Babase Chado:

Chado and it’s Integration with the Babase System
Copyright Notices


Except as otherwise noted permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”

Chado is distributed under the Artistic License 2.0.
The holders of the copyright to Chado are not readily determined so it is not possible to include a copyright notice here.
Some portions of this document are copied verbatim from the Chado documentation. See the document source code for specifics.
# COLLABORATORS

<table>
<thead>
<tr>
<th>TITLE</th>
<th>ACTION</th>
<th>NAME</th>
<th>DATE</th>
<th>SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babase Chado:</td>
<td>WRITTEN</td>
<td>Karl O. Pinc</td>
<td>January 18, 2019</td>
<td></td>
</tr>
</tbody>
</table>

# REVISION HISTORY

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DATE</th>
<th>DESCRIPTION</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>June 29, 2015</td>
<td>Beta version</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>May 22, 2014</td>
<td>Initial document</td>
<td></td>
</tr>
</tbody>
</table>
# Contents

1 Introduction 1
   1.1 This Document .................................................. 1
   1.2 The Chado Approach .............................................. 1
   1.3 Other Resources .................................................. 2

2 Some Chado Documentation 3
   2.1 Table Overview .................................................. 3
   2.2 Entity-Relationship Diagrams .................................... 3

3 The Tables 16
   3.1 Chado Tables .................................................... 16
      3.1.1 ANALYSISFEATURE (Analysis Feature Relationships) .... 16
         3.1.1.1 Analysisfeature_Id ................................... 17
         3.1.1.2 Feature_Id ........................................... 17
         3.1.1.3 Analysis_Id ........................................... 17
         3.1.1.4 Rawscore (Raw Score) ................................ 17
         3.1.1.5 Normscore (Normalized Score) ......................... 17
         3.1.1.6 Significance ......................................... 17
         3.1.1.7 Identity ............................................. 17
      3.1.2 ANALYSISFEATUREPROP (Analysis to Genomic Feature Relationship Proprieties) .... 18
         3.1.2.1 Analysisfeatureprop_Id ................................ 18
         3.1.2.2 Analysisfeature_Id ................................... 18
         3.1.2.3 Type_Id (Type) ....................................... 18
         3.1.2.4 Value ................................................ 18
         3.1.2.5 Rank .................................................. 18
      3.1.3 ANALYSISPROP (Analysis Properties) ........................ 18
         3.1.3.1 Analysisprop_Id ...................................... 18
         3.1.3.2 Analysis_Id ........................................... 18
         3.1.3.3 Type_Id (Type) ....................................... 19
         3.1.3.4 Value ................................................ 19
         3.1.3.5 Rank .................................................. 19
3.1.4 CV (Controlled Vocabularies) .................................................. 19
  3.1.4.1 CV_Id (Controlled Vocabulary Identifier) ............................... 19
  3.1.4.2 Name .............................................................................. 19
  3.1.4.3 Definition ......................................................................... 19

3.1.5 CVTERM (Controlled Vocabulary Terms) ...................................... 20
  3.1.5.1 CVTerm_Id (Controlled Vocabulary Term Identifier) ............... 20
  3.1.5.2 CV_Id (Controlled Vocabulary Identifier) .............................. 20
  3.1.5.3 Name .............................................................................. 20
  3.1.5.4 Definition ......................................................................... 20
  3.1.5.5 DBXref_Id (External Database Cross-Reference Identifier) .... 20
  3.1.5.6 IsObsolete ........................................................................ 20
  3.1.5.7 IsRelationshiptype ............................................................ 20

3.1.6 CVTERM_DBXREF (Controlled Vocabulary Term Ancillary External Database References) ................. 21
  3.1.6.1 CVTerm_DBXref_Id (Controlled Vocabulary Term External Database Cross-Reference Identifier) .... 21
  3.1.6.2 CVTerm_Id (Controlled Vocabulary Term Identifier) .............. 21
  3.1.6.3 DBXref_Id (External Database Cross-Reference Identifier) .... 21

3.1.7 CVTERM_RELATIONSHIP (Controlled Vocabulary Term Relationships) .................................................. 21
  3.1.7.1 CVTerm_Relationship_Id (Controlled Vocabulary Term Relationship Identifier) ....................... 21
  3.1.7.2 Type_Id (Type) .................................................................. 21
  3.1.7.3 Subject_Id ........................................................................ 21
  3.1.7.4 ObjectId .......................................................................... 22

3.1.8 CVTERMPATH (Controlled Vocabulary Term relationship Paths) ......................................................... 22
  3.1.8.1 CVTermpath_Id (Controlled Vocabulary Term relationship Path Identifier) ......................... 22
  3.1.8.2 Type_Id (Type) .................................................................. 22
  3.1.8.3 Subject_Id ........................................................................ 22
  3.1.8.4 ObjectId .......................................................................... 22
  3.1.8.5 CV_Id (Controlled Vocabulary Identifier) .............................. 22
  3.1.8.6 Pathdistance ..................................................................... 22

3.1.9 CVTERMPROP (Controlled Vocabulary Term Properties) ................................................................. 23
  3.1.9.1 CVTermprop_Id (Controlled Vocabulary Term Property Identifier) ........................................ 23
  3.1.9.2 CVTerm_Id (Controlled Vocabulary Term Identifier) ................ 23
  3.1.9.3 Type_Id (Type) .................................................................. 23
  3.1.9.4 Value .............................................................................. 23
  3.1.9.5 Rank ................................................................................ 23

3.1.10 DB (External Databases) ............................................................ 23
  3.1.10.1 DB_Id (Database Identifier) ................................................ 23
  3.1.10.2 Name .............................................................................. 23
  3.1.10.3 Description ...................................................................... 24
3.1.10.4 URLPrefix (Prefix Used for Web Queries of the External Database) ........................................ 24
3.1.10.5 URL (Web Address) .............................................................................................................. 24
3.1.11 DBXREF (External Database Object Cross-References) ........................................................... 24
  3.1.11.1 DBXref_Id (External Database Object Cross-Reference Identifier) ........................................ 24
  3.1.11.2 DB_Id (External Database Identifier) ..................................................................................... 24
  3.1.11.3 Accession ............................................................................................................................ 24
  3.1.11.4 Version ............................................................................................................................... 24
  3.1.11.5 Description ......................................................................................................................... 25
3.1.12 FEATURE (Genomic Feature) .................................................................................................... 25
  3.1.12.1 Feature_Id (Feature Identifier) ............................................................................................ 25
  3.1.12.2 DBXref_Id (External Database Object Identifier) ................................................................. 25
  3.1.12.3 Organism_Id (Organism Identifier) ..................................................................................... 25
  3.1.12.4 Name .................................................................................................................................. 25
  3.1.12.5 Uniquename ....................................................................................................................... 25
  3.1.12.6 Residues ............................................................................................................................. 26
  3.1.12.7 SeqLen (Sequence Length) ............................................................................................... 26
  3.1.12.8 MD5Checksum (Residue MD5 Hash) .................................................................................. 26
  3.1.12.9 Type_Id ............................................................................................................................... 26
  3.1.12.10 Is_Analysis (Feature Is Determined By Analysis) .............................................................. 26
  3.1.12.11 Is_Obsolete ...................................................................................................................... 26
  3.1.12.12 Timeaccessioned (Date and Time Feature Was Recorded in the Database) ......................... 26
  3.1.12.13 Timelastmodified (Date and Time Database Row Was Last Modified) .............................. 26
3.1.13 FEATURE_CVTERM (Feature Typing/Tagging) ......................................................................... 27
  3.1.13.1 Feature_CVTerm_Id ............................................................................................................ 27
  3.1.13.2 Feature_Id .......................................................................................................................... 27
  3.1.13.3 CVTerm_Id (Controlled Vocabulary Term Identifier) ......................................................... 27
  3.1.13.4 Pub_Id (Publication Identifier) ............................................................................................ 27
  3.1.13.5 Is_Not ................................................................................................................................ 27
  3.1.13.6 Rank .................................................................................................................................. 27
3.1.14 FEATURE_DBXREF (Feature to External Database Object Cross-References) .......................... 27
  3.1.14.1 Feature_DBXref_Id ................................................................................................................ 28
  3.1.14.2 Feature_Id (Feature Identifier) ............................................................................................. 28
  3.1.14.3 DBXref_Id (External Database Object Identifier) ................................................................. 28
  3.1.14.4 Is_Current ........................................................................................................................... 28
3.1.15 FEATURE_GENOTYPE (Feature to Genotype Relationships) .................................................... 28
  3.1.15.1 Feature_Genotype_Id ........................................................................................................... 28
  3.1.15.2 Feature_Id .......................................................................................................................... 29
  3.1.15.3 Genotype_Id ........................................................................................................................ 29
  3.1.15.4 Chromosome_Id (Chromosome Identifier) .......................................................................... 29
3.1.15.5 Rank .................................................. 29
3.1.15.6 CGroup (Chromosomal Group) ......................... 29
3.1.15.7 CVTerm_Id (Controlled Vocabulary Term Identifier) .... 29
3.1.16 FEATURE_RELATIONSHIP (Feature Inter-Relationships) .... 30
  3.1.16.1 Feature_Relationship_Id .................................. 30
  3.1.16.2 Subject_Id ........................................... 30
  3.1.16.3 Object_Id ............................................ 30
  3.1.16.4 Type_Id ............................................. 30
  3.1.16.5 Value (Additional Notes) ................................ 30
  3.1.16.6 Rank ................................................. 30
3.1.17 FEATURE_SYNONYM (Genomic Feature to Synonym Relationships) .... 31
  3.1.17.1 Feature_Synonym_Id (Feature to Synonym Relationship Identifier) .... 31
  3.1.17.2 Synonym_Id (Synonym Identifier) ....................... 31
  3.1.17.3 Feature_Id (Feature Identifier) ......................... 31
  3.1.17.4 Pub_Id (Publication Identifier) ......................... 31
  3.1.17.5 Is_Current ........................................... 31
  3.1.17.6 Is_Internal .......................................... 31
3.1.18 FEATURELOC (Feature Locations) ........................ 31
  3.1.18.1 Featureloc_Id (Feature Location Identifier) .............. 32
  3.1.18.2 Feature_Id (Feature Identifier) ......................... 32
  3.1.18.3 Srcfeature_Id (Reference Feature Identifier) ............. 32
  3.1.18.4 FMin (Feature Lower Boundary Coordinate) ............... 33
  3.1.18.5 Is_FMin_Partial (Is the Feature Lower Boundary Coordinate Only Partially Known?) .... 33
  3.1.18.6 FMax (Feature Upper Boundary Coordinate) ............... 33
  3.1.18.7 Is_FMax_Partial (Is the Feature Upper Boundary Coordinate Only Partially Known?) .... 33
  3.1.18.8 Strand (Strand Directionality) ......................... 33
  3.1.18.9 Phase (Translation Phase) ............................ 33
  3.1.18.10 Residue_Info (Alternative Residue Information) ........ 34
  3.1.18.11 Locgroup (Location Group) .......................... 34
  3.1.18.12 Rank ............................................. 34
3.1.19 FEATUREPROP (Feature Properties) .......................... 35
  3.1.19.1 Featureprop_Id ...................................... 35
  3.1.19.2 Feature_Id ......................................... 35
  3.1.19.3 Type_Id ........................................... 35
  3.1.19.4 Value ............................................... 35
  3.1.19.5 Rank ............................................. 35
3.1.20 GENOTYPE (Genotypes) ........................................ 35
  3.1.20.1 Genotype_Id (Genotype Identifier) ....................... 36
  3.1.20.2 Name ............................................... 36
3.1.20.3 Uniquename ................................................................. 36
3.1.20.4 Description ............................................................... 36
3.1.20.5 Type_Id ................................................................. 36

