Oracle® Health Sciences Omics Data Bank

Programmer's Guide

Release 2.5

E35680-04

June 2013

ORACLE®

Oracle Health Sciences Omics Data Bank Programmer's Guide Release 2.5

E35680-04

Contents

4 Loaders for Result Data

7 Miscellaneous Topics

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 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 Secure Installation and Configuration Guide*

- *Oracle® Health Sciences Translational Research Center Implementation Scripts 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 is a reference set to provide the genomic features 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 in the model 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. The patient 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 and does not include any front-end user interfaces but only command line APIs. A set of scripts that can be used to load reference genomic data from specific sources and several types of result data from a limited set of formats are included with the model.

The model is intended to handle very large amounts of data (in the order of terabytes and more). To gauge the scale of the data, one should consider that the human genome is around 3 billion bases in each strand and the number of genes that produce proteins is around 23,000. Each gene has many different variants and attributes that are described in detail. This holds true for each protein, gene component, pathway etc, and 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 2.5 can handle multiple versions of reference data in the same instance, for example GRCh36, GRCh37. You can load multiple versions of each reference data and link the results to a specific reference version based on the requirement. For example user with sequencing data mapped to GRCh36 will link the results to GRCh36 reference data loaded from ENSEMBL. The multiple reference version support is extended to ENSEMBL, SwissProt, HUGO, Pathway, SIFT/PolyPhen, ADF and probe loader.

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 database. The model loads this cross reference information to, including known variation data, let queries to 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 a separate 'DNA' reference version. Current and alternate releases are available at their FTP publication repository.
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 current release of the SwissProt file is given on their download page, older SwissPort releases are stored 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 for identifying human gene locations annotation only. The HUGO gene names are needed 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 a archive of older versions or releases. Any dataset taken will be 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/genes participating in each pathway. The coverage of pathway as a reference is minimal. Pathway provides a list of current and previous releases of dataset to retrieve 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 about 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.

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

Most of the online databases let you download complete references, or specific references for sections of the genome. Since some customers may only need some genes or 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 to be expanded as required by the customer. The HUGO, Pathwaycommons, and HGMD databases are do not keep older versions but rather provide the latest versions only 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. On the other hand, 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 of the above result types in a single SQL statement across multiple versions of references or within 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 to reference. However, there is one exception, the W_EHA_VARIANT 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)

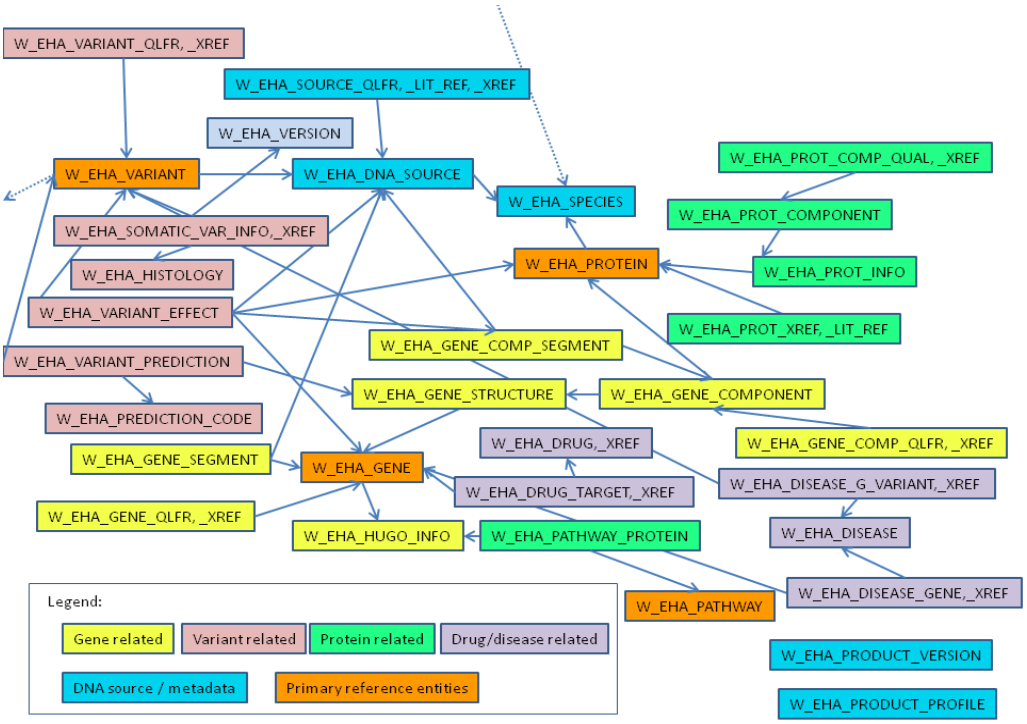
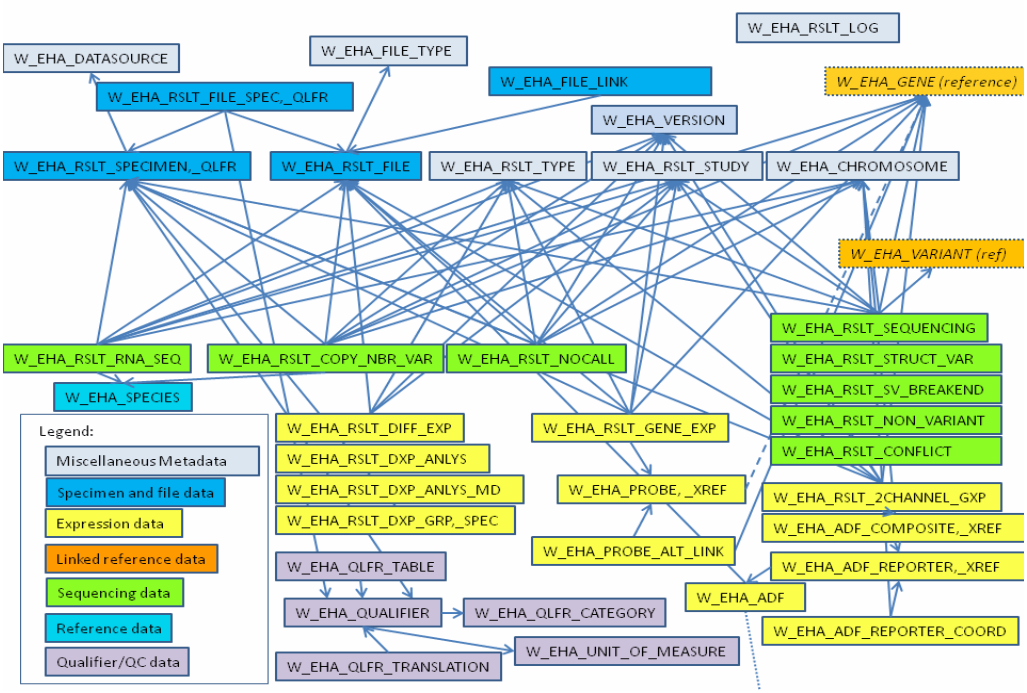


Figure 1-2 shows how the result tables link to create the ODB result tables section. Only table names are shown in the figure.

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



1.3 Reference Data Tables

The reference data starts with the SPECIES and the DNA_SOURCE tables.

W_EHA_SPECIES

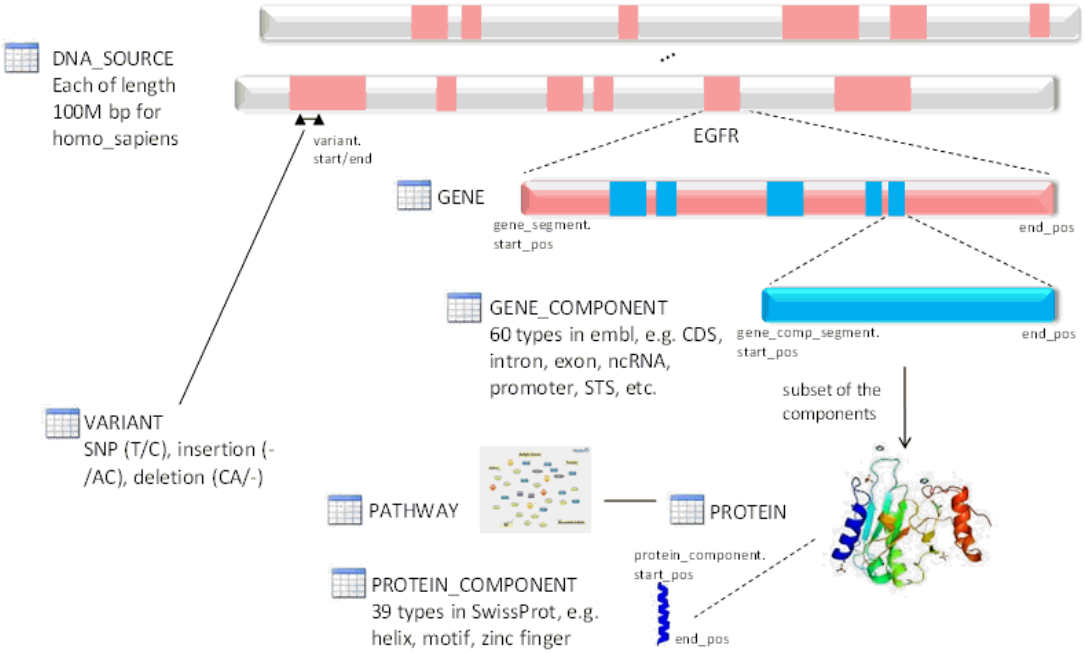
This species table stores information about each genome in the database. The current model permits any number of species genomes to be loaded. You must specify species in queries if there are similar genes between the organisms being tested.

