SNP System Documentation

Author:sc

Created: May 10, 2006

Last Modified: September 6, 2006 09:41

1 Purpose of Document

To describe the SNP database architecture, data model, dbSNP data, products used to load the data, system data flow, and job scheduling.

2 Introduction

With dbSNP Build 125, based on mouse genome build 34, we had to redesign the snp schema and load process. With the expectation that snp data will increase at an ever increasing rate, we redesigned our database architecture and data model to accommodate. We moved the snp tables from mgd to a new snp database to decouple the snp data as much as possible from the rest of MGI. We did this to minimize table size (e.g. snps have their own Accession table), for efficient indexing and querying and allow snp loads to run concurrently with curation and other loads.

3 Definitions

- snp database - most snp data, excluding vocabularies and translations, is encapsulated in tables in this new database. See Section 4.2, “Data Model” on page 6.
- snp front-end database - the snp database used by the production WI, currently on the unix server shire and known as PROD_MGI..snp.
- snp back-end database - the snp database we load, currently on unix server lindon and known as PROD1_MGI..snp.
- mgd front-end database - ‘production’, currently on the unix server shire and known as PROD_MGI..mgd.
- In-sync mode - when snp coordinates are from the same mouse genome build as coordinates for other MGI objects, e.g. gene models, and markers.
- Out-of-sync mode - when snp coordinates are from a different mouse genome build than coordinates for other MGI objects.

