table is 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 4–4.

Table 4–4 Errors Generated while Loading Sequencing Files (CGI masterVar file used as example)

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

Table 4–4 (Cont.) Errors Generated while Loading Sequencing Files (CGI masterVar file used as example)

Column Name	Description	Examples
ERROR_DESC	Error message	ORA-01403 NO DATA FOUND
RECORD_DETAIL	Verifying Study Name	-
ETL_PROC_WID	Each load identifier.	-

4.13.2 VCF Loader Errors

The VCF or gVCF loader processes files in multiple passes. The loader creates 3 external tables.

- If the process fails while processing the specimen specification table, an error 'Processing external specimen table for specimen headers' is logged in the w_eha_rslt_log table.
- If the process fails while creating the metadata external table, an error 'Processing external table for metadata' is logged in the result log table.
- If the process fails while creating qualifier records, an error 'Processing metadata records(w_eha_file_load_qlfr)' is logged in the result log table.
- If the process fails while retrieving specimen from the external table, an error 'Retrieving specimen numbers from external specimen table is logged in the result log table.
- If the loader attempts to load the file, which has more than 986 samples, then the error 'File loaded cannot have more than 986 specimens is logged in the result log table.
- If the process fails while executing variant staging records, an error 'Processing variant staging records (w_eha_variant_stg, w_eha_variant_x_stg)' is logged into the result log table.
- If the process fails while populating variant table from variant staging table, an error 'Processing variant records(w_eha_variant, w_eha_variant_log)' is logged in the result log table.
- If the process fails while populating variant result records, an error 'Processing result sequencing records (w_eha_stg_sequencing, w_eha_stg_sequencing_x, w_eha_stg_struct_var) is logged in the result log table, and if it fails while processing non variant records, an error 'Processing result sequencing records (w_eha_stg_struct_var, w_eha_stg_sv_breakend, w_eha_stg_nocall, w_eha_stg_non_variant)' is logged in the result log table.
- If the process fails while populating conflict records, then an error 'Processing result sequencing records(w_eha_stg_conflict)' is logged in the result log table.
- If the loader processes the xref records and the process fails, then an error 'Processing variant xref records (w_eha_variant_xref)' is logged, if the process fails while computing the nocall collapsing function then an error 'Collapsing result nocall staging records (w_eha_rslt_nocall) is logged in the result log table.
- If there are any non-standard datatype formats in the file, where instead of a number a character is found for the datatype mapped by the loader, an 'Invalid number' error is generated by the loader. To know the exact location of error in the file, use '-validate_numbs Y' parameter in the VCF loader. This parameter will determine if there is a character value instead of the expected numeric value and would output the line number and Sample order number which has the issue.

- Other possible errors are: 'Generating etl process id', 'Generating enterprise id', Verifying result file type ({0}, {1}) and processing result file record(w_eha_file)' 'Verifying {0} reference version={1}', 'Verifying result type', 'Verifying datasource name', 'Verifying species name', 'Verifying study name', 'Parsing global flex fields', 'Retrieving specimen ids', 'Getting flanking offset', 'Processing external data table' and 'Dropping external tables'.

4.13.3 CGI Loader Errors

If the process fail while the loader is unable to verify the file type and version error message 'verifying result file type ({type name}, {version}) and processing result file record(w_eha_rslt_file)', is logged in the result log table.

4.13.4 MAF Loader Errors

Only one external table is created for MAF result data. If the process fails at this step, an error 'Processing external data table' is logged into the w_eha_rslt_log table. A first BULK `insert` statement creates a variant staging and specimen record. If the process fails at this stage, an error 'Processing variant staging and specimen staging records (w_eha_variant_stg, w_eha_maf_specimen, w_eha_maf_specimen_log)' is logged into the result log table.

If the process fails while retrieving the specimen ID for a record from global temporary table, an error 'Processing list of specimens' is logged into the result log table. If the error occurs while processing variant staging records (which populates the w_eha_ variant table), an error 'Processing variant records (w_eha_variant, w_eha_variant_ log)' is logged into result log table.

If the process fails while executing the bulk insert statement which populates the target result table, an error 'Processing result records (w_eha_stg_sequencing, w_eha_ stg_sequencing_x)' is logged into the result log table.

Other possible errors are: 'Generating etl process id', 'Generating enterprise id', Verifying result file type ({0}, {1}) and processing result file record(w_eha_file)' 'Verifying {0} reference version={1}', 'Verifying result type', 'Verifying datasource name', 'Verifying species name', 'Verifying study name', 'Getting flanking offset', 'Processing external data table' and 'Dropping external tables'.

4.13.5 Single Channel Gene Expression Loader Errors

If the process fails while retrieving DNA version ID, specimen ID, datasource name or study ID, an error is logged into the result log table and displayed in the error summary at the end. If the process fails while processing the hybridization header external table, the Processing hybridization header table error is logged. If the process fails while retrieving the hybridization name from the header table, the error 'Retrieving hybridization name from header table' is logged.

Similar to VCF, a batch of 45 expression data is processed at a time and if the process fails at this stage, an error 'Processing data table for the set of gene expression' is logged. If the process fails while dropping the expression and hybridization external tables, error messages 'Dropping expression data table' and 'Dropping hybridization table' respectively are logged.

4.13.5.1 Missing Probe Link Issue

When re-running the probe loader to load probes, with the same probe names as existing probes, the loader updates the existing probes to point to the new DNA reference and probe versions given by the loader.

Now, when running the Gene expression loader, a result record will only be inserted to the result table when there is match for Probe name and the DNA reference version and Species when looking up the probe table.

This means running the gene expression loader but passing a reference version parameter (say 'GRCH37.P7') when the corresponding probe records point to another reference version (say 'GRCH37.P8'); the loader logs a warning message (for example, 'Version and/or species id are not matching for the probe=1554103_at') and does not load the record.

To load result records pointing to multiple reference versions, load gene expression result files for a particular reference version BEFORE re-running the probe loader updating the probes in the probe table to a new reference version. Then load the next set of gene expression result files pointing to the new version.

While existing probes are refreshed with a new probe version, any existing gene expression result records will no longer have a valid probe reference key. This issue can be solved either by reloading such result files to make new result records with reference keys to the refreshed probes.

Alternatively, if Flashback Archive is enabled for the database, any SQL query to retrieve probe annotation for such result records can use a flashback query using the "AS OF" clause with the creation date of the gene expression result.

4.13.6 Dual Channel Gene Expression Loader Errors

The Dual Channel loader verifies the existence of and looks up a W_EHA_ADF record with the relevant User Label. If the record is not found, the error is 'Getting the ROW_WID of the relevant W_EHA_ADF record, and checking its contents'. Subsequently, the ADF record is verified to match the Species ('The Species of the W_EHA_ADF record with user_label=… does not match the Species Name argument' error if it does not), and the DNA Reference Version ('The DNA reference version of the W_EHA_ADF record with user_label=… does not match the version argument', if it does not).

If the process of loading the data from the file into the external table fails, the error is 'Processing external data table'.

If processing the data from the external table and insertion into the result table fails, the error is 'Processing result records (w_eha_stg_2channel_gxp)'.

Other possible errors are: 'Verifying DNA reference version', 'Verifying result type', 'Verifying datasource name', 'Verifying species name', 'Verifying study name', 'Getting flanking offset', and 'Retrieving specimen id', 'Dropping external tables'.

4.13.7 RNA-seq Loader Errors

Only one external table is created for RNA Seq result data. If the process fails at this step, an error 'Processing external table' is logged into the log table.

When the external table processing for the entire RNA seq record set fails, an error 'Processing result records (w_eha_stg_rna_seq)' is logged.

Other possible errors are: 'Verifying DNA reference version', 'Verifying result type', 'Verifying datasource name', 'Verifying species name', 'Verifying study name', 'Getting flanking offset', 'Retrieving specimen number from external table', and 'Retrieving specimen id'.

4.13.8 Copy Number Variation Loader Errors

Only one external table is created for CNV result data. When the process fails while creating an external table, the error 'Processing external data table' is logged in the W_EHA_RSLT_ LOG table. When the external table processing for the entire CNV result record set fails, then an error 'Processing results records (w_eha_stg_copy_nbr_var)' is logged. If the process fails while dropping CNV external table, an error 'Dropping external table' is logged in the W_EHA_RSLT_ LOG table.

Other possible errors are: 'Verifying DNA reference version', 'Verifying result type', 'Verifying datasource name', 'Verifying species name', 'Verifying study name', 'Getting flanking offset', 'Retrieving specimen number from external table', and 'Retrieving specimen id'.

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

4.13.9 File Lineage Linker Errors

The File Lineage Linker can fail because either the parent or child file URI is not provided, or is invalid (there is no file record in W_EHA_FILE with this FILE_URI). The Linker will also fail if there are no common associated specimen for the two files being linked. However, if the "-force_link 1" argument is used, this error becomes a warning, and the files are linked.

4.13.10 Loader Runtime Error: ORA-01460 Unimplemented or Unreasonable Conversion Requested

Errors have been observed while running various ODB loaders. The loader run aborts prematurely with the following error message: *ORA-01460 unimplemented or unreasonable conversion requested*.

Oracle recommends applying an RDBMS patch to the TRC database that fixes this bug. See Oracle Support Bug 13099577 (ORA-1460 WHEN PARALLEL QUERY SERVERS ARE USED) available here https://mosemp.us.oracle.com/epmos/faces/BugDisplay?id=13099577 for details.

4.14 Collecting Oracle Optimizer Statistics

Oracle statistics is a collection of data of database objects such as tables and indexes and is required by Oracle optimizer to estimate the most efficient query execution plan. Missing or stale statistics can profoundly deteriorate query performance.

Oracle recommends gathering table and index statistics after a significant amount of data is loaded into a table. The statistics should be gathered after large bulk loads, most notably after reference tables are populated, but also after initial result runs. Later on, when a batch size becomes relatively small compared to the size of the already loaded data, statistics need not be gathered after each load. Instead, Oracle recommends gathering statistics on a weekly schedule basis. Statistics should be collected when major loading procedures are not running.