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

W_EHA_DNA_SOURCE

The DNA_SOURCE table stores multiple records for different reference DNA strands 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 strands are then used to map detected variations for each organism tested. The W_EHA_DNA_SOURCE table has the foreign key top SPECIES and also has a CLOB field to store the reference strand information. The DNA has a specific character notation to keep track of each DNA base. There are additional characters used for sections that have not been sequenced (*N*) and there are other characters used to represent other possible DNA bases. The records in this table are used as the parent records to map genes and gene components. If Ensembl releases patches that show how some genes are re-defined, new DNA_SOURCE records are created and then linked to the other records as needed. 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, DNA_SOURCE table links each reference DNA record with its DNA version which is stored in W_EHA_VERSION table under version type 'DNA'.

Figure 1–3 Genome Division by the Data Model



W_EHA_GENE

Each chromosome of the DNA strand has many different genes, each of which has a start and end position. The entire size of the gene does not create the protein directly, but there are recognized sections of the DNA that scientists agree should be considered as part of the gene. The W_EHA_GENE table has fields for how the Ensembl database refers to the gene, 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 since the patch DNA sequences loaded do not list chromosomes.

W_EHA_GENE_SEGMENT

This table is required to map the different segments of a gene to each buffer. Some genes are sequenced in multiple buffers and require this joining table to track each segment. This table gives the location in the buffer 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 a coding strand of a gene is on the reverse strand (COMPLEMENT=1) or forward strand (COMPLEMENT=0) of a chromosome.

_XREF, _QUALIFIER

The XREF table associated with many of the different tables is used to list all of the cross reference information stored in the Ensembl database. There is a finite list of databases used, and each database has a specific format for the reference ID. You can use these reference ID values for queries. The _QUALIFIER table associated with different tables is 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 in case it is needed for reference.

W_EHA_GENE_STRUCTURE

The process of gene transcription is accomplished by many different interim molecules that originate from sections of the gene. For each protein created, there is a distinct set of sections which are used. Each of these groups is identified in the EMBL file having the same TRANSCRIPT_ID qualifier. A GENE_STRUCTURE record is created to link to the protein and to be used as a parent record for all of the gene components. A given gene can have multiple proteins that are created (sometimes using the same sections) and each 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 they all have 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). User can run various queries searching for mutations that occur in any of these regions, including the entire gene region.

W_EHA_GENE_COMP_SEGMENT

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

W_EHA_PROTEIN

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

W_EHA_PROT_COMPONENT

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

W_EHA_VARIANT

The VARIANT table is used to record the known reference sequence, REFERENCE_ SEQ, corresponding to one or more variants that differ from the reference DNA_ SOURCE. Most of these variants are well documented and compiled from other research. When results are uploaded, sometimes novel variants are detected and there are no known references for this variant.

These results 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.

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. This is important because of the way the EMBL files store patch sequences. The patch sequences (which are corrections to the human genome project) list the chromosome using the accession number of the DNA used for detection. The W_EHA_HUGO_INFO table is required to look up the HUGO gene names to find various cross references and the correct chromosome number for each gene. Each gene in the HUGO_INFO table is linked to 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 PATHWAY table is linked to W_EHA_VERSION table with specific version of Pathwaycommons 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 each gene or protein with one or more pathways.

W_EHA_VARIANT_PREDICTION

This table stores information relevant to SIFT (Sorting Intolerant From Tolerant) or Polyphen (*Poly*morphism *Pheno*typing) 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.

Polymorphism Phenotyping (**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. SIFT 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*, and *tolerated*, and so on.

W_EHA_VARIANT_EFFECT

This table stores variant impact or effect as computed by 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 an amino acid changes results in place of the reference data.
- 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 2.5 and will be provided in future releases of ODB.

W_EHA_PRODUCT_PROFILE

This table stores default global Promoter and Flanking offsets which are the inputs you provide during installation. It also stores flags for 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 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 default set to 'Y'); Debug, TRACE, and DBMS output which are set to 'N').

W_EHA_PRODUCT_VERSION

This table lists the current version of Omics Data Model and is used primarily by Cohort Explorer application interface (not included in this release). For example, currently it is set to 2.5.

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.

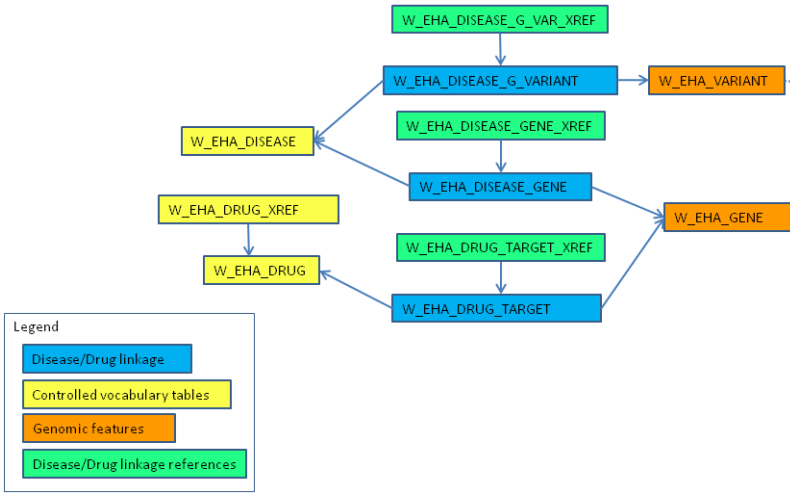
W_EHA_DRUG

This table stores DrugBank-derived drug names.

W_EHA_DRUG_TARGET

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 contains 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 a '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. A 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 to 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 needs to be manually populated. W_EHA_PROBE must be populated by the expression loader prior to loading any results corresponding to gene expression. In addition, any reference information pertaining to probes can be recorded in the W_EHA_PROBE_XREF table.

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:

1. W_EHA_RSLT_SEQUENCING
2. W_EHA_RSLT_GENE_EXP
3. W_EHA_RSLT_NOCALL
4. W_EHA_RSLT_RNA_SEQ
5. W_EHA_RSLT_2CHANNEL_GXP
6. W_EHA_RSLT_COPY_NBR_VAR
7. W_EHA_RSLT_NON_VARIANT
8. W_EHA_RSLT_CONFLICT
9. W_EHA_RSLT_STRUCT_VAR
10. W_EHA_RSLT_SV_BREAKEND

All the major result tables listed above contain foreign keys to W_EHA_RSLT_FILE, W_EHA_RSLT_STUDY, W_EHA_RSLT_SPECIMEN, W_EHA_RSLT_TYPE, W_EHA_GENE and W_EHA_VERSION tables.

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. Additiionally, 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'.

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 insertions, deletions, or substitutions are recorded in this table along with any quality metrics on this information.

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

W_EHA_RSLT_SEQUENCING_X

This is an additional table to store the allele value greater than 500 bases. It is always used as a helper table along with the main RSLT_SEQUENCING table.

W_EHA_RSLT_NOCALL

This table contains results coming from 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 the VCF and gVCF file. The VCF loader will populate 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 variant which conflict with an existing high quality score variant which is reported in W_EHA_RSLT_SEQUENCING table for the same specimen. This table is populated through VCF loader with data coming from the gVCF file.

W_EHA_RSLT_STRUCT_VAR

This table contains the large structural variants coming from the VCF file. This table has a foreign key to W_EHA_VARIANT table and stores information similar to W_EHA_RSLT_SEQUENCING table with some additional information specific to large SV.

W_EHA_RSLT_SV_BREAKEND

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

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 being populated via the newly provided CNV loader, which can load .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 as to be able 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 a lookup of the gene composite name in w_eha_adf_composite table.
- GENE_WID - by a lookup of gene segment table of all complete and partial overlaps of gene segments in reference for 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 the results have been collected for. This partitioning scheme leads to significant performance improvements.

W_EHA_CHROMOSOME

This table holds all the chromosomes names and is pre-seeded with names for all Human chromosomes. Refer to 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 four result data tables. Every record in any of the result tables must have a foreign key that links to a particular specimen in this table. 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.

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_RSLT_FILE

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

W_EHA_FILE_TYPE

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

W_EHA_RSLT_FILE_QLFR

This table contains the header information from the VCF and gVCF file. Any line starting with '##' in the VCF and gVCF file will be stored in this table. This is also where alternative location of file 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_RSLT_FILE and W_EHA_RSLT_SPECIMEN and basically link a specific file which is loaded to ODB through the loaders to the specimen it belongs to.

1.4.1 Result Tables for Qualifier Metadata

Newly enhanced ODB data model has new tables for Qualifier Metadata attributes, however, there is no loader provided to populate these tables. 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 could 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.

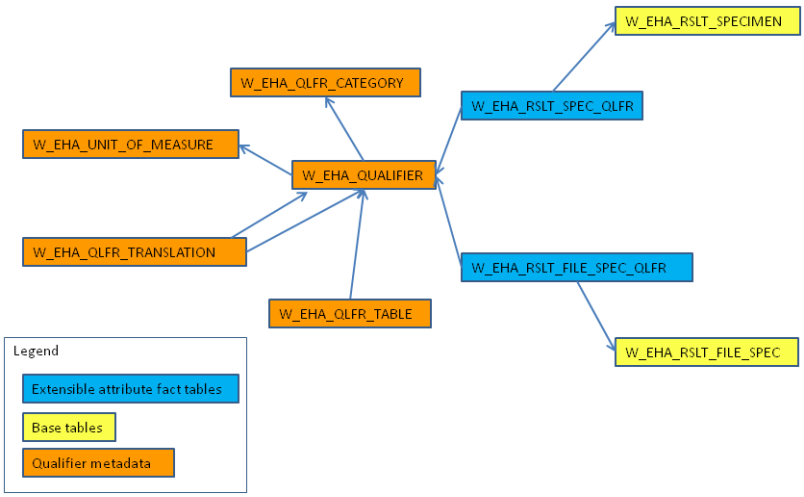
W_EHA_RSLT_FILE_SPEC_QLFR

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. But 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.