4 Database Design

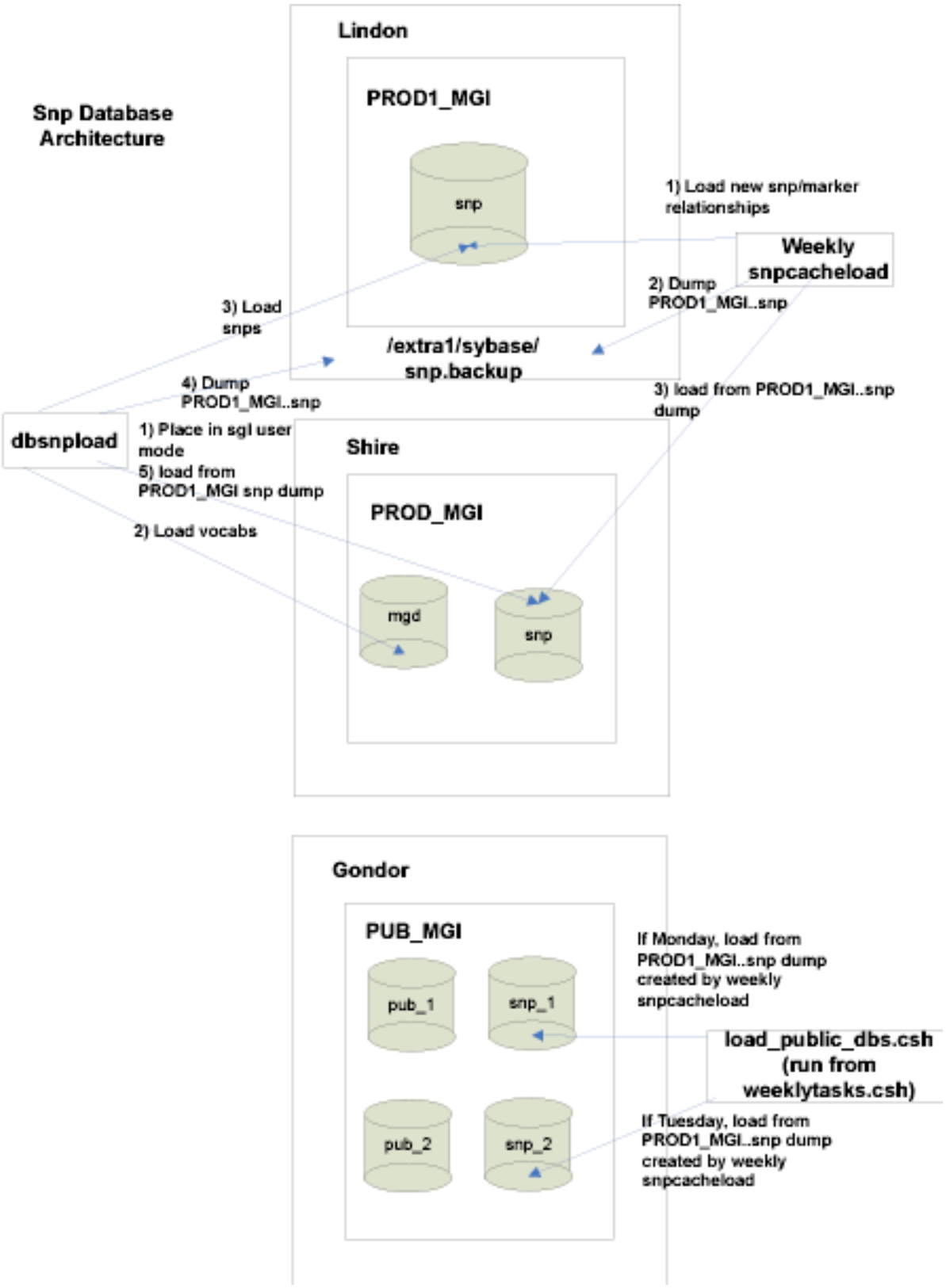
4.1 Database Architecture

4.1.1 Unix servers and their sybase servers/databases.

See Section 9, “Job Scheduling” on page 18 for overview of how these databases are loaded.

- lindon unix server
  - PROD1_MGI..snp - back-end snp database
- shire unix server
  - PROD_MGI..mgd - front-end mgd database (production)
  - PROD_MGI..snp - front-end snp database
- gondor unix server
  - PUB_MGI..pub_1, snp_1 - the live public databases on T/Th/Sat/Sun
  - PUB_MGI..pub_2, snp_2 - the live public databases on M/W/F
  - ADHOC_MGI - no snp database

4.1.2 Snp Database Architecture Diagram


Snp Database Architecture
Lindon
PROD1_MGI
snp
1) Load new snp/marker relationships
Weekly snpcacheload
2) Dump PROD1_MGI..snp
3) Load snps
/extra1/sybase/ snp.backup
4) Dump PROD1_MGI..snp
3) load from PROD1_MGI..snp dump
dbsnpload
1) Place in sgl user mode
5) load from PROD1_MGI snp dump
2) Load vocabs
Shire
PROD_MGI
mgd
snp
Gondor
PUB_MGI
pub_1
snp_1
If Monday, load from PROD1_MGI..snp dump created by weekly snpcacheload
load_public_dbs.csh (run from weeklytasks.csh)
pub_2
snp_2
If Tuesday, load from PROD1_MGI..snp dump created by weekly snpcacheload

4.1.3 Database Architecture Rationale

1. Separate snp database from mgd
   - Allows concurrent snp loading with curation and other loads. Requires exclusive access to production mgd database for a few minutes at the beginning to load new vocabularies. The snp loads also query mgd to create lookups, but this is done once for each lookup, with little impact on production.

     Note that the front-end snp database is not accessible from the production WI during a dbSNP load. We put it in single user mode because mgd snp vocabulary keys will be out-of-sync with those in snp. Curators can use public for snp queries.
   - ACC_Accession has 17.8 million records already. Build 125 would have added 6.7 million snp accessions, causing indexing and query performance issues.
2. Separate Sybase Server and Unix server for back-end snp database
   - So production performance isn't compromised during loads.
3. Front-end snp database on same sybase server as mgd
   - Allows the WI to query across the two databases. Note that querying across sybase servers is not currently an option.
4. Dual public snp databases on same sybase server
   - Pair snp_1 with pub_1, snp_2 with pub_2. Each WI running on public is configured for one set of databases. We load the inactive snp_database (snp_2) after the weekly snpcacheload runs early monday morning. We load snp_1 early tuesday morning.
   - We considered having snp and snp_load. Load snp_load then rename it to snp, but this requires both databases be in single user mode.
   - Same sybase server allows WI to query across the two databases.