3.1.21 ND_EXPERIMENT (Natural Diversity Experiment Result Set) .................................................. 36
3.1.21.1 ND_Experiment_Id (Natural Diversity Experiment Result Set Identifier) ....................................... 37
3.1.21.2 ND_Geolocation_Id (Natural Diversity Geolocation Identifier) ..................................................... 37
3.1.21.3 Type_Id ................................................................. 37

3.1.22 ND_EXPERIMENT_GENOTYPE (Natural Diversity Experiment Result Set to Genotype Relationships) .................................................. 37
3.1.22.1 ND_Experiment_Genotype_Id ........................................ 37
3.1.22.2 Genotype_Id .......................................................... 37
3.1.22.3 ND_Experiment_Id ..................................................... 37

3.1.23 ND_EXPERIMENT_STOCK (Natural Diversity Experiment Result Set to Stock Relationships) .......................................................... 38
3.1.23.1 ND_Experiment_Stock_Id ........................................... 38
3.1.23.2 Stock_Id ............................................................... 38
3.1.23.3 ND_Experiment_Id ..................................................... 38
3.1.23.4 Type_Id ................................................................. 39

3.1.24 ND_EXPERIMENT_STOCKPROP (Natural Diversity Experiment Result Set to Stock Relationship Properties) .......................................................... 39
3.1.24.1 ND_Experiment_Stockprop_Id ..................................... 39
3.1.24.2 ND_Experiment_Stock_Id ........................................... 39
3.1.24.3 Type_Id ................................................................. 39
3.1.24.4 Value ................................................................. 39
3.1.24.5 Rank ................................................................. 39

3.1.25 ND_EXPERIMENTPROP (Natural Diversity Experiment Result Set Properties) .................................................. 40
3.1.25.1 ND_Experimentprop_Id ............................................. 40
3.1.25.2 ND_Experiment_Id ..................................................... 40
3.1.25.3 Type_Id ................................................................. 40
3.1.25.4 Value ................................................................. 40
3.1.25.5 Rank ................................................................. 40

3.1.26 ND_GEOLOCATION (Natural Diversity Geolocation) .......................................................... 40
3.1.26.1 ND_Geolocation_Id (Natural Diversity Geolocation Identifier) ..................................................... 40
3.1.26.2 Description .......................................................... 40
3.1.26.3 Latitude ............................................................. 41
3.1.26.4 Longitude .......................................................... 41
3.1.26.5 Geodetic ............................................................ 41
3.1.26.6 Altitude ............................................................. 41

3.1.27 ORGANISM .............................................................. 41
3.1.27.1 Organism_Id (Organism Identifier) ................................... 41
3.1.27.2 Abbreviation ......................................................... 41
3.1.27.3 Genus ................................................................. 41
3.1.27.4 Species ............................................................ 41
3.1.27.5 Common_Name ................................................... 41
3.1.27.6 Comment .......................................................... 41
3.1.28 ORGANISM_DBXREF (Organism to External Database Object Cross-References) ............... 42
3.1.28.1 Organism_DBXref_Id ............................................ 42
3.1.28.2 Organism_Id ..................................................... 42
3.1.28.3 DBXref_Id (External Database Object Identifier) ......................................................... 42
3.1.29 PUB (Publication or Other Documentation of Provenance) ................................................. 42
3.1.29.1 Pub_Id (Publication Identifier) .................................. 42
3.1.29.2 Title ................................................................. 42
3.1.29.3 Volumetitle (Title of Volume) .................................. 42
3.1.29.4 Volume ............................................................. 43
3.1.29.5 Series_Name ....................................................... 43
3.1.29.6 Issue ............................................................... 43
3.1.29.7 PYear (Publication Year) ......................................... 43
3.1.29.8 Pages ............................................................... 43
3.1.29.9 Miniref ............................................................. 43
3.1.29.10 Uniquename ...................................................... 43
3.1.29.11 Type_Id .......................................................... 43
3.1.29.12 IsObsolete ....................................................... 43
3.1.29.13 Publisher ........................................................ 43
3.1.29.14 Pubplace (Publication Place) .................................. 43
3.1.30 STOCK (Stocks) ...................................................... 44
3.1.30.1 Stock_Id (Stock Identifier) ....................................... 44
3.1.30.2 DBXref_Id (External Database Object Cross-Reference Identifier) ................................. 44
3.1.30.3 Organism_Id (Organism Identifier) .................................. 44
3.1.30.4 Name ............................................................... 44
3.1.30.5 Uniquename ....................................................... 44
3.1.30.6 Description ........................................................ 44
3.1.30.7 Type_Id (Type) .................................................... 44
3.1.30.8 IsObsolete ......................................................... 45
3.1.31 STOCK_DBXREF (Stock to External Database Object Cross-References) .......................... 45
3.1.31.1 Stock_DBXref_Id .................................................. 45
3.1.31.2 Stock_Id (Stock Identifier) ....................................... 45
3.1.31.3 DBXref_Id (External Database Object Identifier) ......................................................... 45
3.1.31.4 IsCurrent ......................................................... 45
3.1.32 STOCK_DBXREFPROP (Stock External Database Entity Properties) ................................. 45
3.1.32.1 Stock_DBxrefprop_Id ............................................. 45
3.1.32 Stock_DBXref_Id ................................................................. 46
3.1.32.3 Type_Id ................................................................. 46
3.1.32.4 Value ................................................................. 46
3.1.32.5 Rank ................................................................. 46

3.1.33 STOCK_CVTERM (Stock Typing/Tagging) ........................................ 46
3.1.33.1 Stock_CVTerm_Id ......................................................... 46
3.1.33.2 Stock_Id ................................................................. 46
3.1.33.3 CVTerm_Id (Controlled Vocabulary Term Identifier) ......................... 46
3.1.33.4 Pub_Id (Publication Identifier) .................................................. 46
3.1.33.5 Is_Not ................................................................. 47
3.1.33.6 Rank ................................................................. 47