1.4.3 Table for Logging

All loaders and procedures created in ODB output 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.

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 an external table to access data in the files which require an Oracle directory object to be created. Oracle directory objects require that the database Operating System user account has 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.

Oracle recommends that the Oracle Directory used for loading files into the database not exist in DBFS secure files. When DBFS is used for the location of data files to be loaded, the database has a contention for resources trying to load the file. When the external table is used to load the file, each read of the data file requires the database to first send the buffer to the Operating System where the DBFS is mounted. Then the Operating System sends this buffer to the database to match the read request of the data file. This slows down the loading process and causes the database to have many wait events while reading the data file. Therefore, Oracle recommends not using the Oracle Directory objects located in DBFS.

All the loaders described here now use external tables. The ODB schema user must have the CREATE ANY DIRECTORY privilege and the CREATE ANY TABLE privilege. The name used for the directory object is used as a parameter to all of the loaders. The Oracle database OS account must have permissions to access the directory specified in the Oracle directory object.

An example of a command to create the 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 convention than Linux servers. Also, the directory used must be mounted on the host OS of the database server before the database server is started.

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

GRANT READ on DIRECTORY <<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 name of the database user that executes the loaders. A database user should have both READ and WRITE grants on Oracle directory in order to process loaders.

2.2 Setting up Oracle Wallet

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

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

```
DB001_Wallet =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 10.178.187.186)(PORT = 1521))
)
(CONNECT_DATA =
(SERVICE_NAME = db001)
)
)
```

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

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

```
>cd d:
>d:
>mkdir wallets
>cd wallets
>mkstore -wrl D:\wallets -create -nologo
Enter password: <type a 8 alphanumeric-character password>
```

```
Enter password again: <retype above password>

>dir

Volume in drive D is Data

Volume Serial Number is C###

Directory of D:\wallets

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

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

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

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

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

3. Add your database credentials to your wallet.

```
>mkstore -wrl D:\wallets -createCredential DB001_Wallet 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 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. On any command prompt terminal enter the following:

```
>sqlplus /@DB001_Wallet
```

You will get the following result:

```
SQL*Plus: Release 11.2.0.1.0 Production on Fri 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. You must create a database user having the following privileges:
   - CREATE SYNONYM
   - CREATE SESSION
2. You must 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 users of the UIs whose credentials are to be passed to database layer for querying only.
   - OmicsDatamartAdmin

     Loads into 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

     Able to load result data into the Omics Data Bank via provided result loaders. This role enables a named user to write to ODB result side of the model.
3. Local synonym should be created for the database user. The create_odb_synonyms.sql is available in the "odb_install" folder in the ODB Software package.
   a. Connect to the database instance with the database user.
   b. Execute the following command by replace the odb_schema_name with the actual name of your Omics Data Bank schema:

      ```
      @create_odb_synonyms.sql &&odb_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 with SPECIMEN_NUMBER and SPECIMEN_VENDOR_NUMBER information, and SPECIMEN_DATASOURCE. These entities should match the record in specimen datasource schema. OHSCE Data Model is the default datasource for specimen in the current release. The SPECIMEN_WID in W_EHA_RSLT_SPECIMEN table in ODB should match the ROW_WID in the W_EHA_SPECIMEN PATIENT_H table in OHSCE Data Model.

Every result record imported into the ODB schema is linked to a SPECIMEN record. The SPECIMEN data can come from any external schema either on the same instance (co-located) or on an external instance. The loaders which load various files call a specific stored procedure to validate that the SPECIMEN exists.

The assumption is that any database used to provide the SPECIMEN information has a single numeric field to identify the correct SPECIMEN. This numeric value is stored in each result record (without direct FK definition). The stored procedure used is in a package ODB_UTIL which is not wrapped to let additional external databases to be supported. Currently, this stored procedure is implemented to support validating SPECIMEN records stored in the Cohort Data Model. You need to expand this stored procedure to support other schemas that are intended to provide specimen data.

Following is the structure of ODB_UTIL.GET_SPECIMEN_WID stored procedure:

function get_specimen_wid

(

i_datasource_id -

number,

i_specimen_number

in varchar2,

i_specimen_vendor

varchar2

,i_study_id number,i_etl_proc_id number, i_enterprise_id number, i_file_wid in number, i_logger result_logger default null, i_call_count)) 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 which is a ROW_WID of W_EHA_RSLT_FILE table. This is the same ROW_WID which loader creates the records for result file used for processing the loader. This is used by 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 name of CDM and then code for looking for specimen data in CDM. If additional database schemas need to be used to provide specimen information, you must first add a record first to W_EHA_DATASOURCE with a unique name.

The stored procedure has to specifically handle that 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 add_file_spec() local procedure which insert record into w_eha_rslt_file_spec table. This established the link between result file and specimen used for loading result data.

If the get_specimen_wid stored procedure raises the NO_DATA_FOUND exception (if specimen does not exists in w_eha_rslt_specimen 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 the external data source, user needs to mention the DB link (w_eha_datasource). If the procedure does not exists on 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 and in w_eha_rslt_file_spec table, and returns specimen_wid to the loader

This procedure also logs relevant warning, error, and such 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 Reference Version Compatibility

User must ensure the compatibility of the ENSEMBL version loaded into ODB with other reference and results data before loading them to ODB.

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

1. GVF data file should belong to the same version of ENSEMBL that exists in ODB. For example, if ODB is loaded with ENSEMBL 66 version, then the GVF file to be loaded should also belong to ENSEMBL 66.
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. 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.
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. You can find mapping reference version details in this file.

2.6 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.

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 has 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, for which the name and password are known.

   Note: Ensembl and SwissProt loaders are written in Java.

2. Java Runtime 1.7 or higher must be installed and should be the default on the machine (can be verified using the java -version command from the command prompt).
3. Copy the Reference Loader folder into a directory of your choice. The program should be run from the directory it is installed in.

3.1.2 Files to Load

Following is a list of files to be loaded:

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

   ftp://ftp.ensembl.org/pub/release-71/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.

   The files are organized by chromosome. There are also some configuration and patch files. At least all chromosome files must be loaded to cover the entire Human genome. The files are gzipped, and can be loaded without un-gzipping.

   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 2.5, allowing 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 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 here:

   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 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 and keeps growing. So, 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, connection failure is not logged. Also, a SUMMARY log record is not created by the EMBL/SwissProt loader 2.5.

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'.

   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 allowing the

user to use the provided version label, and if the user confirms, the version is created. If no version argument is provided, the loaders enter full interactive mode, allowing the user to select an existing version from a list or create a new one. The same is true for the SwissProt loader, except here the version is of type 'PROTEIN'.

The version labels are not case-sensitive and are stored in uppercase.

Note: If the version label is not provided or the provided version does not exist, the EMBL or SwissProt loader prompts the user for input and do not continue until the user responds. Therefore, if you want to run the loader in the non-interactive mode, it is necessary to create the version of the correct type beforehand and to 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 native JAVA command-line interface available described in Section 3.1.4):

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 run you can optionally pass the Species List file. The purpose of the Species List file is to let only loading protein information for the organism you want.

   The format of the file is simple: type in the species primary (Latin) name, one species per line. A file for just the human genome is now included in the distribution — it is named Species.dat and contained in the main Loader Directory. If there is no -protFile option with the name of a species list file, ALL proteins for ALL species 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, otherwise it prompts for a password. If an Oracle Wallet is set up, the user has to 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. Path for Oracle Home
   e. Path to Wallet
   f. Path and name of species list. This is an optional parameter. If you do not have this list enter "
   g. Complete path and name of the data file
   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 Oracle Wallet is set up and Species list is not present:

```
C:\>swissProt.bat "" DB001Wallet trc_gdm
C:\ora11g\product\11.2.0\dbhome_2\NETWORK\ADMIN D:\wallets ""
SwissProt.dat "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 Oracle Wallet is not set up, you have to pass the following parameters when swissProt.bat is run:

i. Username to connect to schema

j. Url — Full DB URL (host:port:instance, or scan-server-name:port:SID for a multiple node DB). For example, Localhost:1613:devdb1

k. Schema name

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

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

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

o. Complete path and name of the data file

p. 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 Oracle Wallet is not set up and Species list is present:

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

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 file 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, if the user wants 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 multiple files can be loaded concurrently.

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

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

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

   c. Schema name

   d. Path for Oracle Home

   e. Path to Wallet

   f. Complete path and name of the data file

   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, and the seventh argument should then be provided as 2048 (that is 2048 MB). Then these files can 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. Then you need to use another machine.

   h. The version of type 'DNA' can be passed as the 8th 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 of 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 Oracle Wallet is not set, you have to pass the following parameters when Embl.bat is run:

a. Username to connect to schema

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

c. Schema name

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

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

f. Complete path and name of the data file

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 of 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 following arguments. Running the application using these arguments provides some additional capabilities not supported by the shell/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 user should specify the full path to the Java executable with version 1.7.

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 2.5 version of the EMBL/SwissProt Loader it is usually not necessary, because the memory usage has been optimized to handle chromosome-wide Ensembl files. However, if you ever encounter an *out-of-memory* error for any Enseml EMBL file, use it (provided that the machine has enough memory and the operating system is Linux or 64-bit Windows).

The arguments (except for the 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/option arguments begin with the "-" character. Some of them require a value as the following argument, others are stand-alone. To get the up-to-date list of all available arguments, run the following command:

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

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

`-url <url>` - specifies the URL of the database to comment to. <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 to log into the database with (needed only when not using an Oracle Wallet).