4.2 Data Model

4.2.1 SNP Schema Document

4.2.2 SNP ER Diagram

4.2.3 mgd to snp - Overview of Schema changes

4.2.3.1 snp Database

1. Moved all SNP_* table from 'mgd' to 'snp', **including** SNP_ConsensusSnp_Marker. The WI will be made aware that some _Marker_key's may no longer exist in 'mgd' (this would be rare).
2. Removed _CreatedBy_key, _ModifiedBy_key, creation_date, modification_date from all snp tables. This saves space and index time. Use MGI_Tables _LoadedBy_key, and loaded_date to track changes to each table.
3. Added MGI_dbinfo snp_data_version to specify the dbSNP build number loaded.
4. Stopped loading MAP* tables, only used to load SNP_Coord_Cache; load SNP_Coord_Cache directly. This change means the schema currently no longer supports multiple snp assemblies. When we decide tolad multiple assemblies per RefSnp we need only to add an 'assembly' column to SNP_Coord_Cache.
5. Added Submitter Handle term as an attribute of SNP_Population because dbsnpload needs to create a lookup that requires getting the term for _SubHandle_key.
6. Removed SNP_Strain_Cache - This table was huge and querying was just as efficient using SNP_SubSnp_StrainAllele and SNP_ConsensusSnp_StrainAllele directly.
7. Added the following tables/view to 'snp':
   - SNP_Accession

     Similar to mgd..ACC_Accession, but excludes unnecessary columns such as preferred adn private.

     This table to hold all SNP accession ids; that is SNP_ConsensusSnp, SNP_SubSnp, SNP_Population accession ids and SNP_ConsensusSnp_Marker associations to RefSeq ids. It is truncated upon each new dbsnpload. It holds mgd..ACC_MGIType and mgd..ACC_LogicalDB keys.
   - SNP_Strain

     This table is created from PRB_Strain and contains only dbSNP strains. SNP_Strain will be reloaded only on a new dbSNP build. Since strain merges happen in mgd, the WI joins to this table rather than PRB_Strain to avoid dangling strain keys.

     Strain merges, in mgd, are handled by stored procedures which update all foreign strain keys in mgd upon each strain merge. Note this also applies to Markers. There is an issue referencing more than one database from a stored procedure or a view. Lori provided this explanation:

     - When a database object, such as a view or stored procedure, is created, it cannot reference environment variables. So, if we have a SP in mgd that needs to reference a table in ìsnpî, then we need to hardcode the reference to the snp database in the SP:

       ```
       create procedure MGD_Foo
       ...
       update snp..SNP_Foo
       ```

- When a binary dump of the "mgd" database is loaded into a development copy of "mgd", say "mgd_lec1", the SP in "mgd_lec1" is still referencing the "snp" database that was used when the SP was originally created in "mgd".
- If we wish the SP in "mgd_lec1" to reference a different "snp" database, say "snp_lec", then we would have to drop and re-create the SP in "mgd_lec1" and have it refer to "snp_lec":

```
create procedure MGD_Foo
...
update snp_lec..SNP_Foo
```

- If we forget to do this, then we will either get an error when we try to execute this SP in "mgd_lec1" (if "snp" does not exist on the development Sybase server) or it will run but will update data in the wrong "snp" database.

- SNP_Summary_View

  A view of the set of RefSNP and SubSNP accession ids in SNP_Accession.

- MRK_Location_Cache

  A copy of the mgd table is loaded to snp just prior to reloading MGI marker relationships.

4.2.3.2 mgd Database

1. Continue to use mgd for vocabularies and translations. snp database tables SNP_ConsensusSnp_Marker, SNP_ConsensusSnp, SNP_Coord_Cache, SNP_Population, and SNP_SubSnp will hold mgd _Term_keys because:
    1. The Function Class vocab is a DAG.
    2. Each vocab is static between dbsnploads.
    3. Note that the Submitter Handle vocabulary has accession ids and these will be stored in mgd..ACC_Accession.
    4. Note that translations from dbSNP terms to VOC_Term._Term_keys and PRB_Strain._Strain_keys (via MGI_Translation) will remain the same.