To collect statistics, connect to a database as ODB_SCHEMA owner using sqlplus and execute the command:

```
exec dbms_stats.gather_schema_stats ('ODB_', cascade=>true,estimate_
percent=>dbms_stats.auto_sample_size);
```

5
Model Dictionary

Refer to the *Oracle Health Sciences Omics Data Bank Electronic Technical Reference Manual* on My Oracle Support for all entities in all the tables of Oracle Health Sciences Omics Data Bank.

6

Use Case Examples

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

6.1 Overview of Use Cases

This section contains the following use case scenarios:

- Scenario 1—Find patients who are poor responders for drug A and have a mutation in the promoter region of gene A.
- Scenario 2—Show expression level of TP53 mutant by cancer tissue.
- Scenario 3—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 4—Ability to research a gene in the sample set.
- Scenario 5—Enable researcher to select a patient cohort based on the expression level for a set of genes.
- Scenario 6—Select mutation with deep functional annotation (for example, high impact based on PolyPhen algorithm).
- Scenario 7—I have a pathway. What mutations are present in the pathway and which study were they identified in (for example, what tumor types)?
- Scenario 8—What is the frequency of co-mutation of two genes in a data set?
- Scenario 9—Display all patients whose cancer cells had a deletion in gene X.
- Scenario 10—Find specimens with homozygous non-variants at the specified location (for example, rs12345 or chr1:13434).
- Scenario 11—Identify samples that have unacceptably low percentage of on-target reads, and exons that fall below threshold read depth. Filter variants with sufficient coverage and include only those that fall within a target region.

Note: To run some of the use case queries, you have to create global temporary tables.

6.2 Use Cases Accompanied by Query Examples

6.2.1 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

```
SELECT VRT.RESULT_SPEC_WID, CH.CHROMOSOME, VRT.START_POSITION, VI.REPLACE_
TAG

FROM W_EHA_RSLT_SEQUENCING vrt, w_eha_chromosome ch,

(

SELECT V.ROW_WID, SG.GENE_WID, V.REPLACE_TAG FROM

W_EHA_VARIANT v,

(

SELECT GSG.SOURCE_WID, GSG.GENE_WID, (GSG.START_POSITION - PP.PROMOTER_
OFFSET) as START_POSITION,

GSG.START_POSITION AS END_POSITION

FROM W_EHA_GENE G, W_EHA_GENE_SEGMENT GSG, W_EHA_PRODUCT_PROFILE PP

WHERE g.HUGO_NAME IN ('BRCA2')  /*-- Enter the approved HUGO symbols of target
genes here.*/

AND GSG.GENE_WID = G.ROW_WID

)SG

WHERE V.SOURCE_WID = SG.SOURCE_WID

AND (V.START_POSITION BETWEEN SG.START_POSITION AND SG.END_POSITION)

)vi, w_eha_rslt_study

WHERE VRT.VARIANT_WID = VI.ROW_WID

AND VRT.GENE_WID = VI.GENE_WID

AND W_EHA_RSLT_STUDY.RESULT_STUDY_NAME = 'STUDY1'

AND VRT.RESULT_STUDY_WID = W_EHA_RSLT_STUDY.ROW_WID

AND VRT.RESULT_SPEC_WID in (250);  /*-- Select a list of specimen patients who are
poor respondents of Drug A to test for mutation.*/
```

6.2.2 Scenario 2

Use Case - Show expression level of TP53 mutant by cancer tissue.

Areas

- Variant
- Gene Annotation
- Gene Expression
- Biospecimen Data

Output queries tables from - ODB+CDM

Query 2 -

```
SELECT RSLT_EXPR.RESULT_SPEC_WID, RSLT_EXPR.INTENSITY, RSLT_SEQ.START_
POSITION, RSLT_SEQ.REPLACE_TAG

FROM (SELECT VRT.RESULT_SPEC_WID, VRT.START_POSITION, V.REPLACE_TAG

FROM W_EHA_RSLT_SEQUENCING VRT, W_EHA_GENE G, W_EHA_VARIANT V

WHERE VRT.VARIANT_WID = V.ROW_WID

AND VRT.GENE_WID = G.ROW_WID

AND

G.HUGO_NAME IN ('TP53') /*-- Enter the Approved HUGO SYMBOL of target genes
here.*/

) RSLT_SEQ,

(SELECT R_EXP.RESULT_SPEC_WID, R_EXP.INTENSITY

FROM W_EHA_RSLT_GENE_EXP R_EXP

WHERE R_EXP.GENE_WID IN

(SELECT G.ROW_WID

FROM W_EHA_GENE G

WHERE G.HUGO_NAME IN ('TP53')) /*-- Enter the Approved HUGO SYMBOL of
targeted genes.*/

) RSLT_EXPR

WHERE (RSLT_EXPR.RESULT_SPEC_WID = 1 AND RSLT_SEQ.RESULT_SPEC_WID = 2);
```

6.2.3 Scenario 3

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 -

```
SELECT VRT.RESULT_SPEC_WID, VRT.START_POSITION, VI.REPLACE_TAG

FROM W_EHA_RSLT_SEQUENCING VRT, W_EHA_VARIANT VI, W_EHA_GENE G,

W_EHA_RSLT_SPECIMEN SP

WHERE VRT.GENE_WID = G.ROW_WID

AND g.HUGO_NAME IN ('KRAS') /*-- Enter the Approved HUGO SYMBOL of target genes here.*/

AND VRT.VARIANT_WID = VI.ROW_WID

AND VRT.RESULT_SPEC_WID = SP.ROW_WID

AND (SP.SPECIMEN_NUMBER in ('TCGA-02-0001-XXX-XXX')); /*-- Give a target list of specimens here.*/
```

6.2.4 Scenario 4

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

Areas

- Gene Annotation

Output queries tables from - ODB: *REFERENCE* + *RESULT*

Query -

```
SELECT VRT.RESULT_SPEC_WID, G.HUGO_NAME, VRT.START_POSITION, VI.REPLACE_
TAG

FROM W_EHA_RSLT_SEQUENCING VRT, W_EHA_VARIANT VI, W_EHA_GENE G

WHERE VRT.VARIANT_WID = VI.ROW_WID

AND VRT.GENE_WID = G.ROW_WID

AND g.HUGO_NAME IN ('BRCA2');
```

6.2.5 Scenario 5

Use Case - Enable researcher to select a patient cohort based on the expression level for a set of genes.

Areas

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

Output queries tables from - ODB+CDM

Query -

```
SELECT R_EXP2.RESULT_SPEC_WID, G.HUGO_NAME, R_EXP2.INTENSITY
```

from

```
W_EHA_RSLT_GENE_EXP r_exp2, W_EHA_GENE G,
```

```
(SELECT AVG(R_EXP1.INTENSITY) EXP_AVG
FROM W_EHA_RSLT_GENE_EXP R_EXP1
WHERE R_EXP1.GENE_WID IN
(SELECT G1.ROW_WID
FROM W_EHA_GENE G1
WHERE G1.HUGO_NAME IN ('TP53','BRCA1','GPR4','PABPC1','SOBP'))
) INTENSITY
WHERE R_EXP2.GENE_WID = G.ROW_WID
AND G.HUGO_NAME IN ('TP53','BRCA1','GPR4','PABPC1','SOBP')
AND R_EXP2.INTENSITY > INTENSITY.EXP_AVG
ORDER BY R_EXP2.RESULT_SPEC_WID;
```

6.2.6 Scenario 6

Use Case - Select mutation with deep functional annotation (for example, high impact based on PolyPhen algorithm)

Areas

- Variant
- Gene Annotation

Output queries tables from - ODB: REFERENCE

Query -

```
SELECT VG.REFERENCE_ID, VG.CODE_TYPE, VG.PREDICTION_SCORE, VG.CODE,
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, VP.PREDICTION_SCORE, PC.CODE, VP.STRUCTURE_
WID, PC.CODE_TYPE
FROM W_EHA_GENE_COMP_SEGMENT GCS, W_EHA_VARIANT V, W_EHA_VARIANT_XREF VX,
W_EHA_VARIANT_PREDICTION VP, W_EHA_PREDICTION_CODE PC
WHERE VP.VARIANT_WID = V.ROW_WID
AND VP.PREDICTION_CODE_WID = PC.ROW_WID
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
AND PC.CODE_TYPE IN ('SIFT', 'polyphen')
AND PC.CODE IN (deleterious, possibly damaging, probably damaging) /*-- Filter by
prediction code, confer W_EHA_PREDICTION_CODE table.*/
AND VP.PREDICTION_SCORE < '0.5' /*-- Filter by prediction score*/
)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

AND VG.STRUCTURE_WID = GC.STRUCTURE_WID;
```

6.2.7 Scenario 7

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

```
SELECT S.RESULT_STUDY_NAME, VRT.RESULT_SPEC_WID, VI.PATHWAY_NAME,
VI.PATHWAY_SOURCE_ID, VI.HUGO_NAME AS GENE_SYMBOL, VI.REFERENCE_ID,
VI.ABSOLUTE_POSITION, VI.CHROMOSOME, VI.REPLACE_TAG

FROM W_EHA_RSLT_SEQUENCING VRT, W_EHA_RSLT_STUDY S, (

SELECT V.ROW_WID, GS.GENE_WID, GS.HUGO_NAME, GS.PATHWAY_NAME, GS.PATHWAY_
SOURCE_ID, VX.REFERENCE_ID, V.ABSOLUTE_POSITION, V.CHROMOSOME, V.REPLACE_
TAG

FROM W_EHA_VARIANT V, W_EHA_VARIANT_XREF VX, (

SELECT G.ROW_WID AS GENE_WID, G.HUGO_NAME, P.PATHWAY_NAME, P.PATHWAY_
SOURCE_ID, GSG.START_POSITION, GSG.END_POSITION, GSG.SOURCE_WID

FROM W_EHA_GENE_SEGMENT GSG, W_EHA_GENE G, (

SELECT DISTINCT PH.PATHWAY_NAME, PH.PATHWAY_SOURCE_ID,PP.HUGO_SYMBOL

FROM W_EHA_PATHWAY_PROTEIN PP, W_EHA_PATHWAY PH

WHERE PP.PATHWAY_WID = PH.ROW_WID

AND PATHWAY_NAME LIKE ('%thyroid hormone%')

)P /*-- Select either a specific KEGG pathway or search for pathway name keywords.
For example, All Thyroid hormone specific pathways.*/

WHERE G.HUGO_NAME = P.HUGO_SYMBOL

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

) VI
```

```
WHERE VRT.VARIANT_WID = VI.ROW_WID
AND VRT.GENE_WID = VI.GENE_WID
AND VRT.RESULT_STUDY_WID = S.ROW_WID;
```