3.1.34 STOCK_RELATIONSHIP (Stock Inter-Relationships) ......................... 47
3.1.34.1 Stock_Relationship_Id .................................................... 47
3.1.34.2 Subject_Id ................................................................. 47
3.1.34.3 Object_Id ................................................................. 47
3.1.34.4 Type_Id ................................................................. 47
3.1.34.5 Value (Additional Notes) .................................................. 47
3.1.34.6 Rank ................................................................. 47

3.1.35 STOCKPROP (Stock Properties) .................................................... 48
3.1.35.1 Stockprop_Id ................................................................. 48
3.1.35.2 Stock_Id ................................................................. 48
3.1.35.3 Type_Id ................................................................. 48
3.1.35.4 Value ................................................................. 48
3.1.35.5 Rank ................................................................. 48

3.1.36 SYNONYM (Alternate Names) .................................................. 48
3.1.36.1 Synonym_Id (Synonym Identifier) ............................................. 48
3.1.36.2 Name (Synonym) ......................................................... 49
3.1.36.3 Type_Id (Type) ......................................................... 49
3.1.36.4 Synonym_SGML ........................................................... 49

3.2 Babase Chado Extensions .............................................................. 49
3.2.1 ANALYSIS (Analyses) ............................................................ 49
3.2.1.1 Analysis_Id ................................................................. 49
3.2.1.2 Name ................................................................. 49
3.2.1.3 Description .............................................................. 50
3.2.1.4 Program ................................................................. 50
3.2.1.5 Programversion .......................................................... 50
3.2.1.6 Algorithm ............................................................... 50
3.2.1.7 Sourcename .............................................................. 50
3.2.1.8 Sourceversion ........................................................... 50
3.2.1.9 SourceURI ............................................. 50
3.2.1.10 Timeexecuted ....................................... 50
3.2.1.11 Term_Id ............................................. 51

3.2.2 ANALYSIS_DBXREF (Analysis to External Database Object Cross-References) .. 51
  3.2.2.1 Analysis_DBXref_Id ................................ 51
  3.2.2.2 Analysis_Id ......................................... 51
  3.2.2.3 DBXref_Id (External Database Object Identifier) ......................... 51
  3.2.2.4 Is_Current .......................................... 51

3.2.3 ANALYSIS_CVTERM (Analysis Typing/Tagging) ......................................... 51
  3.2.3.1 Analysis_CVTerm_Id .................................. 52
  3.2.3.2 Analysis_Id ........................................... 52
  3.2.3.3 CVTerm_Id (Controlled Vocabulary Term Identifier) ..................... 52
  3.2.3.4 Is_Not ................................................. 52

3.2.4 ANALYSIS_RELATIONSHIP (Analysis Inter-Relationships) ......................... 52
  3.2.4.1 Analysis_Relationship_Id .................................. 52
  3.2.4.2 Subject_Id ............................................ 52
  3.2.4.3 Object_Id ............................................. 52
  3.2.4.4 Type_Id ............................................... 52
  3.2.4.5 Value (Additional Notes) ................................ 52
  3.2.4.6 Rank .................................................. 53

3.2.5 ANALYSIS_ND_EXPERIMENT (Analysis to Natural Diversity Experiment Result Set Relationships) 53
  3.2.5.1 Analysis_ND_Experiment_Id .................................. 53
  3.2.5.2 Analysis_Id ........................................... 53
  3.2.5.3 ND_Experiment_Id ....................................... 53

3.2.6 ND_EXPERIMENT_FEATURE (Natural Diversity Experiment Result Set to Feature Relationships) .. 54
  3.2.6.1 ND_Experiment_Feature_Id .................................. 54
  3.2.6.2 Feature_Id ............................................. 54
  3.2.6.3 ND_Experiment_Id ....................................... 54

4 Chado Programs .................................................. 55
  4.1 Command Line Programs ...................................... 55
    4.1.1 abrp-genomics-update-testdir .................................. 55
    4.1.2 chado-funcgen-dump ...................................... 55
    4.1.3 chado-funcgen-load ........................................ 59
    4.1.4 chado-stock-load ......................................... 64
    4.1.5 chado-vcf-load ........................................... 69
    4.1.6 copy-to-abrp-genomics ..................................... 87
    4.1.7 receive-abrp-genomics-schema ................................ 88
4.2 Web Interfaces .......................................................... 89
  4.2.1 VCFDump (Download anonymized VCF files) ................. 89
    4.2.1.1 Description ................................................. 89
    4.2.1.2 Configuration .............................................. 89
  4.3 SQL Utilities ...................................................... 89
  4.3.1 chado_analysis_delete ........................................ 89

A Indexes Added To Chado .............................................. 92
  A.1 Indexes On ANALYSIS .............................................. 92
    A.1.1 analysis_idx1b on ANALYSIS.Type_Id ....................... 92
    A.1.2 analysis_bc1 on ANALYSIS.Name (Unique) .................. 92
  A.2 Indexes On FEATURE .............................................. 92
    A.2.1 feature_idx1b on FEATURE.Feature_Id, FEATURE.DBXref_Id . 92
  A.3 Indexes On FEATURE_RELATIONSHIP ............................ 92
    A.3.1 feature_relationship_idx1b on FEATURE_RELATIONSHIP Object_Id, Subject_Id, Type_Id 92
  A.4 Indexes On FEATURELOC .......................................... 92
    A.4.1 featureloc_idx1b on FEATURELOC Feature_Id, FMin, FMax . 92
## List of Figures

2.1 Key to the Chado Entity Relationship Diagrams ......................................................... 3
2.2 Chado Stock Module ER Diagram .............................................................................. 6
2.3 Chado Sequence Module Entity Relationship Diagram .............................................. 7
2.4 Chado Companalysis Module Entity Relationship Diagram ...................................... 8
2.5 Meta-Data Portions of The Chado Natural Diversity Module ER Diagram ................. 9
2.6 Stock Portions of The Chado Natural Diversity Module ER Diagram ...................... 10
2.7 Babase Extension Relating The Chado Natural Diversity And Analysis Modules ER Diagram .............................................................. 11
2.8 Protocol and Reagent Portions of The Chado Natural Diversity Module ER Diagram ... 12
2.9 Relationships Between The Chado Genetics Module, the Natural Diversity Module, And The Sequence module ER Diagram .................................................................................. 13
2.10 Genomic Data Tracking Entity Relationship Diagram ........................................... 14
2.11 Integrating Genomic Data Tracking With Chado ER Diagram ............................... 15
List of Tables

2.1 Some Chado Tables (Part I) ....................................................... 4
2.2 Some Chado Tables (Part II) ..................................................... 5
2.3 Some Chado Tables (Part III) ..................................................... 5
Acknowledgments

We gratefully acknowledge the support of the National Science Foundation for the supporting the collection of the majority of the data stored in the database; in the past decade in particular we acknowledge support from IBN 9985910, IBN 0322613, IBN 0322781, BCS 0323553, BCS 0323596, DEB 0846286, DEB 0846532 and DEB 0919200. We are also very grateful for support from the National Institute of Aging (R01AG034513-01 and P01AG031719) and the Princeton Center for the Demography of Aging (P30AG024361). We also thank the Chicago Zoological Society, the Max Planck Institute for Demographic Research, the L.S.B. Leakey Foundation and the National Geographic Society for support at various times over the years. In addition, we thank the National Institute of Aging (R03-AG045459-01) for supporting recent work extending the database to incorporate genetic and genomic data.

Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the National Institute of Aging, the Princeton Center for the Demography of Aging, the Chicago Zoological Society, the Max Planck Institute for Demographic Research, the L.S.B. Leakey Foundation, the National Geographic Society, or any other organization which has supplied support for this work.
Chapter 1

Introduction

1.1 This Document

This document describes portions of Chado and how it integrates with the Babase baboon data management system.

1.2 The Chado Approach

Chado is not your grandmother’s database. It is designed to be highly flexible and is very ontology focused. What this means is that where, in other databases, you might expect to see a table having a specific column, e.g., a “hair color” column in a “phenotype” table, you instead see a “hair color” row in a “phenotype property” table. The row substitutes for the “hardcoded” column. The row is identified as one which describes hair color by the presence of a “hair color” ontology id in a “type” column in the “phenotype property” table. (In Chado parlance the column name is often some variant of “cvterm” rather than being named “type”.)

A different phenotype property, say, eye color, would be identified as such by the use of an “eye color” ontology id in the “phenotype property” table. The eye color in question would be a value in a “value” column in the eye color row. Looking at this from another angle, the hair color and eye color and other phenotype properties all appear in the “value” column of the “phenotype property” table. Exactly what any particular value in any given row represents is dependent upon the ontology term used in the row.

This approach to database design allows some phenotypes to have information on hair color and others not, but more importantly it allows each phenotype to have, effectively, an arbitrary and varying number of “columns”. The meaning of these “virtual columns” are defined by the ontologies in use.