5 dbSNP Data

dbSNP Build 126 Data Document

6 Data Requirements

SNP Back-end Data Requirements Document

7 System Dataflow

This diagram shows data flow of the dbsnpload jobstream. See dbsnpload_dataflow.vsd

8 Products

8.1 dbsnpload

Used to load a new dbSNP build.

Description: Loads production mgd database with snp vocabularies and translations. Loads snp back-end database with dbSNP data, including dbSNP and MGI snp to marker associations. Dumps the snp back-end database and loads the snp front-end database.

This product consists of bourne shell and python pre- and post-processing scripts and a java DLA-Loader all managed from a bourne shell jobstream script..

Configuration: See mgiconfig, dbsnpload/dbsnpload.config, snpcacheload Configuration.

Jobstream Script: dbsnpload.sh [-s -f -v -h -p -c -r]

where:

-s DO NOT put front-end snp db in sgl user mode

-f DO NOT run the fxnClass vocload and translation load

-v DO NOT run the varClass vocload and translation load

-h DO NOT run the subHandle vocload

-p DO NOT run Population load - !warning! if you use this option
be sure to configure SNP_OK_TO_DELETE_ACCESSIONS=true

-c DO NOT run snpcacheload

-r DO NOT run post Processing

Description of dbsnpload jobstream processes. Invocation order is important.

8.1.1 Put snp front-end database (PROD_MGI..snp) in sgl user mode.

This disallows access since snp vocabulary keys will be out of sync with mgd when new vocabularies are loaded. Loops until it succeeds.

Product/Script: dbsnpload/bin/setSglUser.sh DBSERVER DBNAME mode outputfile sleepInterval. (Wrapper over mgidbutilities/bin/doSglUser.csh)

Where:

- mode - ‘true’ to set in single user mode, ‘false’ to turn off single user mode.
- outputfile - path to file used as output from doSglUser.csh. This output is parse to determine success/failure.
- sleepInterval - if failed, wait this long before attempting again.

8.1.2 Create Vocabularies

This script creates the dbSNP vocabularies, designated on its command line, in the production mgd database.

Product/Script: dbsnpload/bin/loadVoc.sh [-f -v -h] (This is a wrapper over vocload product**)**

Vocabularies:

1. Variation Class

   Vocabulary type: simple

   - Inputs:
     1. dbsnpload/varClassVocab.config.
     2. dbsnpload/data/varClass.vocab.in - input file for vocload.

        Format: (see vocload)

        - term tab tab tab tab tab tab tab tab

        For Example:

        - SNP tab tab tab tab tab tab tab tab
   - Outputs:
     1. bcp and log files in /data/loads/dbsnp/dbsnpload/output/vocload/varClass.
     2. VOC_Vocab and VOC_term records created in an mgd database.
2. Function Class

   Vocabulary type: DAG

   - Inputs:
     1. dbsnpload/fxnClassVocab.config.
     2. dbsnpload/data/SNP_FXN_DAG.ontology - input file for vocload created by DAG Editor. Function class terms that cannot be resolved are reported to the dbsnpload curation log so they may be added to the vocabulary input file.
     3. dbsnpload/data/SNP_FXN_DAG.def - input file for vocload. Supporting file created by DAG editor.
   - Outputs:
     1. bcp and log files in /data/loads/dbsnp/dbsnpload/output/vocload/fxnClass.
     2. VOC_Vocab and/or VOC_term, DAG_DAG, DAG_Node, DAG_Edge records created in an mgd database.
3. Submitter Handle

   Vocabulary type: simple

   Accession records are created for this vocabulary in order for the WI to link to dbSNP by submitter handle.

- Inputs:
  1. dbsnpload/subHandleVocab.config.
  2. /data/loads/dbsnp/dbsnpload/output/subHandle.vocab.in - input file for vocload.