6.2.8 Scenario 8

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 -

```
select count(a_and_b.result_spec_wid) concurrent_cnt
FROM
(SELECT DISTINCT VRT.RESULT_SPEC_WID
FROM W_EHA_RSLT_SEQUENCING vrt, W_EHA_GENE g
WHERE VRT.GENE_WID = G.ROW_WID
AND g.HUGO_NAME IN ('KRAS')
intersect
SELECT DISTINCT VRT.RESULT_SPEC_WID
FROM W_EHA_RSLT_SEQUENCING vrt, W_EHA_GENE g
WHERE VRT.GENE_WID = G.ROW_WID
AND G.HUGO_NAME IN ('PTEN')
) a_and_b;
```

/* count all specimen with seq result */

```
SELECT COUNT(DISTINCT VRT_ALL.RESULT_SPEC_WID) CNT FROM W_EHA_RSLT_
SEQUENCING VRT_ALL;
```

6.2.9 Scenario 9

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

Areas

Output queries tables from ODB+CDM

Query

```
SELECT COUNT(*) FROM (
SELECT DISTINCT R_CNV.RESULT_SPEC_WID
From W_EHA_GENE G,
(SELECT R_CNV1.*
FROM W_EHA_RSLT_COPY_NBR_VAR R_CNV1
WHERE R_CNV1.RESULT_SPEC_WID BETWEEN (200 +ROUND(DBMS_RANDOM.VALUE(1,3)))
AND (200 +ROUND(DBMS_RANDOM.VALUE(5,12)))/*-- Restrict to a subset of the
specimen list. A random list is taken here.*/
```

```
AND R_CNV1.CALLED_CNV_TYPE = 'gain') R_CNV  /*-- Ensure W_EHA_RSLT_COPY_
NBR_VAR column; called_cnv_type, is updated with type data.*/

WHERE R_CNV.GENE_WID = G.ROW_WID

AND G.HUGO_NAME in ('KRAS'));/*-- Enter gene symbol of gene X.*/
```

6.2.10 Scenario 10

Use Case - Find specimens with homozygous non-variants at the specified location (for example, rs12345 or chr1:13434).

Areas

Output queries tables from

Query

```
select spec1.SPECIMEN_WID

From

(

select rnv.SPECIMEN_WID

from

(

select *

from

(

SELECT RS1.RESULT_SPEC_WID AS SPECIMEN_WID, RS1.*

FROM W_EHA_RSLT_NON_VARIANT_CHR RS1, W_EHA_VERSION V

WHERE

RS1.VERSION_WID = V.ROW_WID

AND V.VERSION_LABEL IN ('GRCH37.P8')

AND V.VERSION_TYPE = 'DNA'

AND RS1.ZYGOSITY IN ('hom-ref')

)

) rnv,

W_EHA_VARIANT_V VV1, W_EHA_VARIANT_XREF VX

WHERE

RNV.CHROMOSOME_WID = VV1.CHROMOSOME_WID

AND VX.VARIANT_WID = VV1.ROW_WID

AND VX.REFERENCE_ID = 'rs4733908'

and vv1.STATUS = 'KNOWN'

AND VV1.STRAND IN ('-','+')

AND VV1.ABSOLUTE_POSITION BETWEEN RNV.START_POSITION AND RNV.END_POSITION

) main1,
```

```
W_EHA_RSLT_SPECIMEN spec1

WHERE SPEC1.ROW_WID = MAIN1.SPECIMEN_WID

and spec1.SPEC_DATASRC_WID = 1;
```

6.2.11 Scenario 11

Use Case - Identify samples that have unacceptably low percentage of on-target reads, and exons that fall below threshold read depth. Filter variants with sufficient coverage and include only those that fall within a target region.

Areas

Output queries tables from ODB

Query

```
SELECT DISTINCT ODBQ3.SPECIMEN_WID VARIANT_SPECIMEN, ODBQ4.SPECIMEN_WID
RNASEQ_SPECIMEN

FROM

(

select spec1.SPECIMEN_WID

from

(

select rsq.SPECIMEN_WID

from

(

select *

from

(

select rs1.RESULT_SPEC_WID as SPECIMEN_WID, rs1.*

from W_EHA_RSLT_SEQUENCING_CHR rs1, W_EHA_GENE g, W_EHA_VERSION V

WHERE

RS1.GENE_WID = g.ROW_WID

AND g.HUGO_NAME i ('ADAM32')

AND RS1.version_wid = V.ROW_WID

AND V.VERSION_LABEL in ('GRCH37.P8')

AND V.VERSION_TYPE = 'DNA'

)) rsq,

(

select gs1.gene_wid, gcc1.start_position, gcc1.end_position

from W_EHA_GENE_STRUCTURE gs1,

(

select v.structure_wid, (v.start_position+s.start_position-1) as start_
position, (v.end_position+s.start_position-1) as end_position
```

```
from W_EHA_PROMOTER_REGION_V v, W_EHA_DNA_SOURCE s

where s.ROW_WID = v.SOURCE_WID

) gcc1

where gcc1.structure_wid = gs1.row_wid

union all

select gseg1.gene_wid, (gseg1.start_position + ds1.start_position-1) as
start_position, (gseg1.end_position + ds1.start_position-1) as end_
position

from W_EHA_GENE_SEGMENT gseg1, W_EHA_DNA_SOURCE ds1

where gseg1.source_wid = ds1.row_wid

) gco1,

W_EHA_VARIANT_V vv1

WHERE

rsq.VARIANT_WID = vv1.ROW_WID

and rsq.gene_wid = gco1.gene_wid

and rsq.start_position between gco1.start_position and gco1.end_position

and vv1.STATUS = 'KNOWN'

and vv1.STRAND in ('-','+')

) main1,

W_EHA_RSLT_SPECIMEN spec1

where spec1.ROW_WID = main1.SPECIMEN_WID

AND SPEC1.SPEC_DATASRC_WID = 1) ODBQ3,

(select spec2.SPECIMEN_WID

from

(

select rnaseq2.SPECIMEN_WID

from

(

select *

from

(

select rs2.RESULT_SPEC_WID as SPECIMEN_WID, rs2.*

from W_EHA_RSLT_RNA_SEQ_CHR rs2

WHERE

RS2.GENE_WID IN (71499,14775)

AND RS2.VERSION_WID IN (8,18)

)) rnaseq2,

(
```

```
select gs2.gene_wid, gcc2.start_position, gcc2.end_position

from W_EHA_GENE_STRUCTURE gs2,

(

select gc2.structure_wid, (gcs2.start_position + ds2.start_position-1) as
start_position, (gcs2.end_position + ds2.start_position-1) as end_position

from W_EHA_GENE_COMPONENT gc2, W_EHA_GENE_COMP_SEGMENT gcs2, W_EHA_DNA_
SOURCE ds2

where gc2.component_type in ('CDS','mRNA','miscRNA','exon')

and gc2.row_wid = gcs2.gene_component_wid

and gcs2.source_wid = ds2.row_wid

) gcc2

where gcc2.structure_wid = gs2.row_wid

) gco2

WHERE

rnaseq2.gene_wid = gco2.gene_wid

and (gco2.end_position >= rnaseq2.start_position

and gco2.start_position <= rnaseq2.end_position)

and rnaseq2.RAW_COUNTS < 15.0

and rnaseq2.RPKM > 0.25

and rnaseq2.STRAND in ('-','+')

) main2,

W_EHA_RSLT_SPECIMEN spec2

WHERE SPEC2.ROW_WID = MAIN2.SPECIMEN_WID

AND SPEC2.SPEC_DATASRC_WID = 1) ODBQ4

WHERE

(ODBQ3.SPECIMEN_WID = 21284 /*-- SpecimenID for Mutation results of a
patient*/

AND ODBQ4.SPECIMEN_WID = 20384); /*-- SpecimenId with Gene Expression results of
the same patient.*/
```

7

Miscellaneous Topics

This chapter contains the following topics:

7.1 Product Version and Product Profile Including Flanking Offsets

Currently, there are two configuration tables in the ODB schema:

1. **W_EHA_PRODUCT_VERSION** table stores product version numbers and is updated and used by upgrade and patch installations. This table records each patch or release that is installed. It does not have ETL_PROC_WID or ENTERPRISE_ID columns. It does not require a loader to be populated and is global to all enterprises or tenants that use the schema. Two fields are used to record version information:

   ```
   RELEASE_VERSION VARCHAR2(200);

   PATCH_VERSION VARCHAR2(200);
   ```

2. **W_EHA_PRODUCT_PROFILE** table stores information specific to each enterprise or tenant - a single record for each ENTERPRISE_ID value, and a unique key to enforce this. It also stores global settings for different logging levels allowed for log records inserted into the log table. It stores the global setting to permit DBMS output. Finally, there is a VCF_FORMAT field intended to store a list of data types that are used in the Format column in the VCF data file.

   The columns in this table are:

   - PROMOTER_OFFSET NUMBER;

- FLANKING_OFFSET NUMBER;
- LOG_WARNING CHAR(1);
- LOG_INFO CHAR(1);
- LOG_DEBUG CHAR(1);
- LOG_TRACE CHAR(1);
- LOG_DBMS_OUTPUT CHAR(1);
- VCF_FORMAT VARCHAR2(4000 CHAR);

The FLANKING_OFFSET parameter is an input (right after PROMOTER_OFFSET) on running TRC install scripts and is used by various loader procedures. It defines the size of the region before and after a gene definition that are to be linked with sequencing results. These associations are important as a lot of research is focusing on areas before and after gene definitions. The value for FLANKING_OFFSET should be equal to or greater than the value used for PROMOTER_OFFSET, so that the queries generated by the Query Engine can find results linked to a gene that may exist in a promoter region that extends before or after the gene definition.

THE LOG_% fields are flags whose values are either Y or N. By default, the WARNING and INFO log levels are turned on and DBMS output is turned off. For the most part these settings are not needed or used, as logging is also configured in the loader scripts.