`-wallet <wallet>` - specifies the directory where the Oracle Wallet is used

`-orahome <oracle home directory>` - specifies the oracle home directory, when a Wallet is used

`-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 the user 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 the user wants 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 the user chooses to create a new version). Omitting this option is useful for verifying that the file parses without errors.

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 Gathering Optimizer Statistics

Oracle recommends gathering table and index statistics after completing Ensembl data load. 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 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 the complete HGNC dataset which can be retrieved from their Statistics and Downloads Webpage here: http://www.genenames.org/cgi-bin/hgnc_stats. There are multiple datasets at this web page. The user has to specifically download 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 in tab-separated values given with column headers.

Important: The format of the complete HGNC files has changed as of May 2013. The ODB HUGO Loader 2.5 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. The content below shows the step-by step process of this load procedure.

3.2.2 Running the Loader

The loader is found bundled in the latest ODB build in the compressed file Hugo_Loader.zip. This folder consists of 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, make sure the scripts are executable (you may need to run 'chmod u+x *.sh')

4. The hugo_loader.sh/.bat scripts can use the credentials stored in an Oracle Wallet to connect to the schema that has the ODB. The following parameters should be passed when the hugo_loader.bat is run:
   a. The 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 "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]

`-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, see "Setting up a Directory Object"

```
-reference_version <VARCHAR2>
```

The version of the Hugo file version 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 interactively ask the user if he wishes 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 -data_file "genefam_list.pl"
-data_directory "ODB_LOAD" -reference_version "DLD_DT_01062013" -read_size
null
```

Windows

```
C:\> hugo_loader.bat -db_wallet odb_user -data_file "genefam_list.pl"
-data_directory "ODB_LOAD" -reference_version "DLD_DT_01062013" -read_size
null
```

Once the loading is complete, log into SQL developer, or SQP*Plus with ODB Schema and verify that 35000 or more records are populated in W_EHA_HUGO_INFO table. The information about the execution is logged in the W_EHA_RSLT_LOG table. If run with the **-print_summary 1** option, the loader will also print on the console information about the execution, including the count of inserted records, and the errors, if there are 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 of any given species for which DNA source records are present. The loader 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). Hence ensure the EMBL loader is run first with the relevant species' EMBL input files.

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

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

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

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

3.3.2 Running the Loader

For the GVF loaders, the.bat and.sh files require the same set of named command line arguments, except that the BAT script only supports using a Oracle Wallet connection (like all other 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 "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 "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 null -preprocess_file null -read_size null
```

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 null -preprocess_file null -read_size null
```

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 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/pc-snapshot/current-release/gsea/by_species/homo-sapiens-9606-gene-symbol.gmt.zip.

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

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

3.4.2 Running the Loader

The loader is made up of three files:

- pathway_loader.bat
- pathway_loader.sh
- pathway_script.sql

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.

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

There are two bulk insert statements written 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.

Now 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 Oracle Wallet while the pathway_loader.sh script can be run with or without Oracle Wallet being set up.

The load_pathway procedure has been altered to include an 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 "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 "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 ask the user if he wishes 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 null
```

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 null
```

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 version of SIFT and PolyPhen data. The versions are recorded in the W_EHA_VERSION and W_EHA_FILE_TYPE tables.

Steps to Download Data from ENSEMBL BioMart

The source data is downloaded from the ENSEMBL BioMart tool. You have to download SIFT and POLYPHEN data separately and load them separately.

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

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

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

- From Dataset: Select following options
  - Select Ensembl Variation <ver>.
  - Select either Homo sapiens Somatic Variation (COSMIC <ver>) or Homo sapiens Variation (dbSNP <ver>;ENSEMBL).
- 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.
- 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

    * Ensemble Transcript ID
    * Polyphen prediction or SIFT prediction
    * Polyphen score or SIFT score.

- Then click **Result** at the top of the screen.
- FromExport all results to select following
  - Select File
  - Select TSV
  - Select Unique results only.
- Click **Go** to download the file.

User should ensure that he selects the attributes in the specified order. Also, user should ensure ensure that he selects PolyPhenprediction and PolyPhen score when downloading PolyPhen data, and SIFT prediction and SIFT score when downloading SIFT data.

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 parameter.

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. The ORACLE_LOADER driver access data stored in any format that can be loaded by the SQL*Loader. No DML can be performed on external tables but they can be used for query, join, and sort operations.

The 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 result 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 written dynamically. One inserts record into the w_eha_variant_prediction table and other inserts record into the w_eha_rslt_ log table.

A select statement which 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 the duplicate records for different variation name which has the same variant record (VARIANT_WID). The dataset of this inline query will then be the outer join with the W_EHA_VARIANT_ PREDICTION table to lookup for 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 (file name same as previously loaded) with 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

Record is inserted in the W_EHA_VARIANT_PREDICTION table for a variant belonging to a specific transcript; hence lookup of reference_id and transcript_id is performed 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.

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 Oracle Wallet to be set up to run correctly. However, the shell script can be run with or without Oracle Wallet being set up.

SIFT/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://useast.ensembl.org/info/docs/variation/vep/vep_script.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. This 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

Table 3–1 (Cont.) Mapping of Polyphen or SIFT File

Column Name in Result File	Table and Column Name in ODB	Description
Ensembl Transcript ID	W_EHA_GENE_STRUCTURE. TRANSCRIPT_ID	This is a FK to W_EHA_ GENE_STRUCTURE table. This 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 "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 "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 interactively ask the user if he wishes 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 null -preprocess_file null
-read_size null
```

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 null -preprocess_file null
-read_size null
```

Typical Errors Associated with prediction_score Loader

"Record not inserted" is logged if same version, same file type loaded but input record being inserted has score different than existing record.

Other possible errors are similar to other loaders: '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 a reference loader rather than result 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 separated.
- 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 one of the script files (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.

This stored procedure creates an external table dynamically and uploads data from the source file into it. External tables let Oracle to query data that is stored outside the database in flat files. The ORACLE_LOADER driver access 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. During updation of a record, it updates all the columns including the row_wid with new row_wid for a particular probe name.

The old records can be referenced using the Flashback Data Archive (FDA) approach.

The FDA approach 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.

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

For Shell scripts, if Oracle Wallet is set up, the shell script uses those credentials to run the Sqlplus. If 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 "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 "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 null
```

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 null
```

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, the file lists out the Features (spots) found on an array, along with its location on the array and its 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 in the above wedpage:
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 one of the script files (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/http location) ,Read Size as an input parameter.

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

The stored procedure dynamically creates ADF_DATA_!!SEQ!! as an external table. This external table stores the complete result data. This table will map 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 which do lookup against user label, file wid, version wid, etl proc wid and enterprise wid dataset. All the values are passed as an input parameters. If the dataset matches with any of the w_eha_adf record then the loader will just update all the columns (except w_update_dt) of w_eha_adf table for corresponding user_label of w_eha_adf table.

The three multi-table insert statements written dynamically inserts record 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.

The FDA approach 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 Oracle Wallet to be set up to run correctly. The shell script can be run with or without Oracle Wallet being set up.

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 "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 "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 null -preprocess_file null -data_file_path null -dbfs_
store null -alt_file_loc null -read_size null
```

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 null -preprocess_file null -data_file_path null -dbfs_
store null -alt_file_loc null -read_size null
```

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 BIOBASE GFF formatted source files to ODB reference tables. The loader currently loads 3 data source file types: 'hgmd_hg*.gff', 'hgmd_disease_hg*.gff', and 'drug_hg*.gff'.

Please note that 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 and Human Genome 19 in UCSC notation (or Build 36 and Build 37 in NCBI notation).

One can load both HG18 and HG19 curated data, but they have to be linked to 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. The first step is to identify all variants listed in the source file and add the novel ones 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. The next step is to add new disease names 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/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 a referenced by the DruBank ids (www.drugbank.ca). Two other tables store gene/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 could be invoked from a provided shell or batch file, The procedure accepts 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 the source reference files are prepared with. The loader parameter called "reference genome version" specifies which reference genome release loaded into ODB schema should the 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 Oracle Wallet to be set up to run correctly. The shell script can be run with or without Oracle Wallet being set up.

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 "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 "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 null
```