     Format: (see vocload)
     - term tab accid tab tab tab tab tab tab

     For Example
     - ROCHEBIO tab ROCHEBIO tab tab tab tab tab tab
- Outputs:
  1. bcp and logfiles in /data/loads/dbsnp/dbsnpload/output/vocload/subHandle.
  2. VOC_Vocab and/or VOC_term, and ACC_Accession records created in an mgd database.

Troubleshooting: Most failures come from badly formatted input files or incorrect config files.

8.1.3 Create Translations

This script creates the dbSNP translations, designated on its command line, into the production mgd database.

Product/Script: dbsnpload/bin/loadTranslations.sh [-f -v] - (This is a wrapper over the translationload product). We do not translate submitter handles

Translations

1. Variation Class
   - Inputs:
     1. dbsnpload/varClassTrans.config.
     2. dbsnpload/data/varClass.goodbad - goodname/badname mapping created by curator (Richard). This maps the varClass name from dbSNP to its MGI term. e.g dbSNP term ‘named-locus’ is mapped to MGI term ‘Named’.

        Format: (see translationload)
        - tab goodName tab badName tab MGI_User..login

        Example:
        - tab Locus-Region tab locus-region tab dbsnp_load
   - Outputs:
     1. bcp and log files in /data/loads/dbsnp/dbsnpload/output/translationload.
     2. MGI_TranslationType and/or MGI_Translation records created in an mgd database.

2. Fxn Class
    - Inputs:
        1. dbsnpload/fxnClassTrans.config.
        2. dbsnpload/data/fxnClass.goodbad.

           Format: See variation class.
    - Outputs:
        1. bcp and log files in /data/loads/dbsnp/dbsnpload/output/translationload.
        2. MGI_TranslationType and/or MGI_Translation records created in an mgd database.

Troubleshooting: Most failures come from badly formatted input files or incorrect config files.

8.1.4 Create Populations

Product/Script: dbsnpload/bin/loadPopulations.sh

This script create SNP_Population objects and associates population ids to those objects via SNP_Accession. It greps lines from the input files, starting with the "<Population" tag and writes them to populations.txt (see 'Inputs' and 'Outputs' below). It then splits each line in populations.txt into the population name, population id, and submitterHandle in order to create the bcp files.

Inputs:

1. /data/downloads/ftp.ncbi.nih.gov/snp/mouse/genotype/*.xml.
2. /data/loads/dbsnp/dbsnpload/output/populations.txt.

Outputs:

1. /data/loads/dbsnp/dbsnpload/output/populations.txt.
2. The following bcp files written to /data/loads/dbsnp/dbsnpload/output and executed against the snp back-end database.
    - SNP_Population.bcp
    - SNP_Accession.pop.bcp
3. log file: writes to /data/loads/dbsnp/dbsnpload/logs/snpPopulation.log.

Troubleshooting:

1. All problem lines in /data/loads/dbsnp/dbsnpload/output/populations.txt it will be written to the log file and the script will exit with exit code 1. This happens when
    1. A line is improperly formatted. This is a dbSNP XML format problem.
    2. A submitter handle cannot be resolved. Add any new handles to the submitter handle vocabulary input file and run the submitter handle vocabulary load. See Section 8.1.2, "Create Vocabularies" on page 10. Populations and SubSnps have submitter handles.

8.1.5 Run java dbsnploader

Product/Script: call to DLA java application DLAStart (in lib_java_dla product) which runs the configured DBSNPLoader (in dbsnpload product). See dbsnpload.sh for invocation.

Java class descriptions - See dbsnpload_java_classes.pdf

Java class diagram - See dbsnpload_class_diagram.gif

This application parses dbSNP input files, resolves attributes where necessary, and creates/executes bcp files in the snp back-end database (See Outputs).

Inputs:

1. mgietc common.config.
2. dbsnpload.config.
3. /data/downloads/ftp.ncbi.nih.gov/snp/mouse/genotype/*.xml - strain alleles for SubSnps.
4. /data/downloads/ftp.ncbi.nih.gov/snp/mouse/XML/*.xml - RefSnp and SubSnp info (excluding strain alleles).