The VCF_FORMAT column is required to specify data types used in the specification of custom FORMAT columns. For a description of its usage, see the Section 4.5.2, "Custom Format Specification in VCF".

7.2 Querying Database Cross-References for Variations

7.2.1 Ensembl db_xref Qualifier Issue

The Ensembl GVF file, which contains nucleotide variation references from dbSNP, COSMIC and EMBL, is used to populate the variation tables in the Omics Data Bank. The cross-reference information for these variants is identified by the Dbxref qualifier and is loaded into the W_EHA_VARIANT_XREF table. The standard format for Dbxref in a GVF file is:

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

To import this data, the program splits it into DATABASE and REFERENCE_ID using the first colon (:) as delimiter. Therefore, for the above example W_EHA_VARIANT_XREF populates columns with the following data:

```
DATABASE = 'dbSNP_132'
```

```
REFERENCE_ID = 'rs79772382'
```

However, for some organisms, Dbxref is defined differently. Following is an example from a Rattus norvegicus GVF file:

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

For such cases, W_EHA_VARIANT_XREF columns are populated with the following data:

```
DATABASE = 'ENSEMBL'
```

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

Since REFERENCE_ID may sometimes contain suffixed or prefixed data, Oracle recommends using the SQL `LIKE` operator when querying the REFERENCE_ID.

> **Note:** The REFERENCE_SUFFIX column in W_EHA_VARIANT_XREF is not populated with any data in the current model.

The same scenario hold good for the SwissProt database cross-reference. Oracle recommends using the SQL `LIKE` operator for querying against the W_EHA_PROT_XREF table.

7.2.2 Swissprot db_xref Qualifier Issue

The W_EHA_PROT_XREF table stores the database cross-reference information for SwissProt. This table populates the DATABASE, REFERENCE_ID and REFERENCE_SUFFIX information. The standard format for database cross-reference in a SwissProt file is:

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

To import this data, the program splits it into DATABASE, REFERENCE_ID and REFERENCE_SUFFIX using the semi-colon (;) as delimiter. Hence, for the above example W_EHA_PROT_XREF, populate the columns with following data:

```
DATABASE = 'InterPro'

REFERENCE_ID = 'IPR007031'

REFERENCE_SUFFIX = 'Poxvirus_VLTF3'
```

There are certain cross-references in SwissProt file that have the same REFERENCE_ID but a different REFERENCE_SUFFIX. For indexing the REFERENCE_ID, only the distinct first found REFERENCE_ID is stored. The other records retain the same REFERENCE_ID.

For example:

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

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

In the above example, the REFERENCE_ID for both the cross-references is AL390732. Therefore, only the first line information is stored in the table for indexing.

There is some loss of information on the REFERENCE_SUFFIX level but not on REFERENCE_ID. All REFERENCE_IDs are captured in the W_EHA_PROT_XREF table. Also, the count of instances where duplicate db_xrefs are not saved is now logged by the SwissProt loader as a warning.

7.2.3 W_EHA_VARIANT_X

The W_EHA_VARIANT_X table stores records for the following scenarios for both VCF and MAF format files:

- When `SVTYPE` tag in the INFO column and < tag in the ALT column of the VCF file are present (for example, in case of large SV, there is some information on the SVTYPE in the INFO column such as `SVTYPE=INS` or `SVTYPE=DEL`, and so on); for such records the `ALT` column will have tags like `<DEL>` or `<DUP>`, and so on.

- When the combined length of REF and ALT sequence is greater than or equal to 999, post trimming of overlapped bases. The checksum is calculated and reported in the W_EHA_VARIANT.SEQUENCE_CHECKSUM column.

  The ALLELE column in the W_EHA_VARIANT_X table stores the complete value present in the ALT column.

  If there is any overlap between the REF and ALT columns and some bases are trimmed from ALT column, then the ALLELE column still stores the untrimmed value. The checksum (stored in W_EHA_VARIANT.SEQUENCE_CHECKSUM) is calculated based on the trimmed ALT value.

  To determine the position of the actual trimmed allele, W_EHA_VARIANT_X.START_POSITION value is used. If none of the bases are trimmed from the ALT column, then the START_POSITION value is 1. If first base is trimmed, then the START_POSITION value of the sequence existing in the ALLELE column is 2.

7.3 Mitochondrial Chromosome Mappings

The references to mitochondrial chromosome are stored as MT on the reference side in the W_EHA_VARIANT and W_EHA_HUGO_INFO tables, and on the result side in the W_EHA_CHROMOSOME tables of the model. Any novel variants reported into the W_EHA_VARIANT table from the result files have the chromosome value converted from M to MT. For example, while inserting in W_EHA_RSLT_COPY_NBR_VAR or W_EHA_RSLT_SEQUENCING, the FK to W_EHA_CHROMOSOME table for chromosome value MT is taken if the result file has chromosome M.

7.4 Promoter Offset

Promoter region information is not available in the reference data set imported from Ensembl EMBL files. Therefore, a column has been provided in the W_EHA_SPECIES table to specify the promoter region upstream to the gene for a specific organism. The column is named PROMOTER_OFFSET. This Promoter Offset is species-specific.

There is also a default Promoter Offset, stored in the PROMOTER_OFFSET column of the PRODUCT_PROFILE table. This default value is populated during ODB installation, from the value of the **-promoter_offset** argument.

If the Promoter Offset for a species is null in the W_EHA_SPECIES table, the default value from W_EHA_PRODUCT_PROFILE is used. The EMBL and SwissProt reference loaders, which typically create W_EHA_SPECIES records, do not populate the PROMOTER_OFFSET column there. Therefore, the only way to define a non-default Promoter Offset for a species, after its record has been created, is to manually edit the value in W_EHA_SPECIES.PROMOTER_OFFSET for it.

ODB provides a special view, W_EHA_PROMOTER_REGION_V, which uses the Promoter Offset described above to enable querying promoter regions for genes or gene structures in ODB.

7.5 Loader Activity Logging

The W_EHA_PRODUCT_PROFILE logging settings are singular levels, where each setting affects one level. These logging settings are generally not used by loaders as logging is configured in the loader scripts where the log levels are inherited - for example, if `TRACE` is on, `DEBUG` is also on. A single ETL_PROC_WID is used for each loader run. Any log records associated with that particular load can be looked up in the W_EHA_RSLT_LOG table, after the fact, using this ID.

The following named command-line option, present in all loaders, sets log level for that loader. `-log_level [TRACE | DEBUG | INFO | WARNING | ERROR] default: INFO` When `-log_level` is set to any other value, except the five listed above (for example, `INHERIT`), logging levels are inherited from W_EHA_PRODUCT_PROFILE.LOG_% settings. The following named command-line argument logs a summary log record to the database and prints the summary to the console.

```
-print_summary ['1'=on, '0'=off] default: '0' if on
```

Table 7–1 Log Events recorded in the W_EHA_RSLT_LOG table

Log Event	Modifiable by User	Log Event Description
START	No, always on	Displays the PLSQL package, operation and parameter list
END	No, always on	Displays the status of PLSQL operation
SUMMARY	Yes	Displays a summary report
FATAL	No, always on	Displays exception messages with error code and error trace
ERROR	No, always on	Displays exception messages with error code and error trace
WARNING	Yes	Displays warning messages
INFO	Yes	Displays informational messages such as sql%rowcounts
DEBUG	Yes	Displays debug messages
TRACE	Yes	Displays the finest grade debug messages

Table 7–2 How to enable log events using batch file parameters

Batch File Parameter Name	Batch File Parameter Value	WARNING Log Events Captured	INFO Log Events Captured	DEBUG Log Events Captured	TRACE Log Events Captured	SUMMARY Log Events Captured
log_level - - - - -	WARNING	Yes	No	No	No	-
	INFO	Yes	Yes	No	No	-
	DEBUG	Yes	Yes	Yes	No	-
	TRACE	Yes	Yes	Yes	Yes	-
	INHERIT	Each log event can be individually turned on or off by setting the corresponding attribute in the W_EHA_PRODUCT table				-
print_ summary -	1	-	-	-	-	Yes
	0	-	-	-	-	No

7.6 User Feedback for Loader Runs

User feedback for loader runs is provided in the form of summary log records that are printed at the console before the loader run terminates. The same summary is inserted as a log level *SUMMARY* record in the W_EHA_RSLT_LOG table. A summary feedback can be created for every ETL load by using its ETL_PROC_WID value.

The following function can be called anytime after a loader run to create a SUMMARY record in database, for instance if it was not run time of load:

```
odb_util.print_loader_summary(etl_proc_wid);
```

Following is an example of a loader summary output after a successful loader run:

Loader Summary

Command Used

ODB_RSLT_DUAL_CHANNEL_UTIL.process_dual_channel(i_data_file => 'unc.edu__AgilentG4502A_07_3__TCGA-A2-A0CX-01A-21R-A00Z-07__gene_expression_analysis_summary.txt', i_data_directory => 'ODB_LOAD', i_species_name => 'Homo sapiens', i_study_name => 'STUDY1', i_datasource_name => 'CDM', i_specimen_number => 'rna001', i_specimen_vendor => 'rnaseq', i_control_specimen => 'Stratagene Univeral Reference', i_user_label => 'ADF_p7', i_reference_version => 'GRCh37.p7', i_file_flg => 'E', i_preprocess_dir => null, i_preprocess_file => null, i_data_file_path => null, i_dbfs_store => null, i_alt_file_loc => null, i_read_size => null)

Properties

Start Date: 2013-06-07 15:36:41

ETL Process ID: 4116

Exadata: False

DB User: ODB25RES

OS User: vkamath

Hostname: NORTH\MUWKS0015

File Name: unc.edu__AgilentG4502A_07_3__TCGA-A2-A0CX-01A-21R-A00Z-07__gene_expression_analysis_summary.txt

File Type Code: 2-Channel Expression

File Type Name: Agilent TCGA 2-channel Expression analysis file

File Type Version: A

End Date: 2013-06-07 15:36:42

Status: SUCCESS

Insert Summary

---

w_eha_rslt_2channel_gxp: 61 rows

W_EHA_STG_2CHANNEL_GXP: 61 rows

w_eha_file: 1 row

w_eha_rslt_file_spec: 1 row

w_eha_file_qlfr: 0 rows

w_eha_rslt_specimen: 0 rows

Error Summary

Warning Summary

Info Summary

2013-06-07 15:36:41

Detail: Found specimen number=rna001

7.7 VCF Loader Log

- **Some variants are skipped from loading**:

  If EMBL reference data load is not complete, it might result in insufficient information for a chromosome or a chromosome region in the W_EHA_DNA_ SOURCE table. In such cases, a reference is not available for a given region, so the variants falling in these regions are not loaded from VCF file.