Note that Chado property tables all have Type_Id and Rank columns. The Type_Id value declares what type of value the column contains. In the phenotype example above one Type_Id value would indicate that the value is a hair color. Another Type_Id value would indicate that the value is an eye color. The Rank value denotes whether or not the virtual column contains multiple values per Type_Id and, if so, indicates the ordinal position of the value amongst the multiple values. A Rank value of 0 indicates that the virtual column does not have multiple values. Other, non-0 values, denote the value’s ordinal position in the the designated set of, in this case, per phenotype per Type_Id, values.

It is worth pointing out that database views can reorganize the presentation of the data such that these “virtual columns” because, in effect, actual columns.

For more information on this approach to database design see the Wikipedia entry on the Entity-Attribute-Value model of database design.

\footnote{A feature of PostgreSQL and other databases.}
1.3 Other Resources

The primary resource, through which other resources related to Babase may be found, is:

- The Babase Technical Specifications.
Chapter 2

Some Chado Documentation

2.1 Table Overview

2.2 Entity-Relationship Diagrams

The Chado Database Schema

Figure 2.1: Key to the Chado Entity Relationship Diagrams
<table>
<thead>
<tr>
<th>General Module</th>
<th>One row for each</th>
<th>Better Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>database, often an “external” database that’s curated by an outside entity.</td>
<td>DBS</td>
<td>Database are generally named after a database authority; the authority that issues or curates the database.</td>
</tr>
<tr>
<td>DBXREF</td>
<td>external database per external database object; i.e., per row or term in the external database.</td>
<td></td>
<td>The purpose is to be able to link back to specific rows in the source database.</td>
</tr>
</tbody>
</table>

| Stock Module   | physical entity held in collection; generally, one row per genetic input into an experiment. | STOCKS | |
| STOCK          | additional categorization of the stock. | STOCK_TYPES | |
| STOCK_DBXREF   | (external) database entry related to the stock. | | |

<table>
<thead>
<tr>
<th>Sequence Module</th>
<th>One row for each</th>
<th>Better Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE</td>
<td>biological sequence or a section of a biological sequence or collection of such sections.</td>
<td>FEATURES</td>
<td>Anything that can be sequenced is a feature.</td>
</tr>
<tr>
<td>FEATURE_CVTERM</td>
<td>annotation of the feature.</td>
<td>F_ANOTATIONS</td>
<td>Annotations are ontology based. There may be more than one per feature.</td>
</tr>
<tr>
<td>FEATURE_DBXREF</td>
<td>(external) database entry related to the feature.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEATURE_RELATIONSHIP</td>
<td>subject-verb-object relationship between 2 features.</td>
<td>F_RELATIONS</td>
<td>Ontologies provide the “verb” terms.</td>
</tr>
<tr>
<td>FEATURE_SYNONYM</td>
<td>feature having a synonym per synonym had by said feature</td>
<td>F_SYNONYM</td>
<td></td>
</tr>
<tr>
<td>FEATURELOC</td>
<td>feature or portion thereof which maps to a reference feature.</td>
<td>F_LOCS</td>
<td></td>
</tr>
<tr>
<td>FEATUREPROP</td>
<td>additional “property” related to the feature.</td>
<td>F_PROPS</td>
<td>Provides the FEATURES table with additional “virtual columns”.</td>
</tr>
<tr>
<td>SYNONYMS</td>
<td>synonym of a feature name.</td>
<td>SYNONYMS</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Some Chado Tables (Part I)
## Companalysis Module

<table>
<thead>
<tr>
<th>Table</th>
<th>One row for each</th>
<th>Better Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYSIS</td>
<td>computational analysis.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANALYSISFEATURE</td>
<td>analysis for each feature on which the analysis produces results.</td>
<td>A_RESULTS</td>
<td></td>
</tr>
<tr>
<td>ANALYSISFEATUREPROP</td>
<td>analysis per additional value produced by the analysis per feature regarding which the value is applicable.</td>
<td>AR_PROPS</td>
<td></td>
</tr>
<tr>
<td>ANALYSISPROP</td>
<td>analysis per additional value related to the analysis as a whole.</td>
<td>A_PROPS</td>
<td></td>
</tr>
</tbody>
</table>

## Natural Diversity Module

<table>
<thead>
<tr>
<th>Table</th>
<th>“Set” of experimental results</th>
<th>ND_E_RESULTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ND_EXPERIMENT</td>
<td>experimental protocol.</td>
<td>ND_PROTOCOLS</td>
<td></td>
</tr>
<tr>
<td>ND_REAGENT</td>
<td>reagent or regent+process used in experiments.</td>
<td>ND_REAGENTS</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Some Chado Tables (Part II)

## Babase Chado Extensions

<table>
<thead>
<tr>
<th>Table</th>
<th>pair of analysis and experiments.</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYSIS_ND_EXPERIMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANALIS_DBXREF</td>
<td>external database object related to the analysis.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANALYSIS_CVTERM</td>
<td>analysis per ancillary type or tag used to classify the analysis.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANALYSIS_RELATIONSHIP</td>
<td>subject/verb/object or parent/child relationship between 2 analyses.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Some Chado Tables (Part III)
Figure 2.2: Chado Stock Module ER Diagram
Figure 2.3: Chado Sequence Module Entity Relationship Diagram
Figure 2.4: Chado Companalysis Module Entity Relationship Diagram
Figure 2.5: Meta-Data Portions of The Chado Natural Diversity Module ER Diagram
Figure 2.6: Stock Portions of The Chado Natural Diversity Module ER Diagram
Figure 2.7: Babase Extension Relating The Chado Natural Diversity And Analysis Modules ER Diagram
Figure 2.8: Protocol and Reagent Portions of The Chado Natural Diversity Module ER Diagram
Figure 2.9: Relationships Between The Chado Genetics Module, the Natural Diversity Module, And The Sequence module ER Diagram
Figure 2.10: Genomic Data Tracking Entity Relationship Diagram
Figure 2.11: Integrating Genomic Data Tracking With Chado ER Diagram
Chapter 3

The Tables

This chapter documents both the genomics related Babase tables and the Chado tables used by Babase. Unless otherwise noted all columns may be NULL.

3.1 Chado Tables

Selected Chado tables that are used by Babase are documented here.

3.1.1 ANALYSISFEATURE (Analysis Feature Relationships)

The ANALYSISFEATURE table is intended to capture relationships between analyses and genomic features in those cases where the analyses discover, predict, or otherwise generate the features. (E.g. Genscan generates transcripts and exons; sim4 alignments generate similarity/match features.) ANALYSISFEATURE contains one row per every relationship between a genomic feature and a analysis -- that is, one row per pairing of FEATURE row with a ANALYSIS row. ANALYSISFEATURE is used to decorate these features with analysis specific attributes. A feature is an analysisfeature if and only if there is a corresponding entry in the ANALYSISFEATURE table.

Note that the 4 analysis metrics provided by the columns of ANALYSISFEATURE do not cover the full range of scores an analysis might possibly generate; it would be undesirable to have columns for every score possible, as this should be kept extensible. Instead, for non-standard scores, use the ANALYSISFEATUREPROP table for record keeping.

ANALYSISFEATURE relates analysis to genomic features in, potentially, a many-to-many fashion. The table contains one row for every pairing of analysis to feature.

Tip
Using ANALYSISFEATURE it is possible to construct many-to-many relationships between ANALYSIS rows and FEATURE rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable, at least not per analysis. It is probably more appropriate to, per analysis, construct one-to-many relationships between ANALYSIS and FEATURE rows, or the reverse, to construct many-to-one relationships between ANALYSIS and FEATURE rows. In some cases it may be appropriate to construct one-to-one relationships between ANALYSIS and FEATURE rows.

The use of ANALYSISFEATURE in Babase
When doing analyses in Babase results are typically produced per-individual, so results are typically stored in the ND_EXPERIMENT_STOCKPROP table. These per-individual results are then linked to features by way of the GENOTYPE table.
In some cases within Babase it may desirable to link analysis results to features directly. One example is SNV analysis. In this case the probability of variation at the SNV site is determined per analysis and it is appropriate to relate the analysis directly to the SNV feature.
When multiple join paths relate the ANALYSIS table to the FEATURE table care must be taken to ensure that the result makes sense. At the present time there are no data validation rules in place to ensure database integrity.

The combination of Feature_Id and Analysis_Id must be unique. ANALYSISFEATURE rows are automatically deleted whenever any related FEATURE row is deleted. ANALYSISFEATURE rows are automatically deleted whenever any related ANALYSIS row is deleted.

### 3.1.1.1 Analysisfeature_Id

A unique number identifying the pairing of ANALYSIS row to FEATURE row. This column may not be NULL.

### 3.1.1.2 Feature_Id

The genomic feature related to the analysis. A FEATURE.Feature_Id value. This column may not be NULL.

### 3.1.1.3 Analysis_Id

The analysis related to the genomic feature. An ANALYSIS.Analysis_Id value. This column may not be NULL.