Outputs:

1. The following bcp files written to /data/loads/dbsnp/dbsnpload/output and executed against the snp back-end database.
   - DP_SNP_Marker.bcp
   - SNP_Accession.bcp
   - SNP_ConsensusSnp.bcp
   - SNP_ConsensusSnp_StrainAllele.bcp
   - SNP_Coord_Cache.bcp
   - SNP_Flank.bcp
   - SNP_Strain.bcp
   - SNP_SubSnp.bcp
   - SNP_SubSnp_StrainAllele.bcp
2. Inline update of snp..MGI_dbinfo.modification_date and snp..MGI_Tables _LoadedBy_key, loaded_date, and modification_date.
3. log files/data/loads/dbsnp/dbsnpload/logs/dbsnpload.*.log.
   - dbsnpload.diag.log - Diagnostic log. Logs progress of each task, processing by by chromosome, bcp and indexing etc. Statistics are reported at the end:
     - Statistics for all snps:
       - Total RefSnps looked at
       - Total SubSnps for the RefSnps looked at
       - Total RefSnp records repeated in the input

- Total RefSnps with no defined strain/alleles for any assay of the SNP
- Total RefSnps mapped to more than 1 chromosome in the C57BL/6J genome
- Total RefSnps mapped to > 2 coordinates on the same chromosome
- Total RefSnps unmapped in the C57BL/6J genome
- Total RefSnps with vocab resolving errors
- Total RefSnps created
- Total SubSnps created
- Total SS loaded with no strain alleles
- Total dbSNP snp/marker relationships created
- Total snp coordinates created
- List of new variation classes
- List of new allele characters
- List of new submitter handles
- List of new strains
- List of new function classes
- List of RefSnps whose allele summary is too long.

- List of subset of above statistics by chromosome

- dbsnpload.proc.log - Process Summary log. Reports success or failure and a subset of the load statistics. If failure, reports which process failed, e.g. 'dbsnp subHandle vocabulary load Failed'. This is the log included in the load email.
- dbsnpload.cur.log logs all discrepancies. This is used primary for debugging/testing. Totals of each discrepancy are logged in the diagnostic log.
  1. RS ID of all RefSnps with no strain/alleles. Tagged as ""RS NO STRAIN/ALLELES for RS*rsid*'.
  2. RS ID and SS ID for all SS with no strain alleles. Tagged as "SS NO STRAIN/ALLELES for RS*rsid* SS*ssid*".
  3. Variation class and RS ID of all unresolved variation classes. Tagged as 'UNRESOLVED CS VARCLASS' or 'UNRESOLVED SS VARCLASS'. Send these to Richard to determine MGI value; add to variation class vocabulary input file and translationload input file. See Section 8.1.2, "Create Vocabularies" on page 10 and .Section 8.1.3, "Create Translations" on page 11
  4. RS ID of RS rejected because no ConsensusSnp Allele Summary. Tagged as 'NO ALLELE SUMMARY for RS*rsid*'.This happens when no strains resolve. (In Build 125, MGI3.44, this also happened when there were only 'N' alleles).

5. RS has a Map Location with missing start coordinate. Tagged as "MISSING BL6 STARTCOORD for *RSrsid*.
6. Allele character and SS ID of SS that has an unrecognized allele character. i.e. other than ‘’, ‘N’, ‘A’, ‘C’, ‘G’, or ‘T’. Tagged as ‘UNRECOGNIZED ALLELE CHARACTER for SS*ssid*. (Note that we don’t load ‘’ and ‘N’ alleles). Talk to Richard to determine how to interpret the new character.
7. Submitter handle and RS ID of all RS with unresolved submitter handles. Tagged as ‘UNRESOLVED SUBMITTER HANDLE’. Need to add the new handle to the submitter vocabulary input file. See Section 8.1.2, “Create Vocabularies” on page 10.
8. Strain name, SS ID and RS ID for all SS with unresolved strains. Tagged as ‘UNRESOLVED STRAIN’. Send these to a strain curator (currently Beverly Richards-Smith) and cc Richard. She will handle creating new strains and/or creating a translations for new strains.
9. Function class and RS ID for all RS with an unresolved function class. Tagged as ‘UNRESOLVED FXNCLASS. Send these to Richard to determine MGI value. He will create DAG input file for fxn class vocabulary load, add new classes to function class translationload input file. See Section 8.1.2, “Create Vocabularies” on page 10 and Section 8.1.3, “Create Translations” on page 11
10. RefSnps with no Build 36 coordinate. Tagged as "OLD BUILD COORDINATE ONLY for RS*rsid*.