  These variants are shown in the summary report with the following warning message:

```
---------------
Warning Summary
---------------
2013-11-26 10:30:53
Detail: Some variants were skipped, check chromosome and genomics coordinates);
[rowcount=16]
```

- **Some specimen numbers in the VCF file in CDM datasource are not found**:

  In a VCF file with multiple specimens, if some of the specimens are not available in the CDM tables, a warning message is logged to indicate how many specimens are found in CDM.

  In the W_EHA_RSLT_LOG.RECORD_DETAIL column, a warning message stating number of specimens found out of total number of specimens in the file is shown. Following is an example:

```
returning 1664; [status=warning 2 of 3 were found, elapsed_
time=00:00:19]
```

  In the summary report, a warning message stating which specimen is not found is shown. Following is an example:

```
Detail: Could not find specimen number=HG00097
```

7.8 Creating Custom Gene Components

The end user can build queries by selecting the Variants or CNVs in regions, defined by Gene Components (the W_EHA_GENE_COMPONENT and associated tables). An example of this is selecting the Exon gene region, which is one of the gene component types provided by Ensembl.

You can add custom gene components to these tables, use custom gene components types and so on. These new gene component types can be made available in the TRC v2.0.1 (or higher) UI. To do this, you must understand the structure and functional use of these tables in the ODB schema.

The heart of the subsystem is the W_EHA_GENE_COMPONENT table, which stores one record per gene component. It has the following important columns:

- ROW_WID - a surrogate primary key (PK), filled from the W_EHA_GENE_ COMPONENT_S sequence.
- STRUCTURE_WID - a foreign key (FK) to the W_EHA_GENE_STRUCTURE table. This is a mandatory column which links the component to its gene. If a gene component is associated with a known transcript with an Ensembl Transcript ID (for example, ENST00000384612), this FK should point to a gene structure with

this TRANSCRIPT_ID. Otherwise, it points to a gene structure record for the same gene with TRANSCRIPT_ID = NO STRUCTURE (a catch-all gene structure).

- COMPONENT_TYPE - identifies the component as belonging to a type. It is of type VARCHAR2(100) and not normalized, so you must use consistent casing for custom component types. Ensembl currently provides components of the following types - exon, CDS, STS, mRNA, misc_RNA, misc_feature.
- PROTEIN_WID - an optional FK to the W_EHA_PROTEIN table. Setting this parameter lets you query by Pathway name, and Genes and Gene Sets.

The second table in the gene component subsystem is W_EHA_GENE_COMP_ SEGMENT, which is used to map the component onto a DNA source and through it onto a chromosome. There can be one or more records per gene component in this table. There are the following important columns in this table:

- ROW_WID - a surrogate PK, filled from the W_EHA_GENE_COMP_SEGMENT_S sequence.
- GENE_COMPONENT_WID - the FK to the W_EHA_GENE_COMPONENT table.
- SOURCE_WID - the FK to the W_EHA_DNA_SOURCE table.
- NUMBER_IN_SEQUENCE - if there are multiple segments per component, their order is defined in this column.
- START_POSITION and END_POSITION - the start and end positions of the segment relative to the W_EHA_DNA_SOURCE record (adding the W_EHA_ DNA_SOURCE.START_POSITION value we get the absolute positions on the chromosome).
- COMPLEMENT- a 0 or 1 flag. If the value is 1, the segment is a complement to the DNA Source sequence.

Finally, there are the W_EHA_GENE_COMP_XREF and W_EHA_GENE_COMP_ QLFR tables, which have the same structure as all other XRef and Qualifier tables. These tables contain optional Database Reference and Qualifiers (such as notes) for the gene components (0 to many per component).

Currently, these tables contain data loaded from Ensembl EMBL files (by the EMBL Reference Loader). However, you can add your own data to the tables after loading the reference data from the Ensembl EMBL and SwissProt files. You can also define customer-specific component types and use them.

You can define a new component type by choosing the value. However, ensure that it does not coincide with any existing or future values from other sources. Since there is no way to know what component types can be added to Ensembl or other sources in the future, Oracle recommends using a customer-specific prefix. For example, to add a "First exon in a gene component" type, and the customer company name is XYZ, use XYZ_first_exon as the new component type. You can use it with the same letter casing for all first exons of genes that we add.

Next, provide the actual components. A superset of the data (exons) is already present in the W_EHA_GENE_COMPONENT table, with their segments already defined, and everything correctly linked to DNA Sources, Gene Structures, and Proteins. An extra set of records is required in W_EHA_GENE_COMPONENT, copied from some of the existing exon records, with the COMPONENT_TYPE = "XYZ_first_exon" instead of "exon", and the new segments for them. Inserting these records requires SQL or PL/SQL code, processing the pre-existing data, and inserting new records. The logic in the code is as follows:

- For each W_EHA_GENE_COMPONENT record with COMPONENT_TYPE of `exon`, determine (by its position relative to the other exons for the same gene, if any) if this is the first exon in its gene.
- If it is, insert a new record into W_EHA_GENE_COMPONENT, with the same values, except that the COMPONENT_TYPE is changed to "XYZ_first_exon" and a new ROW_ID obtained from the sequence.
- Whenever a new component is inserted, insert copies of associated W_EHA_GENE_COMP_SEGMENT records, replacing their GENE_COMPONENT_WID values with the ROW_ID of the new W_EHA_GENE_COMPONENT record.

You must archive the script used to create these components, as it may be needed for re-creating the data in another database, re-installing the same database, or after loading another version of Ensembl reference files.

The previous example was that of copying the already existing components. For the general case, perform the following steps:

After selecting an existing or new Component Type (the same way as described above), ascertain the following for each gene component to be inserted:

1. Whether the gene (already existing in the W_EHA_GENE table) is associated with any information that uniquely identifies it: for example, W_EHA_GENE.ROW_WID or Ensemble Gene Id, or Species + HUGO Name.
2. Whether it is associated with any transcript for this gene (either a W_EHA_GENE_STRUCTURE.ROW_WID or an Ensembl Transcript ID, or not associated).
3. Whether it is associated with any existing protein record (in the W_EHA_PROTEIN table).
4. What range or ranges of chromosomal positions it maps to (chromosome name, start and end positions on the chromosome, complement or not). If there is more than one range, multiple segments should be created.
5. Whether you want to enter any DB Xref and (or) Qualifier entries for each component.

Once this information is gathered, you can create a script (for example, in PLSQL) to insert the new gene components. At the beginning of the script, get a new ETL_PROC_WID from the W_EHA_ETL_PROC_S sequence and use it in each subsequent `INSERT` statement for all tables. Then, looping over the components, perform the following steps:

1. Find the ROW_WID of the W_EHA_GENE_STRUCTURE that the new component will be linked to. This record must belong to the gene the component is associated with, and its TRANSCRIPT_ID must match the Ensembl Transcript ID of the new component, or be NO STRUCTURE if there is none. If there is no such record in W_EHA_GENE_STRUCTURE, it should be inserted and its ROW_WID used as the STRUCTURE_WID in step 3 (if it is found, use its ROW_WID as STRUCTURE_WID).
2. If the new component is linked to a W_EHA_PROTEIN record, find its ROW_WID, and use it as PROTEIN_WID in the next step. Otherwise, use NULL.
3. Insert a new record for the component into W_EHA_GENE_COMPONENT, getting a new ROW_WID for it from the W_EHA_GENE_COMPONENT_S sequence, and using the other values obtained above, and the chosen value for COMPONENT_TYPE. Store the new ROW_WID.
4. If there are DB XRef and Qualifier entries for the new gene component, insert them into the respective tables, using the appropriate sequences for the PKs and the new

W_EHA_GENE_COMPONENT.ROW_WID as the FK to the W_EHA_GENE_COMPONENT record.

5. Create the Gene Component Segments. For each chromosomal range of the new component, you must find the W_EHA_DNA_SOURCE record that the range is contained in. You can do this easily as the DNA Sources are mapped to chromosomes except when the component range spans two (or more) DNA Sources. In this case, it has to be broken up into sub-ranges, each mapped to its own DNA Source.

   There may be more than one matching DNA Source and you are required to select one of these. The selection is rather arbitrary; a reasonable rule is to select the longest of the matching DNA Sources. Once the DNA Sources for all ranges are selected, use their ROW_WIDs to recalculate the range start and stop positions from the chromosomal to the DNA Source coordinates.

   Insert a new record for each range into W_EHA_GENE_COMP_SEGMENT table, using the ROW_WID of the component record inserted into W_EHA_GENE_COMPONENT as GENE_COMPONENT_WID, and getting the new ROW_WID value from the W_EHA_GENE_COMP_SEGMENT_S sequence.

   Note: The START_POSITIONs and END_POSITIONs of all segments and the DNA Sources themselves are 1-based, so when recalculating, use -1's where appropriate.

This outlines the process of adding new gene components and custom gene component types.

There is one extra step needed for each new gene component type to appear in the Query UI. It is necessary to add one or more records to the TRC_LOOKUP_CODE table in the Application schema (not the ODB schema). Enter the component type you added in ODB (for example, XYZ_first_exon) as CODE, TRC_QP_GENE_REGION as CODE_TYPE, the label you want in the UI for this Gene Region (for example, first exon) as CODE_NAME, an optional description as CODE_DESC, and the language or locale code (for example, en_US) as LANGUAGE_CODE. If the installation supports several languages, insert a separate record for each.