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 null
```

4

Loaders for Result Data

This chapter includes the following topics:

4.1 Prerequisites

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

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

In addition, user should 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 W_EHA_RSLT_STUDY table. There is a sequence (W_EHA_RSLT_STUDY_S) associated with this table to let as many study records required for testing. The result data is now partitioned using the FK to study. The required number of studies should be added to this table. The result loaders use the value for RESULT_STUDY_NAME to lookup the corresponding study primary key. For simple testing you only need one record.
2. Configure a W_EHA_DATASOURCE record to identify the CDM schema to validate specimen numbers. Each result record that is to be loaded must have the

specimen exist in the CDM schema in the W_EHA_SPECIMEN_PATIENT_H table. Note that the CDM schema needs the v1.01 patch installed, which adds a SPECIMEN_VENDOR_NUMBER field to be used for vendor specific information. The W_EHA_DATASOURCE can use a database link if the CDM schema is in another instance.

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

To install the loaders, copy the Result_Loader folder into a directory. You must run a loader from the directory it is installed in (on Linux this requires an execute permission for all SH scripts).

4.1.1 Setting Default Cache Sizes for Result Loading

Each of the result tables have a corresponding sequence which is named similar to the table with a suffix of "_S". Each of these result tables sequences have 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 it is recommended to have the DBA adjust all of the corresponding sequence cache sizes to at least 100,000 or larger.

Lower sequence cache sizes can result in waits 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. 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
2. MAF Loader
   - "ODB_RSLT_MAF_UTIL (default cache 15000)
     - W_EHA_RSLT_SEQUENCING1_S
3. RNA-seq loader
   - ODB_RSLT_RNA_SEQ_UTIL (default cache 6000)

  - W_EHA_RSLT_RNA_SEQ_S

4. CNV loader
   - ODB_RSLT_CNV_UTIL (default cache 6000)
     - W_EHA_RSLT_COPY_NBR_VAR_S
5. Single channel loader
   - ODB_RSLT_SINGLE_CHANNEL_UTIL (default cache 6000)
     - W_EHA_RSLT_GENE_EXP_S
6. Dual channel loader
   - ODB_RSLT_DUAL_CHANNEL_UTIL (default cache 6000)
     - W_EHA_RSLT_2CHANNEL_GXP_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
- 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.

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
- CNV_loader.bat
- TCGA_RNA_SEQ_loader.bat

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

If Oracle Wallet is set up, shell script uses those 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
- TCGA_RNA_SEQ_loader.sh
- CNV_loader.sh

4.3 Version Info Utility

The Version info utility is used to check 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 via -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 column VERSION_LABEL values where column VERSION_TYPE equals 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 would display 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 "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', and 'BIOBASE'.

4.4 VCF Sequence Data Loader

4.4.1 Functional Description

The VCF loader takes the chromosome, position and reference version details of a record from VCF file and checks if the corresponding region of that chromosome exists in W_EHA_DNA_SOURCE table for a specific reference version given as input to the loader. If it is present, it 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 does not log in to W_EHA_RSLT_LOG table.

The loader does not validate for invalid chromosome number or positions details. If it encounters such invalid data the loaders ignores that record and does not log it to 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.

Since VCF file contains multiple specimen information and if one or more of the specimen value does not exist in the CDM schema then that particular specimen data is not loaded and 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 in to ODB. The user has to make sure that the reference version matches between the results file being loaded and the reference data available in the ODB. ODB now supports multiple reference version, so VCF loader has to be given information as to which reference data is has to be mapped to. This version information present in VERSION_LABEL column of W_EHA_VERSION table has to be given to the VCF loader as a parameter. Please refer the loader parameter list for more details.

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

Details on 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 on 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 data type supported by VCF loader.

4.4.1.1 1000 genomes VCF4.1 version

The 1000 genomes VCF 4.1 format can be broadly classified in to 3 categories based on the type of variants as 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 existing in that table.

   The loader populates W_EHA_RSLT_NOCALL table based on the GT values having './.'. If there are consecutive records with this type of nocall genotype, then these records will be collapsed while loaded 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 START_POSITION as 1001 and END_POSITION as 1003.

2. **Large Structural Variation**: These large structural changes in genome are recorded in W_EHA_RSLT_STRUCT_VAR, W_EHA_VARIANT and W_EHA_VARIANT_X. The W_EHA_VARIANT and W_EHA_VARIANT_X tables are only populated for novel variants not existing 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.

   Currently any variant with ALT value more than 4000 characters will not be loaded. These records will be logged by the loader in W_EHA_RSLT_LOG table and also on the loader sqlplus console.

3. **Structural re-arrangement**: Currently 1000 genomes data for structural re-arrangements is not yet released. There is neither detailed documentation nor proper examples in the 1000 genomes manual which would cover all scenario's of this data set. In view of this, the loader is built on the following assumptions considered from the little information available at 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 will identity structural rearrangements from 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 the 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, 'SVTYPE' tag, then this mutation is considered as either SNV or small indel and data is loaded to W_EHA_RSLT_SEQUENCING and W_EHA_RSLT_SEQUENCING _X (only if length of allele is greater than 500) , table.

- If the INFO column has 'SVTYPE=BND', then this mutation is considered as structural re-arrangement and records are loaded to W_EHA_RSLT_SV_BREAKEND table.
- If the INFO column has 'SVTYPE' other than 'BND', for example, 'SVTYPE=DEL', then these mutations are considered as large structural variants and records are loaded to W_EHA_RSLT_STRUCT_VAR table.

4.4.1.2 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 the additional feature like siteConflicts. The VCF loader processes gVCF data in to W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING _X (only if length of allele is greater than 500) ,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 already existing in that table. The mutations are stored in W_EHA_RSLT_SEQUENCNG and W_EHA_RSLT_SEQUENCING _X (only if length of allele is greater than 500) , 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 W_EHA_RSLT_NOCALL table. Collapsing of nocall data for the consecutive positions is not done in gVCF load because the nocall data is already compressed in gVCF file based on the similar quality scores and other parameters defined while creating the gVCF file.

Following logic is used by the loader while populating gVCF records in target tables.

1. Any gVCF file record with GT value 0/1 or 1/0 or 1/2 or n/n, where n is not zero are stored in W_EHA_RSLT_SEQUENCING table.
2. Any record with GT value as 0/0 or just 0 are stored in W_EHA_RSLT_NON_VARIANT table.
3. Any record with GT value as '.' or './.' and ALT value as '.' is stored in W_EHA_RSLT_NOCALL table.
4. Any record with GT value as '.' and ALT value not '.' is stored in W_EHA_RSLT_CONFLICT table.

4.4.1.3 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 ie., SNP and small indel, large structural variation, structural re-arrangement and gVCF data. Since there is just one loader for loading all these data types, the loader 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 determines the input file type and the mode in which user want to load this data.

These parameters are:

1. **FILE_TYPE_CODE**: The 'FILE_TYPE_CODE' parameter is used to provide the file type information based on which the file type related package is called internally by the loader. If a user is loading a VCF file containing either SNP and small indel or large SV or SV-rearrangements, then user has to give FILE_TYPE_CODE as 'VCF' and if user is loading a gVCF file then FILE_TYPE_CODE should be given as 'GVCF'.

2. **LOAD_MODE**: The VCF loader has options to load the VCF and gVCF data in 3 different modes using the parameter 'load_mode'. Following are the three types of modes identified by the loader

   a. **'VCF' mode**: This mode can be used for both VCF and gVCF file types. This mode will only load mutations and nocall data and will only populate W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING _X (only if length of allele is greater than 500) , and W_EHA_RSLT_NOCALL tables respectively. If a user gives FILE_TYPE_CODE as GVCF and LOAD_MODE as VCF, then all the non-variant and conflict records from the gVCF file will be skipped and only variants and nocalls will be loaded.

   b. **'GVCF' mode**: This mode will load data to all the tables made for gVCF, ie., W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING _X (only if length of allele is greater than 500) , 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 like variants, non-variants, nocalls and conflicts, this mode is best suited for gVCF file type. However, if user has a VCF file with all non-variant information for a specific genome, then they can also use this mode to load the data. 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. For such files it is not advisable to load it using GVCF mode because the user will end up loading incomplete non-variant information for that specimen.

   c. **'NON-VAR' mode**: This mode will load data to only W_EHA_RSLT_NON_VARIANT and W_EHA_RSLT_NOCALL tables. This mode is designed for scenarios like the user has already loaded mutations through the VCF file and now they want to load only the non-variant information, then they can use this mode using a gVCF file. Just like the GVCF mode this mode is also mostly suited for gVCF file type as the gVCF file contains all non-variant information. Furthermore, it is not advisable to load a VCF file which doesn't contain all the non-variant information for a specimen in NON-VAR mode as incomplete non-variant information will be stored in the database.

4.4.2 Custom Format Specification in VCF

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

Custom format option helps user to load certain VCF FORMAT column fields which are currently not mapped in ODB. Before executing the loader the user has to manually create a column in W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SV_BREAKEND and W_EHA_RSLT_STRUCT_VAR tables. For Exadata, the staging tables like W_EHA_STG_SEQUENCING, W_EHA_STG_SV_BREAKEND and W_EHA_STG_STRUCT_VAR 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. If a user wants to map a 'PL' data type from the FORMAT column in the input file, then user has to create a column with the field name 'CUST_PL'. Once user creates this column, give the details of the mapping to the loader under 'custom_format' parameter for the loader as "PL=CUST_PL". In case the user wants to load multiple custom format columns, give the values as comma separated, for example, "PL=CUST_PL,GL=CUST_GL".

There is now no limitation on number of custom formats supported by the loader as such, but the loader can read only 32 format data types at a time, so if a custom format

data type is beyond the 32 data type then that data type will not be loaded. The order of the custom columns created should be same in the main tables and in the staging tables, otherwise there could be a mismatch in the data loaded. The custom column is advised to be added 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.

This is a global change and having a global PRODUCT_PROFILE should always map to the correct DDL structure of the SEQUENCING table. The loader stored procedure does allow for per-load override of the mapping. There is an extra parameter in the call to process_vcf named i_custom_format that allows the user to use a per-load mapping. But in all practicality, this would not be used in a real situation.

The reason being is that the SEQUENCING table could end up with different data mapped to user defined columns. So it will always be recommended that the users set the proper mapping in the PRODUCT_PROFILE table. In PRODUCT_PROFILE table, if there are 2 or more custom formats (For example, 'CUST_PL' & 'CUST_GL'), the VCF_FORMAT column should be inserted with the value 'CUST_PL, CUST_GL'.

4.4.3 Data Load

The execution call of the stored procedure `odb_result_util.process_vcf()` is designed in one of the script files (load_vcf.sql). file. This stored procedure accepts FILE NAME, ORACLE DIRECTYORY OBJECT, SPECIES NAME, STUDY NAME, DATA SOURCE, SPECIMEN VENDOR, REFERNCE VERSIOn, FILE FLAG, CUSTOM FORMAT (for new columns), PREPROSECC DIR, PREPROCESS FILE, DATA FILE PATH, DBFS STORE, FILE TYPE CODE, LOAD MODE, XREF DB, FILE VERSION, ALT FILE LOCATION and READ SIZE as an input parameter.

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

The loader creates 3 external tables, one to store metadata, one to store the specimen information and one to store the actual data of specimens for result table.

Dynamic SQL first parses the header row in the data file and stores it in an external table named gvcf_spec_ !!SEQ!! that contains nine header columns.

The loader then creates an external table to store the metadata of result file. The metadata of result file resides in header part of 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.

Looking at the metadata of the file, most of this data is in XML format where there is some identifier or tag followed by an attribute value or XML definition. The metadata load will set the QUALIFIER_TAG to the identifier before the "=" character (i.e. FORMAT, INFO, FILTER) and everything after the equal sign will be copied to the QUALIFIER_VALUE column of W_EHA_RSLT_FILE_QLF table.

The next external table declaration for data had 10 columns existing for the static VCF fields including row number. This means that the table declaration can allow for 986 specimen columns. There is a check in the loader for any file that has more than 986 specimens to give an error. Users can use various command line tools (awk, cut, etc...) to create files with appropriate number of columns if a file has more than 986 columns.

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 was broken up into 3 separate strings. The statements used now allow for the list of specimen columns to be added as a separate string. This allows for more than 32K statement to create the external table.

The loader retrieves the list of specimens from external tables and will store in the dynamic array. Also it validates and parses the flex fields and stores in an array.

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

The loader will first process 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 that overlap value, the reference sequence and the allele sequence is shortened and the start position and end position is incremented. Also, this overlap value 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 which populates the W_EHA_VARIANT table.

The loader then process to parse the first set of result data which do not use W_EHA_ VARIANT foreign key. This includes W_EHA_RSLT_NOCALL, W_EHA_RSLT_NON_ VARIANT, W_EHA_RSLT_SV_BREAKEND tables.

The loader then process to parse to link all records to W_EHA_VARIANT table. This includes W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING_X(if allele length is > 500), W_EHA_RSLT_STRUCT_VAR.

The loader process a separate pass to parse the conflict data.This process populates the W_EHA_RSLT_CONFLICT table.

At the end of all data parsing, a loader will call ext_tables_error_log() procedure which will parse all rows that have ALT_SEQ columns larger than 4000 characters those are actually not processed and logs an error in W_EHA_RSLT_LOG table.

The loader code is design in such a way that most of the parsing process shares some common code. The code declares some constant variables which compute the data for gene wid and variant wid. Two constant variables c_insert_result_nonvar and c_ insert_result_var defines which generates the dynamic inert statement for Non variant and Variant records respectively.

The code also creates a temporary table which stores the gene information after computation using first 9 fields of result file. This temporary table which loaded the possible set of gens then tied up with specimen to populate the gene_wid for variant as well as for non variant records.

A string constant defined get each value from the INFO column that calculate format offset positions also, is parsed in dynamically for each field into the cursor separately.

4.4.3.1 Data Files

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

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

Following are 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 then are stored in the W_EHA_RSLT_FILE_QLFR table.
2. The file is parsed to create all referenced W_EHA_VARIANT records. Note that this pass does not require "GT" format field so that reference VCF files can be loaded.
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, W_EHA_RSLT_STRUCT_VAR.
5. In GVCF mode, the file is parsed to add W_EHA_RSLT_CONFLICT records.
6. The last validation pass checks for all rows that have ALT_SEQ columns larger than 4000 characters to log warnings.
7. The loader can read only 32 data types from the FORMAT column in the VCF file. Any data type either supported or custom data types 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.

Table Mapping of VCF Result File (snps, indels, large SVs, and re-arrangements) 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_STRUCT_VAR.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 needed to find existing VARIANT records. The chromosome, the POS, and the replace tag, which is notation combining reference and allele sequences. For NOVEL variants, a new record is created in the W_EHA_VARIANT table with chromosome value.

Column Name in Result File	Table and Column Name in ODB	Description
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_STRUCT_VAR.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 columm. This value in ID column is stored in BREAKEND_ID column.
REF	W_EHA_VARIANT.REPLACE_TAG	This value is used in conjunction with ALT sequence to construct a replace tag value used to find existing VARIANT records. The replace tag is constructed with reference first followed by "/" and then the allele sequence. Note that for insertions, the reference sequence uses a "-" and for deletions the allele sequence uses "-". This is standard notation used in most references. At some in-dels, the representation can be as follows: ins can be AT/ATCTA and del can be ATCTA/AT. The logic is implemented in the procedure which can be called with any of the above formats.

Column Name in Result File	Table and Column Name in ODB	Description
ALT	W_EHA_RSLT_SEQUENCING.ALLELE W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_RSLT_CONFLICT.ALLELE W_EHA_RSLT_CONFLICT.ALLELE_CLOB W_EHA_RSLT_CONFLICT.ALLELE_CLOB W_EHA_RSLT_STRUCT_VAR.ALLELE W_EHA_RSLT_STRUCT_VAR.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. If the value of ALT is greater than 500 char then it is stored in W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB, W_EHA_RSLT_CONFLICT.ALLELE_CLOB and W_EHA_RSLT_STRUCT_VAR.ALLELE_CLOB.
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 W_EHA_RSLT_STRUCT_VAR.END_POSITION	For gVCF and large SV data, END_POSITION value using 'END=' tag present in this INFO column.
INFO.CIPOS	W_EHA_RSLT_STRUCT_VAR.CIPOS_START W_EHA_RSLT_STRUCT_VAR.CIPOS_END	For large SV, INFO column contains tag 'CIPOS' which contains two values. First value is stored in CIPOS_START and second value is stored in CIPOS_END.
INFO.CIEND	W_EHA_RSLT_STRUCT_VAR.CIEND_START W_EHA_RSLT_STRUCT_VAR.CIEND_END	For large SV, INFO column contains tag 'CIEND' which contains two values. First value is stored in CIEND_START and second value is stored in CIEND_END.

Column Name in Result File	Table and Column Name in ODB	Description
INFO.HOMLEN	W_EHA_RSLT_STRUCT_VAR.HOMLEN	For large SV, the value for the tag 'HOMLEN' present in INFO column is stored here.
INFO.HOMSEQ	W_EHA_RSLT_STRUCT_VAR.HOMSEQ	For large SV, the value for the tag 'HOMSEQ' present in INFO column is stored here.
INFO.MEINFO	W_EHA_RSLT_STRUCT_VAR.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 re-arrangement 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 re-arrangement data, if 'IMPRECISE' tag present in 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_STRUCT_VAR.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 is 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_STRUCT_VAR.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_STRUCT_VAR.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_STRUCT_VAR.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_STRUCT_VAR.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_STRUCT_VAR.ALLELE_READ_COUNT W_EHA_RSLT_STRUCT_VAR.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

Column Name in Result File	Table and Column Name in ODB	Description
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_STRUCT_VAR.RMS_BASE_QUAL W_EHA_RSLT_SV_BREAKEND.RMS_BASE_QUAL	RMS base quality at this position.
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_STRUCT_VAR.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_STRUCT_VAR.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}

4.4.4 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 "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 "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]

`-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

4.4.5 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 null
-preprocess_dir null -preprocess_file null -data_file_path null -dbfs_
store null -file_type_code "GVCF" -load_mode "GVCF" -alt_file_loc null
-read_size null
```