### 3.1.1.4 Raw_score (Raw Score)

This is the native score generated by the program; for example, the bitscore generated by blast, sim4 or genscan scores. One should not assume that high is necessarily better than low.

### 3.1.1.5 Normscore (Normalized Score)

This is the rawscore but semi-normalized. Complete normalization to allow comparison of features generated by different programs would be nice but too difficult. Instead the normalization should strive to enforce the following semantics: normscores are floating point numbers $\geq 0$, high normscores are better than low one. For most programs, it would be sufficient to make the normscore the same as this rawscore, providing these semantics are satisfied.

### 3.1.1.6 Significance

This is some kind of expectation or probability metric, representing the probability that the analysis would appear randomly given the model. As such, any program or person querying this table can assume the following semantics: $0 \leq$ significance $\leq n$, where $n$ is a positive number, theoretically unbounded but unlikely to be more than 10; low numbers are better than high numbers.

### 3.1.1.7 Identity

Percent identity between the locations compared.

---

The use of ANALYSISFEATURE.Identity within Babase

This column is uninteresting within Babase.
3.1.2 ANALYSISFEATUREPROP (Analysis to Genomic Feature Relationship Proprieties)

The ANALYSISFEATUREPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the ANALYSISFEATURE table. ANALYSISFEATUREPROP contains one row for every (non-NULL) additional column value, per ANALYSISFEATURE row, per Rank value.

The combination of Analysisfeature_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per analysis feature pairing per value type.

ANALYSISFEATUREPROP rows are automatically deleted whenever any related ANALYSISFEATURE row is deleted. ANALYSISFEATUREPROP rows are automatically deleted whenever any related CVTERM row is deleted.

3.1.2.1 Analysisfeatureprop_Id
A unique number identifying the ANALYSISFEATUREPROP row. This column may not be NULL.

3.1.2.2 Analysisfeature_Id
The analysis feature pairing having the property -- the Analysisfeature_Id value of the related ANALYSISFEATURE row. This column may not be NULL.

3.1.2.3 Type_Id (Type)
The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.2.4 Value
The analysis feature pairing property value -- as text.

3.1.2.5 Rank
A number which gives the value an ordinal position among the other ANALYSISFEATUREPROP values, per the row’s Type_Id value, related to the ANALYSISFEATURE row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.3 ANALYSISPROP (Analysis Properties)

The ANALYSISPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the ANALYSIS table. ANALYSISPROP contains one row for every (non-NULL) additional column value, per ANALYSIS row, per Rank value.

The combination of Analysis_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per analysis per value type.

ANALYSISPROP rows are automatically deleted when the related ANALYSIS row is deleted. ANALYSISPROP rows are automatically deleted when the related CVTERM row is deleted.

3.1.3.1 Analysisprop_Id
A unique number identifying the ANALYSISPROP row. This column may not be NULL.

3.1.3.2 Analysis_Id
The analysis having the property -- the Analysis_Id value of the related ANALYSIS row. This column may not be NULL.
3.1.3.3  Type_Id (Type)

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.3.4  Value

The analysis property value -- as text.

3.1.3.5  Rank

A number which gives the value an ordinal position among the other ANALYSISPROP values, per the row’s Type_Id value, related to the ANALYSIS row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.4  CV (Controlled Vocabularies)

The CV table contains one row per controlled vocabulary set (i.e. per ontology). Ontologies may be locally defined or developed by some standard setting body. There are various names for sets of controlled vocabularies:

**NAMES FOR SETS OF CONTROLLED VOCABULARIES**

- Ontologies
- Terms
- Classes
- Types
- Universals
- Properties
- Controlled Vocabularies
- Support Tables
- Namespaces

The CV table encompasses all of these, having one row per each set of terms.

One or more of the rows in the CV table represents a set of terms some of which define relationships. These relationships are used by Chado to construct acyclic directed graphs describing relationships between Chado database objects. For instance, genomic features may be related to one another in various ways, RNA may be transcripts of DNA, genes may be composed of exons and introns, and so forth. Stocks may be derived from other stocks, or split into other stocks.

3.1.4.1  CV_Id (Controlled Vocabulary Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.4.2  Name

The name of the ontology. The Name value must be unique. This column may not be NULL.
3.1.4.3 Definition
A textual description of the criteria for membership in the ontology.

3.1.5 CVTERM (Controlled Vocabulary Terms)
The CVTERM table contains one row per controlled vocabulary term. Each term is assigned membership in a specific ontology by assignment of a CV_Id value.
The combination of Name, CV_Id, and IsObsolete must be unique.
A CVTERM row is automatically deleted when it’s related CV row is deleted. A CVTERM row is automatically deleted when it’s related DBXREF row is deleted.

3.1.5.1 CVTerm_Id (Controlled Vocabulary Term Identifier)
A unique number identifying the row. This column may not be NULL.

3.1.5.2 CV_Id (Controlled Vocabulary Identifier)
The controlled vocabulary to which the term belongs -- the CV_Id value of the related CV row. This column may not be NULL.

3.1.5.3 Name
A concise, human readable, label for the term suitable for identifying the term within the ontology. This column may not be NULL.

3.1.5.4 Definition
A human readable definition of the term.

3.1.5.5 DBXref_Id (External Database Cross-Reference Identifier)
The primary external database object represented by the row -- the related DBXREF.DBXref_Id value. This column may not be NULL. (See also the CVTERM_DBXREF table.)

3.1.5.6 IsObsolete
A number representing whether or not the term is obsolete. The number 0, the default, means the term is not obsolete. Any other value means the term is obsolete. This column may not be NULL.

Tip
Use the CVTERMPROP table to record when the term became obsolete.

3.1.5.7 IsRelationshiptype
A number representing whether or not the term can be used to declare relationships between Chado database objects. The number 0, the default, means the term may not be used to declare relationships. Any other value means the term may be used to declare relationships. This column may not be NULL.
Terms that may be used to define relationships are called Typedefs in the OBO format. They are also known as properties or slots.
3.1.6 CVTERM_DBXREF (Controlled Vocabulary Term Ancillary External Database References)

The CVTERM_DBXREF table contains one row per ancillary external database object related to the controlled vocabulary term. Multiple external database references can be related to a controlled vocabulary term, and vice-versa.

The combination of CVTerm_Id and DBXref_Id must be unique.

CVTERM_DBXREF rows are automatically deleted when the related CVTERM row is deleted. CVTERM_DBXREF rows are automatically deleted when the related DBXREF row is deleted.

3.1.6.1 CVTerm_DBXref_Id (Controlled Vocabulary Term External Database Cross-Reference Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.6.2 CVTerm_Id (Controlled Vocabulary Term Identifier)

The controlled vocabulary term with which the ancillary external database object is related -- the CVTerm_Id value of the related CVTERM row. This column may not be NULL.

3.1.6.3 DBXref_Id (External Database Cross-Reference Identifier)

The ancillary external database reference with which the controlled vocabulary term is related -- the DBXref_Id value of the related DBXREF row. This column may not be NULL.

3.1.7 CVTERM_RELATIONSHIP (Controlled Vocabulary Term Relationships)

CVTERM_RELATIONSHIP contains one row for every relationship between ontology terms in the CVTERM file.

Each CVTERM_RELATIONSHIP row represents a relationship linking two CVTERM rows. Each CVTERM_RELATIONSHIP row constitutes an edge in the graph defined by the collection of CVTERM rows and CVTERM_RELATIONSHIP rows. The meaning of the CVTERM_RELATIONSHIP row depends on the definition of the CVTERM row referred to by CVTERM_RELATIONSHIP.Type_Id.

However, in general the definitions are such that the statement “all Subject_Id predicate some Object_Id” is true. The CVTERM_RELATIONSHIP statement is about the subject, not the object. For example "insect wing part_of thorax".

For more on relationships between CVTERM rows see CVTERMPATH.

3.1.7.1 CVTerm_Relationship_Id (Controlled Vocabulary Term Relationship Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.7.2 Type_Id (Type)

The nature of the relationship between subject and object. A CVTERM.CVTerm_Id value. Relations are typically from the OBO relationship ontology although other relationship types are allowed.

3.1.7.3 Subject_Id

The subject of the subj-predicate-obj sentence. A CVTERM.CVTerm_Id value. The CVTERM_RELATIONSHIP row is about the subject. In a graph, this typically corresponds to the child node. This column may not be NULL.

\[1\] Type_Id
3.1.7.4 Object_Id

The object of the subject-relationship-object sentence. A CVTERM.CVTerm_Id value. In a graph, this typically corresponds to the parent node. This column may not be NULL.

3.1.8 CVTERMPATH (Controlled Vocabulary Term relationship Paths)

CVTERMPATH contains one row for every relationship chain between ontology terms in the CVTERM file.

The CVTERM file contains various ontologies. The vocabulary terms within these relationships may be related to one another in various ways. E.g. one of the possible relationships is the “is a” relationship; a chromosome is_a sequence. Further, these relationships can be chained and this chain used to reason about the data. E.g. a mitochondrial_chromosome is_a chromosome. So therefore a mitochondrial chromosome is a sequence.