Troubleshooting:

1. Process Summary Log reports: “Could not start a new job stream for this load”. This means a jobstream for the load is registered as running in the configured radar database APP_Jobstream table. This happens when:
   - Another instance of the load is running.
   - The load failed without doing cleanup or you killed the load. Delete the record from APP_Jobstream and start again.
2. Memory Exception. This is a runtime java error that will be reported to stderr. If you run the load from cron you will see it in the cron email, else in your terminal window. In dbsnpload.config set LOG_DEBUG=true, this logs available memory to diagnostic log after each snp is processed.

8.1.6 Run snpcacheload

See Section 8.2, “snpcacheload” on page 16.

8.1.7 Run Strain Order Updates

Product/Script: dbsnpload/bin/updateSnpStrainOrder.sh.

This script orders snp strains by in-line updates to SNP_Strain.sequenceNum.

Inputs:

1. dbsnpload/data/mgiSnpStrainOrder.txt - a hand curated file (Richard) with all snp strains listed in the order they should appear in the WI. There are two tiers with strains alphabetized within each. As a rule new strains are added to tier 2.
2. SNP_Strain

Outputs:

1. inline updates to SNP_Strain.sequenceNum
2. log files: /data/loads/dbsnp/dbsnpload/logs/updateSnpStrainOrder.log.

8.1.8 Run postProcessing

Product/Script: dbsnpload/bin/postProcess.sh

This script dumps the back-end snp database, loads the front-end snp database, and updates the production mgd database MGI_dbinfo table.

Inputs:

1. back-end snp database.

Outputs:

1. backup file from dumping the back-end snp database.
2. loads front-end snp database from the backup file. The load takes the database out of single user mode when it is done.
3. updates mgd database MGI_dbinfo table.
4. log files: /data/loads/dbsnp/dbsnpload/logs/postProcess.log.

8.2 snpcacheload

Description: This product creates SNP marker associations in the back-end snp database. It can be run in two modes.Out-of-sync mode loads only dbSNP MGI to marker associations. In-sync mode loads dbSNP and MGI derived snp to marker associations. It also manages loading from the back-end snp database to the front-end snp database. There are two jobstream scripts 1) snpmarker.sh and 2) snpmarker_weekly.csh

Configuration: See snpcacheload/Configuration.

Main Jobstream Script: snpcacheload/snpmarker.sh.

This script runs from dbsnpload jobstream and creates dbSNP to MGI marker associations in the snp back-end database using dbSNP RefSNP EntrezGene ids. In addition, if configured to do so (IN_SYNC=true), creates additional MGI to marker associations based on distance of the snp from the marker (See snpmrkwithin.sh below).

Inputs:

1. mgd..ACC_Accession - query for EntrezGene ID to MGI Marker associations.

2. snp..DP_SNP_Marker - raw data from SNP with which to resolve and create SNP_ConsensusSnp_Marker objects.
3. snp..SNP_Accession - to resolve RefSnp IDs from snp..DP_SNP_Marker to _ConsensusSnp_Keys.
4. snp..SNP_Coord_Cache - get the _Coord_Cache_key for each given _ConsensusSnp_key, chromosome, start coordinate triple.

Outputs:

1. The following bcp files written to /data/loads/mgi/snpcacheload/output and executed against the snp back-end database.
   - SNP_ConsensusSnp_Marker.bcp.
   - SNP_Accession.bcp (associates SNP_ConsensusSnp_Marker objects with Ref-Seq IDs).
2. log file: /data/loads/mgi/snpcacheload/logs/snpmarker.sh.log.
3. archives and/or cleans logs and/or output directories if specified on the command line.