Following is information that will be useful when creating custom gene components. When inserting into a table, get the new ROW_WID from the appropriate sequence:

Table 7–3 Tables and Sequences

Table	Sequence to get the ROW_WIDs from
W_EHA_GENE_COMPONENT	W_EHA_GENE_COMPONENT_S
W_EHA_GENE_COMP_SEGMENT	W_EHA_GENE_COMP_SEGMENT_S
W_EHA_GENE_COMP_XREF	W_EHA_GENE_COMP_XREF_S
W_EHA_GENE_COMP_QLFR	W_EHA_GENE_COMP_QLFR_S
W_EHA_GENE_STRUCTURE	W_EHA_GENE_STRUCTURE_S
TRC_LOOKUP_CODE (column: CODE_ID)	S_ROW_ID_SEQ

For example, select W_EHA_GENE_COMPONENT_S.NEXTVAL from DUAL;

ETL_PROC_WID NUMBER(38), is a column in all ODB tables that should be filled using insertion scripts. For every run of a script, a new unique value should be obtained from the sequence W_EHA_ETL_PROC_S, stored in a script variable, and

used whenever the script inserts a record or records into any ODB table. During insertion the current date and time should also be filled (using select SYSDATE from DUAL) in W_INSERT_DT DATE column of each table.

DNA Sources and Chromosomes

In ODB, Gene Component Segments are not mapped directly to Chromosomes, but rather to DNA Sources. The DNA Sources are stored in the W_EHA_DNA_SOURCE table and contain the actual genomic DNA sequence, an entire chromosome or about 1,000,000 bases per DNA Source, depending on the Ensembl version.

When inserting a new Gene Component, its Segments must be mapped to one or more DNA sources. For example, if we insert a new Gene Component with a single chromosomal Segment, spanning positions from 1450000 through 1460000 on Chromosome 1. We know that the species PK is 1. We must find the DNA source record, or records, covering this range.

The following SQL finds the W_EHA_DNA_SOURCE record the segment should be assigned to:

```
select min(ds.row_wid) from w_eha_dna_source ds

where ds.species_wid = 1

and ds.chromosome = '1'

and ds.start_position <= 1450000

and 1460000 <= ds.end_position
```

This works if the entire segment CHR1:1450000-1460000 fits into a single DNA source range. Instead if it spans across 2 or more DNA Source ranges, the query will not have a result. A more complex query must be used to get the partial DNA Sources. However, this is very unlikely to happen for Ensembl versions 67 or higher, where a DNA Source typically covers an entire chromosome.

Once you get the DNA source, the Segment can be mapped on it, with the following recalculation of the start and end positions:

```
W_EHA_GENE_COMP_SEGMENT.START_POSITION = 1450000 - W_EHA_DNA_SOURCE.START_
POSITION + 1
```

and

```
W_EHA_GENE_COMP_SEGMENT.END_POSITION = 1460000 - W_EHA_DNA_SOURCE.START_
POSITION + 1
```

7.8.1 Creating Custom Gene Region Views

The end user can build queries by selecting Variants or CNVs in regions defined in gene region views. An example of such a view is W_EHA_PROMOTER_REGION_V (selectable in the UI as Promoters). Now, TRC also supports custom gene region views, which can be created at customer sites and are specific to these sites.

Oracle recommends using custom gene region views only where calculated positions are involved (and not to use them instead of custom Gene Components). Using custom Gene Components does not significantly increase the size of the query SQL, but using each custom view in a query increases the SQL size by up to 500 characters.

Following are the guidelines for creating custom views:

The custom gene region views must be named following the ODB view naming conventions — the name must start with the W_EHA prefix and end with the _V suffix

(indicating that it is a view). Use the standard Promoter Region view as an example - W_EHA_PROMOTER_REGION_V.

A gene region view must have the following columns:

- STRUCTURE_WID (NUMBER) - an FK to the W_EHA_GENE_STRUCTURE table. This is a mandatory column, and is very important as it ultimately links the region to its gene. If the region is associated with a known transcript with an Ensemble Transcript ID (for example, ENST00000384612), this FK should point to a gene structure with this TRANSCRIPT_ID. Otherwise it points to a gene structure record for the same gene with TRANSCRIPT_ID = NO STRUCTURE (a catch-all gene structure).

  There can be multiple W_EHA_STRUCTURE records for the same gene, if the gene has multiple transcripts, and the catch-all structure record for the gene must be used for all gene regions not associated with transcripts.
- PROTEIN_WID (NUMBER) - an optional FK to the W_EHA_PROTEIN table. If this is set, you can query by Pathway name, Genes and Gene Sets.
- COMPLEMENT (NUMBER) - a 0 or 1 flag. If the value is 1, the region is a complement versus the DNA Source sequence.
- SOURCE_WID (NUMBER) - the FK to the W_EHA_DNA_SOURCE table. All genomic positions in ODB are relative to W_EHA_DNA_SOURCE records, not chromosomes, as DNA sequences are stored in this table.
- START_POSITION and END_POSITION (both numbers) - the start and end positions of the region relative to the W_EHA_DNA_SOURCE record (adding the W_EHA_DNA_SOURCE.START_POSITION value we get the absolute positions on the chromosome).
- The standard view in the ODB schema, W_EHA_PROMOTER_REGION_V, can be used as an example when creating custom gene region views. It is used in exactly the same way as the custom views are. The view calculates the estimated promoter position for a gene by using a standard offset from the start of the gene's first CDS. Other standard views that can be used as examples, are W_EHA_5P_FLANKING_REGION_V and W_EHA_3P_FLANKING_REGION_V.

To make a new custom view available in the TRC UI, perform the following steps:

- A select privilege on the view must be granted to the TRC schema and the appropriate user roles, for example:

```
GRANT SELECT ON ODB.W_EHA_MY_CUSTOM_REGION_V TO OMICSDATAMARTADMIN;

GRANT SELECT ON ODB.W_EHA_MY_CUSTOM_REGION_V TO OMICSDATAMARTCONTRIBUTOR;

GRANT SELECT ON ODB.W_EHA_MY_CUSTOM_REGION_V TO OMICSDATAMARTUSER;

GRANT SELECT ON ODB.W_EHA_MY_CUSTOM_REGION_V TO TRC;
```

(run these commands as SYSTEM and customize the view name and the ODB and TRC schema names, if necessary).

- A synonym for the view created in the Application (TRC) schema - it should have the same name as the view itself, for example:

```
CREATE SYNONYM TRC.W_EHA_MY_CUSTOM_REGION_V FOR ODB.W_EHA_MY_CUSTOM_
REGION_V;
```

- The view must be registered in the TRC Application's seed data, as follows:

add one or more records for it to the TRC_LOOKUP_CODE table in the Application (TRC) schema (not the ODB schema).

Enter `CVIEW_ || <view name>` as CODE (for example, if the view is named W_EHA_MY_CUSTOM_REGION_V, enter CVIEW_W_EHA_MY_CUSTOM_REGION_V), TRC_QP_GENE_REGION as CODE_TYPE, the label you want in the UI for this Gene Region (for example, Upstream region) as CODE_NAME, an optional description as CODE_DESC, and the language or locale code (for example, en_US) as LANGUAGE_CODE. If the installation supports several languages, insert a separate record for each.

The CVIEW_ prefix to the Code is necessary for the Query Engine to correctly identify the code as a custom gene region view.

7.8.2 Comparing Genomic Coordinates to Reference

For all result data loaded in the result tables, the given genomic coordinates are chromosome position specific, while the coordinate values (stored in the columns START_POSITION, END_POSITION) in the reference tables, with the exception of W_EHA_DNA_SOURCE, are specific to the source sequence stored in the W_EHA_DNA_SOURCE table. The genomic coordinates of this table are chromosome position specific.

Reference positions are counted from the first base with the positional base inclusive. This results in a single base increment when compared to other genomic coordinate systems. This should be taken into account when matching coordinates from result tables to reference table values.

Following is an example code for discovering overlapping regions of CNV against gene segments in the reference:

```
SELECT CNV.ROW_WID, GS.GENE_WID FROM W_EHA_RSLT_COPY_NBR_VAR CNV, W_EHA_
GENE_SEGMENT GS, W_EHA_DNA_SOURCE DS, W_EHA_CHROMOSOME CH

WHERE CH.ROW_WID = CNV.CHROMOSOME_WID

AND DS.CHROMOSOME = CH.CHROMOSOME

AND GS.SOURCE_WID = DS.ROW_WID

AND (CNV.START_POSITION <= (GS.END_POSITION + DS.START_POSITION -1)

AND (CNV.END_POSITION >= (GS.START_POSITION + DS.START_POSITION -1))
```

7.9 Additional Step on Exadata versus Non-Exadata

All loaded result files are stored in staging tables on Exadata because direct path loading is required to get the best compression. The script, load_exadata.sh, triggers the SQL scripts required to load the data into result tables, including chromosome partitioned tables (suffixed *_CHR*). This script accepts a parameter which inquires the degree of parallelization for queries that insert data.

For each staging table, the script locks the staging table and loads into the result table directly. 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 actual result table. For better compression run the script after a minimum of 50 sample result data have been loaded into the staging tables.

The load_exadata script takes the following parameters:

- Username - to connect to the ODB schema. If using an Oracle Wallet this value is not used.
- DBNAME or TNS NAME - WALLET NAME if using a wallet

- Number for degree of parallelization - number which varies based on the Exadata machine type - quarter rack, half rack or a full rack. Consult the system administrator to evaluate and monitor resource capabilities to come up with the appropriate number.
- 1 for running script with Oracle Wallet, or 0 for running without Wallet.

7.10 UNDO Tablespace Auto-extendable Issue

Oracle DB `UNDO` tablespace should be auto-extendable. For Exadata systems the following error can occur on running Variant loaders:

```
Description: ORA-30036: unable to extend segment by 8 in undo tablespace
'UNDOTBS8'
```

Ensure that the undo tablespace is increased or is created as Big File, with maxsize around 120G (with an option to increase its size, if required).

7.11 Chromosome Partitioned Tables

Chromosome partitioned tables have identical columns as the existing result tables. The partitioning strategy uses a primary range or interval partition based on CHROMOSOME_WID. There is a subpartition using a range on the START_POSITION. This range partition uses a subpartition template that declares new partitions for every 1 million bases. This type of subpartition enables database administrators to easily add new partitions to the range, if any subpartition gets too large.