CVTERMPATH contains the reflexive transitive closure of the CVTERM_RELATIONSHIP table -- a row for every link in the chain between related CVTERM rows. Regards the above example, there is a row in CVTERMPATH which says that a mitochondrial chromosome is a sequence.

3.1.8.1 CVTermpath_Id (Controlled Vocabulary Term relationship Path Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.8.2 Type_Id (Type)

The relationship type that the row is a closure over, a CVTERM.CVTerm_Id value. If NULL the row is a closure over all relationship types. If non-NULL the related CVTERM row should be a relationship term -- should have a non-0 Is_RelationshipType value.

When this value is non-NULL the closure applies to both the given relationship and the OBO:is_a (subclass) relationship.

3.1.8.3 Subject_Id

The subject of the subject-relationship-object sentence. A CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.8.4 Object_Id

The object of the subject-relationship-object sentence. A CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.8.5 CV_Id (Controlled Vocabulary Identifier)

Closures will mostly be within one CV. If the closure of a relationship traverses a CV, then this refers to the CV.CV_Id of the Object_Id. This column may not be NULL.

3.1.8.6 Pathdistance

The number of steps required to get from the subject CVTERM to the object CVTERM, counting from zero (reflexive relationship). This column may not be NULL.

---

Note

This number may be less than zero when counting backwards from the object to the subject.
3.1.9 CVTERMPROP (Controlled Vocabulary Term Properties)

The CVTERMPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the CVTERM table. CVTERMPROP contains one row for every (non-NULL) additional column value, per CVTERM row, per Rank value. A row corresponds to an AnnotationProperty in W3C OWL format.

The combination of CVTerm_Id, Type_Id, Rank, and Value must be unique. Note that this arrangement does not allow for duplicate values per controlled vocabulary term per type.

CVTERMPROP rows are automatically deleted when any related CVTERM row is deleted, whether the row is related by CVTerm_Id or Type_Id.

3.1.9.1 CVTermprop_Id (Controlled Vocabulary Term Property Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.9.2 CVTerm_Id (Controlled Vocabulary Term Identifier)

The controlled vocabulary term to which the property value belongs -- the CVTerm_Id value of the related CVTERM row. This column may not be NULL.

3.1.9.3 Type_Id (Type)

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.9.4 Value

The controlled vocabulary property value -- as text. This column may not be NULL. The default value is the empty string.

3.1.9.5 Rank

A number which gives the value an ordinal position among the other CVTERMPROP values, per the row’s Type_Id value, related to the CVTERM row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.10 DB (External Databases)

The DB table contains one row per database, often an “external” database that’s curated by an outside entity. Databases, rows in the DB table, are generally named after a database authority; the authority that issues or curates the database. Examples of external databases are ontologies, reference genomes, collections of files in a filesystem, and Babase itself. Pretty much any collection of data can be a database in this sense.

3.1.10.1 DB_Id (Database Identifier)

A unique number used to identify the external database within Chado. This column may not be NULL.

3.1.10.2 Name

The name of the external database. This is generally the name of the database authority that produces and curates the database. The Name value must be unique. This column may not be NULL.
3.1.10.3 Description

A textual description of the external database.

3.1.10.4 URLPrefix (Prefix Used for Web Queries of the External Database)

A URL to which a term in the designated database may be appended to produce a complete URL which queries the external database for the given term.

3.1.10.5 URL (Web Address)

The URL used to retrieve from the web information on the given database.

3.1.11 DBXREF (External Database Object Cross-References)

The DBXREF table contains one row per external database per database object; i.e., per row or term in the external database. The purpose is to be able to link data in Chado back to specific rows in the external database. Hence, while there need not be a row in DBXREF for every row in the external database there must at least be a row for every row Chado references in the external database.

Since the DB table can refer to locally curated databases, including the local Chado database, DBXREF rows may also refer to rows within Chado.

Rows within Chado are often representations of rows within external database. In this case the DBXref_Id linking the two is stored in a column of the Chado table and the DBXref_Id is said to be the primary identifier. Chado rows may also be related to multiple other, so called ancillary or secondary DBXref_Id identifiers, to support one-to-many and many-to-many relationships between rows within Chado and external database objects. In this case there is a table within Chado ending in _DBXREF to record these sorts of relationships between the Chado rows and the secondary external database objects.

Although the DBXref_Id is a number, when referring to DBXREF rows the usual written notation uses the DB.Name, DBXREF.Accession, and DBXREF.Version values written in the form Name:Accession or Name:Accession:Version.

The combination of DB_Id, Accession, and Version must be unique.

3.1.11.1 DBXref_Id (External Database Object Cross-Reference Identifier)

A unique number used internally within Chado to represent the external database object. This column may not be NULL.

3.1.11.2 DB_Id (External Database Identifier)

The id of the external database in which the external database object resides. A DB.DB_Id value. This column may not be NULL.

3.1.11.3 Accession

An identifier which the curator of the external database guarantees is unique among the objects in the database. (One of) the external database’s key(s) to the external database object. This column may not be NULL.

3.1.11.4 Version

The version number of the external database object.\(^2\) This column may not be NULL. The version defaults to the empty string.

\(^2\) The version should not really go on this table but should be in a separate DB_VERSION table that is linked to instead of linking to the DB table (and which, in turn, links to the DB table).
3.1.11.5 Description

A textual description of the external database object.

3.1.12 FEATURE (Genomic Feature)

The FEATURE table contains one row per biological sequence or a section of a biological sequence or collection of such sections. Anything that can be sequenced is a feature.

A feature is a biological sequence or a section of a biological sequence, or a collection of such sections. Examples include genes, exons, transcripts, regulatory regions, polypeptides, protein domains, chromosome sequences, sequence variations, cross-genome match regions such as hits and HSPs and so on; see the Sequence Ontology for more.

The use of FEATURE in Babase

Babase creates rows in FEATURE to represent each SNV site discovered. Note however that this is per SNV site, regardless of how many analyses were done of the site.

The combination of Organism_Id, Uniquename, and Type_Id is unique.

FEATURE rows are automatically deleted when the related DBXREF row is deleted. FEATURE rows are automatically deleted when the related ORGANISM row is deleted. FEATURE rows are automatically deleted when the related CVTERM row is deleted.

3.1.12.1 Feature_Id (Feature Identifier)

A unique number identifying the feature. This column may not be NULL.

3.1.12.2 DBXref_Id (External Database Object Identifier)

Identifier of the external database object the FEATURE row represents; the primary external database identifier. A DBXREF.DBXref_Id value.

3.1.12.3 Organism_Id (Organism Identifier)

Identifier of the ORGANISM row with which the feature is related. An ORGANISM.Organism_Id value. This column may not be NULL.

3.1.12.4 Name

A human readable name for the Feature.

3.1.12.5 Uniquename

The unique name for a feature; may not necessarily be particularly human-readable, although this is preferred.

Note

The Uniquename value is unique only per organism per type. See above.
3.1.12.6 Residues

A sequence of alphabetic characters representing biological residues (nucleic acids, amino acids). This column does not need to be manifested for all features; it is optional for features such as exons where the residues can be derived from the featureloc. It is recommended that the value for this column be manifested for features which may have non-contiguous sublocations (e.g. transcripts), since derivation at query time is non-trivial. For expressed sequence, the DNA sequence should be used rather than the RNA sequence. The default storage method for the residues column is EXTERNAL, which will store it uncompressed to make substring operations faster.

3.1.12.7 SeqLen (Sequence Length)

The length of the residue feature. See column: Residues. This column is partially redundant with the residues column, and also with featureloc. This column is required because the location may be unknown and the residue sequence may not be manifested, yet it may be desirable to store and query the length of the feature. The seqlen should always be manifested where the length of the sequence is known.

3.1.12.8 MD5Checksum (Residue MD5 Hash)

The 32-character checksum of the sequence, calculated using the MD5 algorithm. This is practically guaranteed to be unique for any feature. This column thus acts as a unique identifier on the mathematical sequence.

3.1.12.9 Type_Id

The kind of feature that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL. This will typically be a Sequence Ontology identifier.

3.1.12.10 Is_Analysis (Feature Is Determined By Analysis)

Boolean indicating whether this feature is annotated or the result of an automated analysis. Analysis results also use the companalysis module. Note that the dividing line between analysis and annotation may be fuzzy, this should be determined on a per-project basis in a consistent manner. One requirement is that there should only be one non-analysis version of each wild-type gene feature in a genome, whereas the same gene feature can be predicted multiple times in different analyses.

This column may not be NULL. The default value for this column is FALSE.

3.1.12.11 Is_Obsolete

Boolean indicating whether this feature has been obsoleted. Some chado instances may choose to simply remove the feature altogether, others may choose to keep an obsolete row in the table.

This column may not be NULL. The default value for this column is FALSE.