In-sync Script: snpcacheload/snpmrkwithin.sh - run from snpmarker.sh when IN_SYNC=true.

Create new MGI marker associations based on distance of the snp from the marker. Also reassigns dbSNP 'locus-region' function class to 'upstream' or 'downstream' counterpart where applicable. For Build 126 we are storing only locus-regions upstream/downstream associatins and 'within coordinates of' associations. The WI is calculating the distance relationships ('within 2 KB of' etc) on the fly.

This script can, but is not meant to, run by itself as we would always want to update the dbSNP snp to marker associations. (i.e. pick up new markers after the EntrezGene load is run) before doing this step.

Inputs:

1. mgd..MRK_Location_Cache - to load snp..MRK_Location_Cache.
2. mgd.VOC_Term - to create function class lookup.
3. snp..SNP_Coord_Cache - process snps by chromosome and by coordinate region within that chromosome.
4. snp..MRK_Location_Cache - create marker lookups for given regions.
5. SNP_ConsensusSnp_Marker - create lookup of dbSNP snp to marker associations (we don't want to create duplicates).

Outputs:

1. The following snp database bcp files written to /data/loads/mgi/snpcacheload/output and executed against the snp back-end database.
   - MRK_Location_Cache.bcp

- SNP_ConsensusSnp_Marker_Within.bcp1 (*.bcp2..*.bcpN as needed). Note that we split this bcp files into separate files <= 2Gb of data because, at the time, bcp would not recognize large files. Since we aren't loading distance relationships this bcp file would never exceed 2Gb.
- tempdb..TMP_SNP_Marker_Fxn.bcp - work table for updating dbSNP snp to marker associations with 'locus-region' function class to their 'upstream' or 'downstream' counterparts where applicable.

2. log file: /data/loads/mgi/snpcacheload/logs/snpmarker.sh.log.

Weekly Jobstream Script: snpcacheload/snpmarker_weekly.sh

This script is a wrapper that runs snpmarker.sh (which in turn runs snpmrkwithin.sh if configured to do so). It then dumps the back-end snp database and loads the front-end snp database. This script runs weekly (Sunday night/monday morning from mgidbutilities/bin/prod/weeklytasks.csh) after the EntrezGene load (which assigns eg IDs to MGI Markers). It is run asynchronously (invoked with &) so that weeklytasks.csh may continue. It needs to have exclusive access to load the front end snp database, so is run during non-curation hours.

Inputs:

1. See Inputs for snpmarker.sh and snpmrkwithin.sh.

Outputs:

1. See Outputs for snpmarker.sh and snpmrkwithin.sh.
2. Back-end snp database dump file: shire:/extra1/sybase/snp.backup*
3. Archives and/or cleans logs and/or output directory if specified on the command line.

9 Job Scheduling

See Weekly SNP Loads table in http://prodwww.informatics.jax.org/software/system_docs/jobschedule.pdf

10 How to run dbsnpload

With each new dbSNP Build we run dbsnpload in development to see if there are any parsing errors (indicating file format changes), and see what kicks out to the curator log (new vocab terms, strains, etc).

1. Run dbsnpload.sh without vocabulary options.
2. Look at statistics at the end of the diagnostic logs carefully. e.g typically only very few snps have multiple locations on a chromosome. I noticed that the coordinates loaded was about double that of the snps which was fishy. Turns out the Build 126 files contain coordinates for both Mouse Assembly build 35 and 36. Once I adjusted the parser

and processing for that, I found there were no markers loaded. Turns out the dbSNP database dump to XML was bad and had to be redone.

3. Have Beverly add new strains to the strain translation
4. Add any new submitter handles, variation classes or function classes to their respective vocabulary input files
5. If new variation or function classes also add to their translation input files
6. If run out of space on a database segment, have Mike M. add a device.
7. If new strains add strain to tier 2 of the strain order input file
8. If an attribute exceeds the size of the database column, update the schema product. This has happened with the ConsensusSnp allele summary such that we increased it to varchar (200).
9. Run dbsnpload, if you are lucky, you'll be done. :-)