The Exadata version of ODB uses a secondary step to move results from staging tables in order to maximize the hybrid columnar compression used on all result tables. This *load exadata* task now has SQL statements to move staging data to each of the result tables, the specimen based partitioned table and the chromosome based partitioned tables as well.

On non-Exadata installations, there is no secondary task to move data from staging tables since hybrid columnar compression does not exist on Oracle 11g non-Exadata versions. There are indexes declared based on CHROMSOME_WID and START_POSITION to give some better performance. There are views declared for non-Exadata that let application code, referencing the chromosome based result tables, to work seamlessly on any database platform.

TRC currently only queries the chromosome based partitioned tables for file export. When users specify a genomic range to export, the SQL used to export this data dynamically uses the chromosome based partition tables for VCF (variant) and SEG (copy number variant) export files. For RES (gene expression) and GCT (dual channel gene expression), there are no chromosome based partitioned result tables since all results are always linked to specific genes. Any export that uses a genomic range for RES or GCT is used to find all genes in that range. For non-Exadata, all types of export always finds the list of genes to export based on genomic ranges. This is done primarily because chromosome partitioning only exists for the Exadata platform.

A

Additional Result Tables

The appendix contains the following topics:

A.1 Pre-Seeded Tables

W_EHA_RSLT_TYPE

This table stores information regarding type of result stored and is based on what data is being inserted into ODB (one row inserted per loader per file). User may choose to seed more types.

Table A–1 W_EHA_RSLT_TYPE

RESULT_TYPE_NAME	RESULT_TYPE_DESC	Which loader inserts
GENE_EXPRESSION	Gene expression results	Gene expression loader
NOCALL	Nocall result for sequencing given allele	CGI masterVar loader
SEQUENCING	Sequencing results including simple variants such as SNP, insertions, deletions	VCF, MAF, CGI masterVar[1] loaders
COPY_NUMBER_ VARIATION	Copy Number Variation results	CNV loader
TCGA_RNA_SEQ_ EXON	TCGA RNA Seq results for exon information	TCGA RNA seq loader[1]
2-CHNL_GENE_ EXPRESSION	Gene expression results from 2-channel gene expression analysis	Dual channel loader

[1] CGI masterVar loader is temporarily removed from ODB 3.0 and will be available in the next release.

W_EHA_FILE_TYPE

This table is pre-seeded with file types currently handled by loaders, it should contain six rows and the pre-seeded values are mentioned in the following table:

Table A–2 W_EHA_FILE_TYPE

FILE_TYPE_CODE	FILE_TYPE_NAME	FILE_TYPE_DESC	FILE_TYPE_VERSION
VCF	Variant Call Format	File containing variant information including SNPs, insertions, and deletions.	4.1
MAF	Mutation Annotation Format	Mutation Annotation Format containing snps, inserts, and deletions.	2.2
Tab-delim Expression	Tab delimited Expression file	Gene Expression tab delimited file format containing probe hybridization results, 3 values per hybridization: Intensity, Call, P-value.	A
CGI masterVar	Complete Genomics MasterVar	Master Variation file from Complete Genomics containing SNPs, insertions, deletions, and no-call information.	2.0
SIFT	Sorting Tolerant From Intolerant	SIFT predicts whether an amino acid substitution affects protein function.	4.0
PolyPhen	Polymorphism Phenotyping	PolyPhen predicts possible impact of an amino acid substitution on the structure and function of a human protein using straightforward physical and comparative considerations.	5.0
BAM	Binary Alignmentor Map Format	Sequencing alignment file for sequencing runs.	1.4
SAM	Sequence Alignmentor Map Format	Sequencing alignment file for sequencing runs.	1.4
TCGA RNA SEQ EXON	TCGA RNA SEQ EXON	TCGA RNA Seq tab delimited file format for exon information.	3.1.4.0
CNV_SEG	CNV .seg file	Segmented data file format is the output of the Circular Binary Segmentation algorithm (Olshen et al., 2004)	1.0
2-Channel Expression	Agilent TCGA 2-channel Expression analysis file	TCGA's Agilent platform Gene Expression analysis file format containing gene level results; Log2-transformed sample or control intensity ratios.	A
CGI cnv	Complete Genomics cnv	Copy Number Variation file from Complete Genomics containing cvg, ploidy and score information.	2.0
2-Channel ADF	Agilent TCGA Array Description File	TCGA's Agilent platform G4502A_07_01 ADF file containing the array probe information and corresponding genomic or gene annotation.	A
gVCF	genome variant call format	A VCF file following VCF 4.1 specifications combines information on variant calls (SNVs and small-indels) with genotype and read depth information for all non-variant positions in the reference.	20120906a
COSMIC Coding Mutations, tab-delimited	COSMIC Coding Mutations, tab-delimited	COSMIC export file CosmicCompleteExport_ vXX_<<date>>.tsv.gz	1.0
COSMIC Non-coding Mutations, comma-separated	COSMIC Non-coding Mutations, comma-separated	COSMIC export file CosmicNCV_vXX_ <<date>>.csv.gz	1.0

W_EHA_CHROMOSOME

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

Table A–3 W_EHA_CHROMOSOME

Table Name	Column Name	Description	Values Pre-seeded
W_EHA_ 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,MT

W_EHA_CHROM_MAPPING

This table is pre-seeded with multiple aliases for each of the chromosome record in W_EHA_CHROMOSOME. For example, **CHR1** can also be represented as **1**, similarly **chrM** will have alias like CHRMT, MT, and M.

W_EHA_SOMATIC_STATUS

This table is pre-seeded with all the somatic status codes currently present in the VCF file. The W_EHA_RSLT_SEQUENCING table has the foreign key to this table through SOMATIC_STATUS_WID.

Table A–4 W_EHA SOMATIC STATUS

SOMATIC STATUS_CODE	SOMATIC_STATUS
0	Wildtype
1	Germline
2	Somatic
3	LOH
4	Post-transcriptional modification
5	Unknown

W_EHA_PREDICTION_CODE

This table is pre-seeded with all the prediction codes for SIFT/PolyPhen annotation loader supports.

Table A–5 W_EHA_PREDICTION_CODE

CODE	CODE_TYPE
tolerated	SIFT
deleterious	SIFT
unknown	polyphen
benign	polyphen
possibly damaging	polyphen
probably damaging	polyphen

W_EHA_VERSION

This table is used to store a version label for all reference datatypes loaded in ODB. Version types are inserted during individual reference loads. The only data pre-seeded here is a version label, 'VERSION 3.9', with version type as 'GENETIC_CODE' used by the codon translation reference.

W_EHA_GENE_CODE

This table is used to store the name of the NCBI translation table for a set of genetic codon translations to amino acids. For each translation set the table stores the

descriptive code name, abbreviation of the name, FK to the curation source version, curation src version type, and the external ID for the translations.

Table A–6 W_EHA_GENE_CODE

GEN_CODE_NAME	GEN_CODE_ABBR	CURATION_SOURCE	EXTERNAL_ID
Standard	SGC0	GENETIC_CODE	1
Euplotid Nuclear	SGC9	GENETIC_CODE	10
Bacterial, Archaeal and Plant Plastid	Bacterial, Archaeal and Plant Plastid	GENETIC_CODE	11
Alternative Yeast Nuclear	Alternative Yeast Nuclear	GENETIC_CODE	12
Ascidian Mitochondrial	Ascidian Mitochondrial	GENETIC_CODE	13
Alternative Flatworm Mitochondrial	Alternative Flatworm Mitochondrial	GENETIC_CODE	14
Blepharisma Macronuclear	Blepharisma Macronuclear	GENETIC_CODE	15
Chlorophycean Mitochondrial	Chlorophycean Mitochondrial	GENETIC_CODE	16
Vertebrate Mitochondrial	SGC1	GENETIC_CODE	2
Trematode Mitochondrial	Trematode Mitochondrial	GENETIC_CODE	21
Scenedesmus obliquus Mitochondrial	Scenedesmus obliquus Mitochondrial	GENETIC_CODE	22
Thraustochytrium Mitochondrial	Thraustochytrium Mitochondrial	GENETIC_CODE	23
Pterobranchia Mitochondrial	Pterobranchia Mitochondrial	GENETIC_CODE	24
Yeast Mitochondrial	SGC2	GENETIC_CODE	3
Mold Mitochondrial; Protozoan Mitochondrial; Coelenterate Mitochondrial; Mycoplasma; Spiroplasma	SGC3	GENETIC_CODE	4
Invertebrate Mitochondrial	SGC4	GENETIC_CODE	5
Ciliate Nuclear; Dasycladacean Nuclear; Hexamita Nuclear	SGC5	GENETIC_CODE	6
Echinoderm Mitochondrial; Flatworm Mitochondrial	SGC8	GENETIC_CODE	9

W_EHA_GEN_CODE_TABLE

This table is pre-seeded with linkage of every triplet codon with the corresponding Amino Acid for all listed 18 gene code translations in the W_EHA_GEN_CODE table. Initiation codon indicators are also seeded for initiation codons.