Consider recording the date the feature became obsolete in the FEATUREPROP table.

3.1.12.12 Timeaccessioned (Date and Time Feature Was Recorded in the Database)

The date and time the feature was first accessioned/determined/discovered. This is independent of entry into the database.

This column may not be NULL. The default value for this column is the current date and time. (The date and time of insert into the database.)

3.1.12.13 Timelastmodified (Date and Time Database Row Was Last Modified)

The date and time the feature was last accessioned/determined/discovered. This is independent of entry into the database.

This column may not be NULL. The default value for this column is the current date and time. (The date and time of insert into the database.)
3.1.13 FEATURE_CVTERM (Feature Typing/Tagging)

FEATURE_CVTERM contains one row per feature, per publication, per each ancillary type or tag used to classify the feature. As usual in Chado, the CVTERM table is used to supply the vocabulary (or vocabularies) used in classification.

The use of FEATURE_CVTERM within Babase
Babase is uninterested in publication. Every row has an identical Pub_Id value which may be safely ignored.

The combination of Feature_id, CVTerm_Id, Pub_Id, and Rank must be unique.
FEATURE_CVTERM rows are automatically deleted when the related FEATURE row is deleted. FEATURE_CVTERM rows are automatically deleted when the related PUB row is deleted. FEATURE_CVTERM rows are automatically deleted when the related CVTERM row is deleted.

3.1.13.1 Feature_CVTerm_Id
A unique number identifying the row. This column may not be NULL.

3.1.13.2 Feature_Id
The feature assigned an ancillary type or tag. An FEATURE.Feature_Id value. This column may not be NULL.

3.1.13.3 CVTerm_Id (Controlled Vocabulary Term Identifier)
The ancillary type or tag applied to the feature. A CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.13.4 Pub_Id (Publication Identifier)
The publication associated with the typing or tagging of the feature. A PUB.Pub_Id value. This column may not be NULL.

3.1.13.5 Is_Not
A Boolean value which serves to negate the type or tag (to negate the controlled vocabulary term). This column may not be NULL.

3.1.13.6 Rank
A number which gives the value an ordinal position among the other FEATURE_CVTERM values, per the row’s Pub_Id value, related to the FEATURE row. This column may not be NULL.
The default Rank value is 0.
For more on Rank values see The Chado Rank Column [1].

3.1.14 FEATURE_DBXREF (Feature to External Database Object Cross-References)

FEATURE_DBXREF contains one row per external database object related to an feature.

The use of FEATURE_DBXREF within Babase
FEATURE_DBXREF is used within Babase to supply chromosomes (and possibly other features) alternative identifiers. These alternative identifiers appear within VCF files and are used by chado-vcf-load.

The combination of Feature_Id and DBXref_Id must be unique.
FEATURE_DBXREF rows are automatically deleted when the related FEATURE row is deleted. FEATURE_DBXREF rows are automatically deleted when the related DBXREF row is deleted.
3.1.14.1 Feature_DBXref_Id

A unique number identifying the row. This column may not be NULL.

3.1.14.2 Feature_Id (Feature Identifier)

Identifier of the FEATURE row with which the external database object is related. An FEATURE.Feature_Id value. This column may not be NULL.

3.1.14.3 DBXref_Id (External Database Object Identifier)

Identifier of the external database object related to the FEATURE row. A DBXREF.DBXref_Id value. This column may not be NULL.

3.1.14.4 Is_Current

A Boolean. When TRUE the relationship between the feature and the external database object is in current use. When FALSE the relationship is is outdated. This column may not be NULL. The default value for this column is TRUE.

3.1.15 FEATURE_GENOTYPE (Feature to Genotype Relationships)

The FEATURE_GENOTYPE table contains one row per every relationship between a genomic feature and a genotype -- that is, one row per pairing of FEATURE row with a GENOTYPE row.

FEATURE_GENOTYPE relates genotype to genomic features in, potentially, a many-to-many fashion. The table contains one row for every pairing of genotype to feature.

Tip

Using FEATURE_GENOTYPE it is possible to construct many-to-many relationships between GENOTYPE rows and FEATURE rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable, at least not per analysis\(^{a}\). It is probably more appropriate to, per analysis, construct one-to-many relationships between GENOTYPE and FEATURE rows, or the reverse, to construct many-to-one relationships between GENOTYPE and FEATURE rows. In some cases it is appropriate to construct one-to-one relationships between GENOTYPE and FEATURE rows.

\(^{a}\) Where an analysis is an analysis of an experiment; in effect, where an analysis is an actual experiment.

The use of FEATURE_GENOTYPE within Babase

FEATURE_GENOTYPE is not used in Babase. It is replace with the Babase Chado extension table ND_EXPERIMENT_FEATURE.

The combination of Feature_Id, Genotype_Id, CVTerm_Id, Chromosome_Id, Rank, and Cgroup must be unique.

FEATURE_GENOTYPE rows are automatically deleted whenever any related FEATURE row is deleted. FEATURE_GENOTYPE rows are automatically deleted whenever any related GENOTYPE row is deleted. FEATURE_GENOTYPE rows are automatically deleted whenever any related CVTERM row is deleted.

The FEATURE_GENOTYPE.Chromosome_Id column is automatically set to NULL when the related FEATURE row is deleted.

3.1.15.1 Feature_Genotype_Id

A unique number identifying the pairing of GENOTYPE row to FEATURE row. This column may not be NULL.
3.1.15.2 Feature_Id

The genomic feature related to the genotype. A FEATURE.Feature_Id value. This column may not be NULL.

3.1.15.3 Genotype_Id

The genotype, typically the input material, related to the genomic feature. An GENOTYPE.Genotype_Id value. This column may not be NULL.

3.1.15.4 Chromosome_Id (Chromosome Identifier)

The chromosome on which the feature resides. A FEATURE.Feature_Id value.

⚠️ Caution

This should be a feature with an SO type of chromosome but this is not enforced. There is also no validation to ensure that the features resides on the referenced chromosome.

3.1.15.5 Rank

The Chado documentation states: “rank can be used for n-ploid organisms or to preserve order.”

The use of FEATURE_GENOTYPE.Rank within Babase

This column is not of interest in Babase.

This column may not be NULL. Although not allowed to be NULL this column has no default value.

For more on Rank values see The Chado Rank Column [1].

3.1.15.6 CGroup (Chromosomal Group)

The Chado documentation states: “Spatially distinguishable group. group can be used for distinguishing the chromosomal groups, for example (RNAi products and so on can be treated as different groups, as they do not fall on a particular chromosome).”

The use of FEATURE_GENOTYPE.CGroup within Babase

This column is not of interest in Babase.

This column may not be NULL. Although not allowed to be NULL this column has no default value.

3.1.15.7 CVTerm_Id (Controlled Vocabulary Term Identifier)

The type of relationship between the genetic feature and the genotype -- a CVTERM.CVTerm_Id value. This column may not be NULL.
3.1.16 FEATURE_RELATIONSHIP (Feature Inter-Relationships)

FEATURE_RELATIONSHIP contains one row per subject/verb/object or parent/child relationship between 2 features. This table allows organization of features into acyclic directed graphs to record things like which features are composed of which other features, which features are transcriptions of which other features, and so forth. It is one of the many “relationship” tables in Chado.

An example graph is “exon part_of transcript part_of gene”. If type is thought of as a verb, the each arc or edge makes a statement [Subject Verb Object]. The object can also be thought of as parent (containing feature), and subject as child (contained feature or subfeature). We include the relationship rank/order, because even though most of the time we can order things implicitly by sequence coordinates, we can not always do this - e.g. transspliced genes. It is also useful for quickly getting implicit introns.

The combination of Subject_Id, Object_Id, Type_Id, and Rank must be unique.

FEATURE_RELATIONSHIP rows are automatically deleted when either the related Subject_Id FEATURE row is deleted or when the related Object_Id FEATURE row is deleted. FEATURE_RELATIONSHIP rows are automatically deleted when the related CVTERM row is deleted.

3.1.16.1 Feature_Relationship_Id

A unique number identifying the row. This column may not be NULL.

3.1.16.2 Subject_Id

The subject of the subject-predicate-object sentence. A FEATURE.Feature_Id value. The subject is typically a sub-feature. This column may not be NULL.

3.1.16.3 Object_Id

The object of the subject-predicate-object sentence. A FEATURE.Feature_Id value. The object is typically the containing feature. This column may not be NULL.

3.1.16.4 Type_Id

The relationship between the subject feature and the object feature. A CVTERM.CVTerm_Id value that has a non-0 Is_Relationshiptype value. This column may not be NULL.

The Type_id is typically from the OBO relationship ontology, although other relationship types are allowed. The most common relationship type is OBO_REL:part_of. Valid relationship types are constrained by the Sequence Ontology.

3.1.16.5 Value (Additional Notes)

Additional textual notes on the relationship between the analyses.