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 null
-preprocess_dir null -preprocess_file null -data_file_path null -dbfs_
store null -file_type_code "GVCF" -load_mode "GVCF" -alt_file_loc null
-read_size null
```

For file type VCF and load mode as VCF above same command will be used, only user will have to pass the -file_type_code as VCF an d -load_mode as VCF as a parameter with SH and BAT command.

4.5 MAF Sequence Data Loader

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

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

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

4.5.1 Functional Description

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

- If one of the allele is WT while the other is variant then the loader only records 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 as two separate records

The loader takes the chromosome and position details of a record from MAF file and checks if the corresponding region of that chromosome exists in W_EHA_DNA_SOURCE table for a specimen reference version specified as the user input for the loader. If it is present, it 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 does not log in to W_EHA_RSLT_LOG table.

The loader does not validate for invalid chromosome number or positions details. If it encounters such invalid data the loaders ignores that record and does not log it 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. You have to ensure that the reference version matches between the results file being loaded and the reference data available in the ODB.

A single record of MAF file has data for both normal and tumor sample. The loader loads data for both these samples.

Since MAF file contains multiple specimen information and if one or more of the specimen value does not exist in the CDM schema then the loaders skips that row and logs the error with deatils in W_EHA_RSLT_LOG .

4.5.2 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 an input parameter.

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

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

Another bulk insert statement then inserts the data into W_EHA_RSLT_SEQUENCING, W_EHA_RSLT_SEQUENCING _X.

A select statement which parses the data from the external table uses a join with w_eha_variant and w_eha_dna_source table which gets the dataset which will then used to compute the gene_wid from w_eha_gene_segment table. Dataset returned with variant and dna source table checks whether the start position of variant record is less than or equal to the end position of gene segment. It also checks whether the End position variant record greater than or equal to the start position of gene segment and populate the GENE_WID in W_EHA_STG_SEQUENCING table.

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

4.5.2.1 Data files

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

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

The W_EHA_RSLT_SEQUENCING _X tables only populates rows for which the length of allele is computed greater than 500. A loader validates the reference version which is passed as the input parameter against the W_EHA_VERSION table and populates the VERSION_WID in corresponding result table.

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

Table Mapping of MAF Result File

Column Name in Result File	Table and Column Name in ODB	Description
Chromosome	W_EHA_RSLT_SEQUENCING.CHROMOSOME_WID W_EHA_VARIANT.CHROMOSOME	This field is used with the begin position to find the correct DNA_SOURCE record, to find a VARIANT record, or create a VARIANT record. 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. This value is used in conjunction with Tumor_Seq_Allele1 and Tumor_Seq_Allele2 to find existing VARIANT records for tumor sample. Similarly for normal sample, the Reference allele is used for REPLACE_TAG. Note that for insertions, the reference sequence uses a "-" and for deletions the allele sequence uses "-". This is standard notation used in most references. Logic for loader will be implemented in called procedure to variant table. For deletion, this value has deleted sequence and for insertion it has "-".
Tumor_Seq_Allele1	W_EHA_RSLT_SEQUENCING_X.ALLELE/ W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_VARIANT.REPLACE_TAG	For sequencing results this value constructs the replace tag. '-' value represents a deletion.
Tumor_Seq_Allele2	W_EHA_RSLT_SEQUENCING_X.ALLELE / W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_VARIANT.REPLACE_TAG	For sequencing results, this value 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_RSLT_SEQUENCING_X.ALLELE / W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB 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_X.ALLELE / W_EHA_RSLT_SEQUENCING_X.ALLELE_CLOB W_EHA_RSLT_SEQUENCING.ALLELE W_EHA_VARIANT.REPLACE_TAG	For sequencing results this value constructs the replace tag. '-' value represents a deletion.

Column Name in Result File	Table and Column Name in ODB	Description
Score	W_EHA_RSLT_ SEQUENCING.SCORE_VAF	This is mapped to W_EHA_RSLT_ SEQUENCING.SCORE_VAF.

4.5.3 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 "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 "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

4.5.4 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 null -preprocess_file null
-data_file_path null -dbfs_store null -alt_file_loc null -read_size null
```