Table A–7 W_EHA_GEN_CODE_TABLE (*Initiation codon is populated where given in a separate column)

GEN_CODE_WID	List of CODON value	AA1+(IS_INITIATION)* for each CODON value
1	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L(M),L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,*,Y,S,S,S,S,*,C,W,C,L,F,L(M),F
2	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,*,S,*,S,M(M),I(M),M(M),I(M),Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L,F
3	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,M(M),I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,T,T,T,T,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L,F
4	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I(M),I(M),M(M),I(M),Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L(M),L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L(M),F,L(M),F
5	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,S,S,S,S,M(M),I(M),M(M),I(M),Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L(M),F
6	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,Q,Y,Q,Y,S,S,S,S,*,C,W,C,L,F,L,F
9	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	N,N,K,N,T,T,T,T,S,S,S,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L,F
10	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,*,Y,S,S,S,S,C,C,W,C,L,F,L,F

Table A–7 (Cont.) W_EHA_GEN_CODE_TABLE (*Initiation codon is populated where given in a separate column)

GEN_CODE_WID	List of CODON value	AA1+(IS_INITIATION)* for each CODON value
11	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I(M),I(M),M(M),I(M),Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L(M),L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,*,C,W,C,L,F,L(M),F
12	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,S(M),L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,*,Y,S,S,S,S,*,C,W,C,L,F,L,F
13	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,G,S,G,S,M(M),I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L(M),F
14	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	N,N,K,N,T,T,T,T,S,S,S,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,Y,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L,F
15	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,Q,Y,S,S,S,S,*,C,W,C,L,F,L,F
16	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,L,Y,S,S,S,S,*,C,W,C,L,F,L,F
21	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	N,N,K,N,T,T,T,T,S,S,S,S,M,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L,F
22	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V,V,*,Y,L,Y,*,S,S,S,*,C,W,C,L,F,L,F

*Table A–7 (Cont.) W_EHA_GEN_CODE_TABLE (*Initiation codon is populated where given in a separate column)*

GEN_CODE_WID	List of CODON value	AA1+(IS_INITIATION)* for each CODON value
23	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,R,S,R,S,I,I,M(M),I(M),Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L,L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,*,C,W,C,*,F,L,F
24	AAA,AAC,AAG,AAT,ACA,ACC,ACG,ACT,AGA,AGC,AGG,AGT,ATA,ATC,ATG,ATT,CAA,CAC,CAG,CAT,CCA,CCC,CCG,CCT,CGA,CGC,CGG,CGT,CTA,CTC,CTG,CTT,GAA,GAC,GAG,GAT,GCA,GCC,GCG,GCT,GGA,GGC,GGG,GGT,GTA,GTC,GTG,GTT,TAA,TAC,TAG,TAT,TCA,TCC,TCG,TCT,TGA,TGC,TGG,TGT,TTA,TTC,TTG,TTT	K,N,K,N,T,T,T,T,S,S,K,S,I,I,M(M),I,Q,H,Q,H,P,P,P,P,R,R,R,R,L,L,L(M),L,E,D,E,D,A,A,A,A,G,G,G,G,V,V,V(M),V,*,Y,*,Y,S,S,S,S,W,C,W,C,L,F,L(M),F

W_EHA_GEN_CODE_USAGE

This table is used to map how GEN_CODE tables are used with each species and chromosome in ODB.

Table A–8 W_EHA_GEN_CODE

CHROMOSOME_WID	SPECIES_WID	GEN_CODE_WID
(null)	(null)	1
25	(null)	2

W_EHA_PRODUCT_VERSION

This table is used to record each installation or patch upgrade of the ODB schema. The current release_version seeded is '3.0.0.1'.

W_EHA_DATASOURCE

This table stores information about specimen source and is intended to be used primarily with CDM (four records in W_EHA_DATASOURCE). However, if needed, other databases with specimen information can be linked.

Table A–9 W_EHA_DATASOURCE

Table Name	Description	Value Pre-seeded
DATASOURCE_CD	Data source for Specimen	CDM, CDM_PATIENT, CDM_SUBJECT, CDM_BOTH
DATASOURCE_NM	Name of datasource for Specimen	CDM, CDM_PATIENT, CDM_SUBJECT, CDM_BOTH
DATASOURCE_DESC	Description of datasource for specimen	Cohort Data Model Patient Specimens (first 2 recs), Cohort Data Model Subject Specimens, Cohort Data Model Patient and Subject Specimens
SCHEMA_NAME	Name of schema	<<user input parameter at install time>> (for example, 'CDM')
DB_LINK_NAME	Link to database if needed	-
VALIDATION_PROC	Stored procedure used to validate specimens	ODB_UTIL.VALIDATE_CDM_PATIENT_SPEC, ODB_UTIL.VALIDATE_CDM_PATIENT_SPEC, ODB_UTIL.VALIDATE_CDM_SUBJECT_SPEC, ODB_UTIL.VALIDATE_CDM_BOTH_SPEC

A.2 Populated by User or Loader

Table A–10 W_EHA_VERSION

Reference Loader	Version_Type	Description of the File Version	Example VERSION_ LABEL values
EMBL Loader	DNA	Ensemble genome reference version specific to the genome build release.	GRCH37.P8 - for embl release 68 files.
Swiss-Prot loader	PROTEIN	Swissprot releases do not have labels assigned to them. However, they have release timestamps. The time stamp of a Uniprot file release is used.	01012012
Pathway loader	PATHWAY	Pathway release files do not have version labels. File release timestamps are used.	03032013
Prediction Loader	POLYPHEN	Polyphen file version and timestamp.	POLY_VER1_22042012
Prediction Loader	SIFT	SIFT file version and timestamp.	SIFT_VER1_22042012
Hugo Loader	HUGO	Hugo does not have a file archive. Use the data the file has been downloaded.	03032013
HGMD Loader	BIOBASE	The timestamp of the HGMD file release is used.	HGMD_04282013
Probe Loader	PROBE	This is a user-specific label. Use a label that describes the target microarray platform.	Affy_Hs_U133+_2.0_ GPL570
Cosmic Loader	COSMIC	Intended to store Cosmic release upgrade version number	COSMIC.V67
Genetic Code (seeded)	GENETIC_CODE	NCBI translation tables release version	VERSION 3.9

W_EHA_FILE

This table is populated with input file identity specific annotations by loaders while loading result data.

Table A–11 W_EHA_FILE

Column Name	Description	If Used Together With Regular Files
FILE_URI	Globally Unique identification tag used to differentiate files named similarly	User input; otherwise creates and updates with unique string: 'fille://trc/' + etl_proc_ wid + '<<input file name>>'
FILE_TYPE_WID	FK to W_EHA_ RSLT_FILE_ TYPE	Corresponds to WID in RSLT_ FILE_TYPE
FILE_NAME	Name of file	User input

W_EHA _FILE_LOAD

This table is populated with information on loading of files by loaders while loading result data.

Table A–12 W_EHA _FILE_LOAD

Column Name	Description	If Used Together With Regular Files	If Used Together With SecureFiles
FILE_WID	Foreign key to the FILE record	Corresponds to ROW_WID of W_EHA_FILE	-
FILE_LOAD_ SEQ_NUM	Sequence of each file that is loaded consecutive times	Autonumber	Autonumber
FILE_PATH	Path to input file	For example, C:/inputfile.txt	-
SPECIMEN_ VENDOR_ NUMBER	Vendor identifier in linked specimen datasource database	User input parameter	-
VERSION_WID	Foreign key to Version	Corresponds to ROW_WID of W_EHA_VERSION	-
FILE_STORAGE_ FLG	E for External, S for SecureFiles	E	S
FILE_SIZE	File size in bytes	-	-
LOADER_NAME	Name of loader package that generates the load	For example, 'ODB_RSLT_ CNV_UTIL' for CNV loader	-
DIRECTORY_ NAME	Oracle Directory used for copies of file	For example, ODB_RES_DIR	-
DBFS_STORE	DBFS store name if file is copied to DBFS	-	USER input name of file system

W_EHA _FILE_LOAD_QLFR

This table is populated with metadata information for each file load.

Table A–13 W_EHA _FILE_LOAD_QLFR

Column Name	Description	If Used Together With Regular Files	If Used Together With SecureFiles
FILE_LOAD_WID	Foreign key to the FILE_ LOAD record	Corresponds to ROW_WID of W_EHA_FILE_LOAD	-
QUALIFIER_WID	Foreign key to qualifier table	Corresponds to ROW_WID of W_EHA_QUALIFIER	-
QLFR_CHAR_ VALUE	Qualifier character attribute value	User input qualifier or file qualifier character values	-

W_EHA_RSLT_FILE_SPEC

This table stores foreign keys to W_EHA_FILE and W_EHA_RSLT_ SPECIMEN and links a specific file which is loaded to ODB via the loaders to the specimen that has results in the file.

W_EHA_RSLT_STUDY

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

Additionally, you can merge Study records stored in OHCE's W_EHA_STUDY_D table by running a function call from a package in OHCE; COHORT_PROTOCOL_ UTIL.Process_Protocols().

Table A–14 W_EHA_RSLT_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 Tables or Columns Not Populated Through Loader Scripts

The following table indicates result and reference tables or columns that are currently not being populated.

Table A–15 Unpopulated Tables

Table Name	Column Name	Description
W_EHA_RSLT_CNV_X	-	Copy Number Variation additional data table.
W_EHA_PROBE_ALT_LINK	-	Table to store alternative links between probes and genes, for example, if a single probe is linked to multiple genes.
W_EHA_ADF_COMPOSITE_ XREF	-	Additional reference table
W_EHA_ADF_REPORTER_ XREF	-	Additional reference table
W_EHA_ANATOMICAL_SITE	-	No loader for this table
W_EHA_HISTOLOGY	-	No loader for this table
W_EHA_DISEASE_G_VAR_ QLFR	-	Additional qualifier table
W_EHA_GENE_XREF	-	Additional reference table
W_EHA_QLFR_CATEGORY	-	One of QC metadata tables
W_EHA_QUALIFIER	-	One of QC metadata tables
W_EHA_QLFR_TABLE	-	One of QC metadata tables
W_EHA_QLFR_TRANSLATION	-	One of QC metadata tables
W_EHA_RSLT_DXP_ANLYS	-	No loader for this table
W_EHA_RSLT_DXP_ANLYS_ MD	-	No loader for this table
W_EHA_RSLT_DXP_GRP	-	No loader for this table
W_EHA_RSLT_DXP_GRP_SPEC	-	No loader for this table
W_EHA_RSLT_FILE_SPEC_ QLFR	-	Additional qualifier table
W_EHA_RSLT_SPEC_QLFR	-	Additional qualifier table
W_EHA_FILE_QLFR	-	Additional qualifier table
W_EHA_REF_SAMPLE_XREF	-	Additional reference table
W_EHA_SPEC_EXTERNAL	-	Additional table to link to non-OHSCE specimens
W_EHA_VARIANT_FILE	-	Used to link all reference variants to the file used to load that variant
W_EHA_SOMATIC_VAR_INFO	-	No loader for this table

Table A–15 (Cont.) Unpopulated Tables

Table Name	Column Name	Description
W_EHA_SOMATIC_VAR_QLFR	-	Additional qualifier table
W_EHA_SOMATIC_VAR_XREF	-	Additional reference table
W_EHA_SOURCE_LIT_REF	-	No loader for this table
W_EHA_VARIANT_QLFR	-	Additional variant qualifier table

Index

S

U

V