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 null -preprocess_file null
-data_file_path null -dbfs_store null -alt_file_loc null -read_size null
```

4.6 CGI masterVar Loader

CGI masterVar loader has been removed from Omics Data Bank 2.5 and will be reintroduced in future ODB version.

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

Currently, support is provided for loading of the exon version of the data files; 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.

Once the exon data file is loaded, the user can choose genes which then can map to specific chromosome regions. RNA Sequencing-based data have 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, but from ODB v2.5 onwards, only files with this column can be loaded.
- exon: Gives 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 exon quantification file inputs the data into the following table: W_EHA_RSLT_RNA_SEQ.

The last four columns of the TCGA exon files are populated into the two tables 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 exact result table value is stored in RESULT_EXON_NAME in W_EHA_RSLT_RNA_SEQ for user reference.

The table will store an additional FK value for SPECIES, a FK value for the version of DNA reference mapped to the result file, and a FK value to 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 input into the column. If the RPKM value for a input row is a null value (a blank) or has the text 'null' then this row 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, 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.

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

Only one external table is created dynamically and will hold the complete result data.There is one bulk insert statement which will insert data into W_EHA_RSLT_RNA_SEQ table. The query uses the 2 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 2 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 due to the gene record identifiers having a different querying requirement and hence separated in 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 batch file of the RNA-seq loader requires Oracle Wallet to be set up to run correctly.

Specimen number -

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

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

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 it 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.
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, if the user does not pass the sample number 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 "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 "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. 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 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

4.7.4 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 ''
```

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"
```

4.8 File Lineage Linker

File lineage linker has been removed from Omics Data Bank 2.5 release and will be reintroduced in the next release.

4.9 Copy Number Variation Loader

4.9.1 Functional Description

The CNV loader is meant to load 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:

Table Extract of input CNV file

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.

This stored procedure creates an external table 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 stored procedure creates cnv_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 run time.

There is a single bulk insert statement written. This inserts the records into the W_EHA_STG_COPY_NBR_VAR table.

A select statement which parses the data from the external table uses an inline query which gets the dataset of gene segment records. The query lookups with the dataset returned from the inline query and checks whether the start position of result file is less than or equal to the end position of gene segment + start position of DNA source. It also checks whether the End position of result file greater than or equal to the start position of gene segment + start position of DNA source. The dataset of both the inline queries are then outer joined with the ROW_WID of external table lookup for GENE_WID and populate the records in W_EHA_STG_COPY_NBR_VAR table.

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

Note: The batch file requires 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 "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 "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. 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 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

Single channel gene expression loader loads gene expression data into W_EHA_RSLT_GENE_EXP table. The loader starts 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 it has a maximum of 15 hybridization sets (which translates to maximum of 45 columns. If it doesn't have then it uses empty fillers in the staging table for the unavailable hybridization sets. Then for each record from the staging table, it verifies if the probe name exists in the W_EHA_PROBE table along with matching version and species id which are passed as input parameters.

If it matches, then it inserts all the sets of hybridizations available for that record into W_EHA_RSLT_GENE_EXP table excluding the empty fillers. If probe name doesn't exist in the W_EHA_PROBE table, then it skips that record and if probe exist with unmatched version and/or species id, then it logs a warning into W_EHA_RSLT_LOG table saying that version and/or species do not match.

The W_EHA_RSLT_LOG table will contain error records if the 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 the insert of this record.

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

4.10.2 Data Load

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

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

The input file for this loader should contain normalized intensity values, and optional inputs of present/absent calls and P-value (such as the output of Affymetrix's MAS5 algorithm). The input file allows for multiple hybridization intensity data in a tabular format. The following section lists out the specific of the format of the input data file.

4.10.2.1 Assumptions for Data File

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

1. The file is tab separated.
2. The first row is always the header.
3. The first column is named DATA.
4. Each hybridization present in the data file should have three columns in the following order:
   - Intensity - Header value should be the Hybridization Name
   - Call
   - P-Value
5. The first column for each hybridization should contain only the hybridization name. The values in this column will be the hybridization intensity value.
6. 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

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

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

4.10.3 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 "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 "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 null -preprocess_file null -data_file_path
null -dbfs_store null -alt_file_loc null -read_size null
```

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 null -preprocess_file null -data_file_path
null -dbfs_store null -alt_file_loc null -read_size null
```

4.11 Dual Channel Loader

4.11.1 Functional Description

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 symbol 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 is used to link the Dual Channel data with with genes with the specific DNA Reference Version via 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 matches 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 input of 2channel result data with this loader.

If the Log2 ratio value for a input row is a null value (a blank) or has the text 'null', then this row is skipped by the loader on 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.

This stored procedure creates an external table 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 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 statements written dynamically. This inserts the record into the W_EHA_STG_2CHANNEL_GXP table.

A select statement which parses the data from the external table uses an inline query which gets the dataset of gene segment records. The query lookups with 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 the inline queries is then the outer join with the ROW_WID of external table to lookup the GENE_WID and populates the records in W_EHA_STG_2CHANNEL_GXP table.

Note: The batch file requires 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 "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 "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 Typical Errors Associated with Result Loaders

4.12.1 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 4-1.

Table 4-1

Table 4–1 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
ERROR_DESC	Error message	ORA-01403 NO DATA FOUND
RECORD_DETAIL	Verifying Study Name	-
ETL_PROC_WID	Each load identifier.	-

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

4.12.2 VCF Loader Errors

VCF/gVCF loader processes the loaders in multiple passes. The loader creates 3 external tables. If the process fails while processing specimen specification table an error 'Processing external specimen table for specimen headers' logs in w_eha_rslt_log table. If the process fails while creating metadata external table an error 'Processing external table for metadata' logs in result log table. If the process fails while creating qualifier records an error 'Processing metadata records(w_eha_rslt_file_qlfr)' logs in result log table.

If the process fails while retrieving specimen from external table an error 'Retrieving specimen numbers from external specimen table' logs in result log table. If the loader tries to load the file which has more than 986 samples then the error 'File loaded cannot have more than 986 specimens' log in 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)' log into result log table. If the process fails while populating variant table from variant staging table an error logs 'Processing variant records(w_eha_variant, w_eha_variant_log)'; in the result log table.

If the process fails while populatin variant result records an error 'Processing result sequencing records(w_eha_stg_sequencing, w_eha_stg_sequencing_x, w_eha_stg_ struct_var) logs in result log table and if fails while processing non variant records then 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)' in result log table. If the process fails while populating conflict records then an error 'Processing result sequencing records(w_eha_stg_conflict)' log in result log table.

If the loader processes the xref records and if the process fails then an error 'Processing variant xref records(w_eha_variant_xref)' logs, if the process fails while computing nocall collapsing function then an error 'Collapsing result nocall staging records(w_ eha_rslt_nocall) logs in esult log table.

At the end of all data parsing, a loader will call ext_tables_error_log() procedure which will parse all rows that have ALT_SEQ columns larger than 4000 characters those are actually not processed and logs an error 'Log failed external table records' logs in W_ EHA_RSLT_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_rslt_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'.

4.12.3 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 error 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 error 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 error log table.

If the process fails while executing bulk insert statement which populates the target result table an error 'Processing result records (w_eha_stg_sequencing, w_eha_stg_ sequencing_x)' getting logged into error 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_rslt_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'.

4.12.4 Single Channel Gene Expression Loader Errors

If the process fails while retrieving any of: DNA version ID, specimen ID, datasource name or study ID, an error is logged into the error table and is shown in the error summary at the end. If the process fails while processing the hybridization header external table, the Processing hybridization header table 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 expression and hybridization external tables, error messages Dropping expression data table and Dropping hybridization table respectively are logged.

4.12.5 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'.

4.12.6 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'.

4.12.7 Copy Number Variation Loader Errors

Only one external table is created for CNV result data. When the process fails during creation of 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 Collecting Oracle Optimizer Statistics

Oracle statistics is a collection of data about 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 comparing to a size of a schema, it is not required to gather statistics after each load. Instead, periodic statistics gathering on a weekly schedule are recommended. 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 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 that 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—The researcher should be able 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—Show me 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 need 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') — Give 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 of 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')
```

— Give 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'))
```

— Give 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') — Give 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 - The researcher should be able 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 - Show me 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. Here a random list is taken.

```
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'));
```
— Give 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 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 rs1, W_EHA_GENE g, W_EHA_VERSION V

WHERE

RS1.GENE_WID = g.ROW_WID

AND g.HUGO_NAME IN ('ADAM32')

AND RS1.version_wid = V.ROW_WID

AND V.VERSION_LABELin ('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 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 version numbers 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. It does not use a loader to populate it, and it 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 various logging levels allowed for log records inserted into the log table. It also stores global setting to allow 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 VARCHAR2(1 CHAR);

LOG_INFO VARCHAR2(1 CHAR);

LOG_DEBUG VARCHAR2(1 CHAR);

LOG_TRACE VARCHAR2(1 CHAR);

LOG_DBMS_OUTPUT VARCHAR2(1 CHAR);

VCF_FORMAT VARCHAR2(4000 CHAR);

FLANKING_OFFSET parameter is an input (right after PROMOTER_OFFSET) on running TRC install scripts. The FLANKING_OFFSET value is used by various loader procedures and defines the size of the region before and after gene definition that sequencing results are to be linked with . 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 or larger than the value used for PROMOTER_ OFFSET, so that 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. DBMS output is by default 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 '**Instructions on Custom Format Specification in VCF'** under chapter sub-section '4.6.1 Functional Description of VCF Loader'.

7.2 Querying Database Cross-References for Variations

7.2.1 Ensembl db_xref Qualifier Issue

The ENSEMBL GVF file, which involves nucleotide variation references from dbSNP, COSMIC and EMBL, is used to populate the variation tables in Omics Data Bank. The cross-reference information for these variants is identified by the Dbxref qualifier and is loaded onto W_EHA_VARIANT_XREF table. The standard format for Dbxref in 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. Hence, for the above example W_EHA_VARIANT_ XREF, populate columns with following data:

```
DATABASE = 'dbSNP_132'
```

```
REFERENCE_ID = 'rs79772382'
```

However, for some organisms, Dbxref is defined differently. Following is an example from 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 contain suffixed or prefixed data for some of the cases mentioned above, when querying the REFERENCE_ID, Oracle recommends using the SQL LIKE operator.

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

The same scenario exists for the SwissProt database cross-reference. Hence, 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 stored 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 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 which 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.
```

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

There is some loss of information on the REFERENCE_SUFFIX level but not on REFERENCE_ID, that is, all REFERENCE_IDs are captured in the W_EHA_PROT_XREF table. Also, all instances of not saving duplicate db_xrefs are now logged by the SwissProt loader as warnings.

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 table) 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. When inserting into result tables, that is W_EHA_RSLT_COPY_NBR_VAR and 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

The promoter region information is not available in the reference data set imported from ENSEMBL. Therefore, a column has been provided in W_EHA_SPECIES table for you to specify the promoter region upstream to the gene for a specific organism. 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. So 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 allow querying promoter regions for genes/gene structures in ODB.

7.5 Loader Activity Logging

The W_EHA_PRODUCT_PROFILE logging settings are singular levels, where each setting affects 1 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 run of a loader. 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, like "INHERIT", logging levels are inherited from W_EHA_PRODUCT_PROFILE.LOG_% settings. The following named command-line argument,

-print_summary ['1'=on, '0'=off] default: '0' if on, logs a summary log record to the database and prints the summary to the console.

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 pl/sql package, operation and parameter list
END	No, always on	Displays the status of pl/sql 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 of loader runs is provided to the user in the form of summary log records that is printed at 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 to every ETL load 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);

The 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_rslt_file: 1 row

w_eha_rslt_file_spec: 1 row

w_eha_rslt_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 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, and these new gene component types can be made available in the TRC v2.0.1 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. Else, 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 is 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. When this is set, you can query by Pathway name, and by 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 in this table per Gene Component. 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 more than one segment 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 (filled 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, you must ensure that it does not coincide with any existing or future value 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, if we want to add a First exon in a gene component type, and the customer company name is XYZ, we can use the XYZ_first_exon as the new component type.

We now have the component type we can use, and we can use it with the same letter casing for all first exons of genes that we add.

Next, we provide the actual components. We already have a superset of the data (exons) in the W_EHA_GENE_COMPONENT table, with their segments already defined, and everything correctly linked to DNA Sources, Gene Structures, and Proteins. All we need is an extra set of records 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 can be 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 or when re-installing the same database.

The previous example was a case of copying the already existing components. For a general case, perform the following steps:

After selecting an existing or choosing a 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, more than one segments need to 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 PL/SQL) 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 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. 5.Next, yo must 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 the case 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 is a further complication here - there may be more than one matching DNA Source - and we need 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.

Here is some practical information, useful when creating custom gene components.

Tables and sequences (when inserting into a table, get the new ROW_WID from the appropriate sequence):

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(10), is a column in all ODB tables that needs to 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, typically about 1,000,000 bases per DNA Source.

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. If it does not, but 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.

If we get our 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.

7.7.1 Creating Custom Gene Region Views

The end user can build queries 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.

The following are the guidelines for creating custom views:

The custom gene region views must be named following the TRC 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, W_EHA_PROMOTER_REGION_V, can be used as an example when creating custom gene region views. It is used by the TRC application 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.

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_ || <the 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.

7.7.2 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 1st 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))
```

7.8 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. This script accepts a parameter which asks about the degree of parallelization for the 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 real result table. For better compression run the script after a minimum of 50 samples results data has been loaded into the staging tables.

The load_exadata script takes the following parameters:

- Username: to connect to the database 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.

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 2.5 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, inserts and deletions.	4.1
MAF	Mutation Annotation Format	Mutation Annotation Format containing snps, inserts, and deletions.	2.0
MAF	Mutation Annotation Format	Mutation Annotation Format containing snps, inserts, and deletions.	2.1
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, inserts, deletions, and no-call information.	2.0
TCGA RNA SEQ EXON	TCGA RNA SEQ EXON	TCGA RNA Seq tab delimited file format for exon information.	3.1.4.0
Genome_Wide_SNP_6	Affymetrix Genome-Wide Human SNP Array 6.0	This file represents the TCGA data format for Affymetrix Genome-Wide Human SNP Array 6.0'.	6.0
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
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	Binary sequencing alignment file for sequencing runs.	1.4
SAM	Sequence Alignmentor Map Format	Binary sequencing alignment file for sequencing runs.	1.4
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

W_EHA_CHROMOSOME

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

Table A–3 W_EHA_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 alias 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_PREDICTION_CODE

This table is pre-seeded with all the prediction codes for SIFT or PolyPhen annotation loader support

Table A–4 EHA_PREDICTION_CODE

CODE	CODE_TYPE	CODE_NAME
tolerated	SIFT	tolerated
deleterious	SIFT	deleterious
unknown	polyphen	unknown
benign	polyphen	benign
possibly damaging	polyphen	possibly damaging
probably damaging	polyphen	probably damaging

A.2 Populated by User or Loader

W_EHA_DATASOURCE

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

Table A–5 W_EHA_DATASOURCE

Table Name	Column Name	Description	If used together with Oracle Health Sciences Cohort Explorer (OHSCE) Cohort Data Model
W_EHA_DATASOURCE	DATASOURCE_CD	Data source for Specimen	CDM
W_EHA_DATASOURCE	DATASOURCE_NM	Name of datasource for Specimen	Cohort Data Model
W_EHA_DATASOURCE	DATASOURCE_DESC	Description of datasource for specimen	Cohort Data Model
W_EHA_DATASOURCE	SCHEMA_NAME	Name of schema	CDM
W_EHA_DATASOURCE	DB_LINK_NAME	Link to database if needed	

This table is populated by reference loaders while loading new versions of reference files. Each reference loader can populate a version record for a particular version type associated with it.

Table A–6 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

W_EHA_RSLT_FILE

This table is populated by loaders while loading results.

Table A–7 W_EHA_RSLT_FILE

Table Name	Column Name	Description	If Used Together With Regular Files	If Used Together With SecureFiles
W_EHA_RSLT_ FILE	FILE_STORAGE_ FLG	E for External, S for SecureFiles	E	S
W_EHA_RSLT_ FILE	FILE_PATH	Path to input file	For example, C:/inputfile.txt	
W_EHA_RSLT_ FILE	VENDOR_NAME	Name of vendor providing file	For example, Affymetrix	For example, Affymetrix
W_EHA_RSLT_ FILE	FILE_CONTENT_ ID	File identifier utilized by Secure File system	Not used	Generated by SecureFiles
W_EHA_RSLT_ FILE	FILE_TYPE_WID	FK to W_EHA_ RSLT_FILE_TYPE	Corresponds to WID in RSLT_ FILE_TYPE	Corresponds to WID in RSLT_FILE_TYPE

W_EHA_RSLT_FILE_SPEC

This table stores the foreign key to W_EHA_RSLT_FILE and W_EHA_RSLT_SPECIMEN and links a specific file which is loaded to ODB through the loaders to the specimen it belongs to.

W_EHA_RESULT_STUDY

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

Table A–8 W_EHA_RESULT_STUDY

Table Name	Column Name	Description	Values Pre-seeded
W_EHA_RESULT_STUDY	RESULT_STUDY_NAME	Name of the study	<user-defined values>
W_EHA_RESULT_STUDY	RESULT_STUDY_DESC	Description of the study	<user-defined values>

A.3 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–9 Unpopulated Tables or Columns

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

Table A–9 (Cont.) Unpopulated Tables or Columns

Table Name	Column Name	Description
W_EHA_SOMATIC_VAR_INFO		No loader for this table
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_UNIT_OF_MEASURE		One of QC metadata tables
W_EHA_VARIANT_QLFR		Additional variant qualifier table

Index