3.1.16.6 Rank

A number which gives the value an ordinal position among the other FEATURE_RELATIONSHIP values, per Subject_Id/Object_Id pairing, related to the CVTERM row. This column may not be NULL.

The ordering of subject features with respect to the object feature may be important (for example, exon ordering on a transcript - not always derivable if you take transspliced genes into consideration). Rank is used to order these.

Rank is zero-based, counting starts from zero. The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].
3.1.17 FEATURE_SYNONYM (Genomic Feature to Synonym Relationships)

The FEATURE_SYNONYM table contains one row per feature having a synonym per synonym had by said feature. Typically a synonym exists so that somebody querying the db with an obsolete name can find the object they’re looking for (under its current name). FEATURE_SYNONYM allows for a many-to-many relationship between features and synonyms.

The combination of Synonym_Id, Feature_Id, and Pub_Id must be unique.

FEATURE_SYNONYM rows are automatically deleted when the related SYNONYM row is deleted. FEATURE_SYNONYM rows are automatically deleted when the related FEATURE row is deleted. FEATUREPROP rows are automatically deleted when the related PUB row is deleted.

3.1.17.1 Feature_Synonym_Id (Feature to Synonym Relationship Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.17.2 Synonym_Id (Synonym Identifier)

The synonym; an alternate name for the feature. A SYNONYM.Synonym_Id value. This column may not be NULL.

3.1.17.3 Feature_Id (Feature Identifier)

The feature having the synonym. A FEATURE.Feature_Id value. This column may not be NULL.

3.1.17.4 Pub_Id (Publication Identifier)

The publication in which the synonym was used to refer to the feature. A PUB.Pub_Id value. This column may not be NULL.

3.1.17.5 Is_Current

A Boolean. When TRUE the synonym is a relationship between the feature and the database object is a current, official, synonym for the feature. When FALSE the synonym is outdated. This column may not be NULL. The default value for this column is FALSE.

3.1.17.6 Is_Internal

A Boolean. When TRUE the synonym was never used publicly and intentionally (e.g. in a paper). This column may not be NULL. The default value for this column is FALSE.

3.1.18 FEATURELOC (Feature Locations)

FEATURELOC contains one row per feature or portion thereof which maps to a reference feature. The mapping coordinate system origin is zero-based and relative to the start of the reference feature.

Features typically have exactly ONE location but this need not be the case. Some features may not be localized (e.g. a gene that has been characterized genetically but no sequence or molecular information is available). Therefore each feature can have 0 or more locations. (However, multiple locations are not used to designate non-contiguous locations. If a feature such as a transcript has a non-contiguous location then the subfeatures such as exons should always be manifested). Multiple FEATURELOC rows for a feature designate alternate locations or grouped locations. For instance, a feature designating a blast hit or HSP will have two locations, one on the query feature and one on the subject feature. Features representing sequence variation could have alternate locations instantiated on a feature on the mutant strain. The column Rank is used to differentiate these different locations.

Reflexive locations should never be stored; FEATURELOC is for “proper” (i.e. non-self) locations only. Nothing should be located relative to itself.
Important

No cycles are allowed in the feature location graph.¹

¹ There are no validation rules in place to ensure this.

---

The use of FEATURELOC in Babase

Babase creates FEATURELOC rows to locate the SNV FEATURE rows is creates. See above.

---

The FMin value may not be greater than the FMax value.

---

Caution

There are no data validation rules in place to ensure correspondence between FMin and Is_FMin_Partial.

---

Caution

There are no data validation rules in place to ensure correspondence between FMax and Is_FMax_Partial.

---

The combination of Feature_Id, Locgroup, and Rank must be unique.

FEATURELOC rows are automatically deleted when any related FEATURE row is deleted, whether the row is related by Feature_Id or Srcfeature_Id.

Example 3.1 Using both FEATURELOC.Locgroup and FEATURELOC.Rank

Imagine a feature indicating a conserved region between the chromosomes of two different species. We may want to keep redundant locations on both contigs and chromosomes. We would thus have 4 locations for the single conserved region feature - two distinct Locgroups (contig level and chromosome level) and two distinct Ranks (for the two species).

---

3.1.18.1 Featureloc_Id (Feature Location Identifier)

A unique number identifying the mapping between the target feature (Feature_Id) and the reference feature (Srcfeature_Id). This column may not be NULL.

---

3.1.18.2 Feature_Id (Feature Identifier)

The feature being located. A FEATURE.Feature_Id value. This column may not be NULL.

---

3.1.18.3 Srcfeature_Id (Reference Feature Identifier)

The reference (or source or anchor) feature. The feature that the Feature_Id is located relative to. A FEATURE.Feature_Id value. This column may be NULL when the source feature is not known.
3.1.18.4 **FMin (Feature Lower Boundary Coordinate)**

The leftmost/minimal boundary in the linear range represented by the FEATURELOC row. Sometimes (e.g. in Bioperl) this is called “start” although this is confusing because it does not necessarily represent the 5-prime coordinate.

Important

**Chado** uses space-based (interbase) coordinates, counting from zero. To convert this to the leftmost position in a base-oriented system (e.g. GFF, Bioperl), add 1 to FMin.

Note that this value may be **NULL** when the lower boundary is not known.

3.1.18.5 **Is_FMin_Partial (Is the Feature Lower Boundary Coordinate Only Partially Known?)**

A Boolean. Whether or not the lower boundary coordinate of the feature (the FMin value) is partially known. This is typically **FALSE**, but may be **TRUE** if the value for FMin is inaccurate or the leftmost part of the range is unknown or unbounded.

This column may not be **NULL**. The default value for this column is **FALSE**.

3.1.18.6 **FMax (Feature Upper Boundary Coordinate)**

The rightmost/maximal boundary in the linear range represented by the FEATURELOC row. Sometimes (e.g. in Bioperl) this is called “end” although this is confusing because it does not necessarily represent the 3-prime coordinate.

Important

**Chado** uses space-based (interbase) coordinates, counting from zero. No conversion is required to go from FMax to the rightmost coordinate in a base-oriented system that counts from 1 (e.g. GFF, Bioperl).

Note that this value may be **NULL** when the lower boundary is not known.

3.1.18.7 **Is_FMax_Partial (Is the Feature Upper Boundary Coordinate Only Partially Known?)**

A Boolean. Whether or not the upper boundary coordinate of the feature (the FMax value) is partially known. This is typically **FALSE**, but may be **TRUE** if the value for FMax is inaccurate or the rightmost part of the range is unknown or unbounded.

This column may not be **NULL**. The default value for this column is **FALSE**.

3.1.18.8 **Strand (Strand Directionality)**

The orientation/directionality of the location. Should be **0**, **-1** or **+1**. However, **NULL** is also allowed. The difference between **NULL** and **0** is undocumented.

**The use of FEATURELOC.Strand within Babase**

Since Babase works relative to the reference genome, and is not presently interested in cleavage sites etc, the value of Strand will always be **+1**.

3.1.18.9 **Phase (Translation Phase)**

Phase of translation with respect to **Srcfeature_Id**. Acceptable values are **0**, **1**, **2** (although this is not enforced). It may not be possible to manifest this column for some features such as exons, because the phase is dependent on the spliceform (the same exon can appear in multiple spliceforms). This column is mostly useful for predicted exons and CDSs.
3.1.18.10 Residue_Info (Alternative Residue Information)

Alternative residues, when these differ from FEATURE.Residues. For instance, a SNP feature located on a wild and mutant protein would have different alternative residues. For alignment/similarity features this column is used to represent the alignment string (CIGAR format).

The use of FEATURELOC.Residue_Info and variation features

Even if we do not want to instantiate a mutant chromosome/contig feature we can still represent a SNP etc with 2 locations: one location (Rank 0) on the genome, the other (Rank 1) would have most fields NULL -- except for alternative residues.

The use of FEATURELOC.Residue_Info in Babase

Babase chooses to store alternative SNV residues in GENOTYPE.Description since these can be better related to multiple versions of the reference genome.

Babase may someday choose to store other alternative residues in ND_EXPERIMENTPROP.Value or ND_EXPERIMENT_STOCKPROP.Value since these also can be made to refer to more than one version of the reference genome.

3.1.18.11 Locgroup (Location Group)

The Locgroup column is used to manifest redundant, derivable extra locations for a feature. The default Locgroup of 0 is used for the direct location of a feature.

This column may not be NULL. The default value of this column is 0.

Important

Most Chado users may never use FEATURELOC rows with a Locgroup value greater than 0.

Transitively derived locations are indicated with Locgroup values greater than 0. For example the position of an exon on a BAC when represented in global chromosome coordinates would have a Locgroup value > 0.

The Locgroup column is used to differentiate the transitively derived groupings of locations. The default Locgroup 0 is used for the main or primary location, from which the others are derived via coordinate transformations.

Another example of redundant locations is storage of ORF coordinates relative to both transcript and genome.

Redundant locations open the possibility of the database getting into inconsistent states; this schema gives us the flexibility of both warehouse instantiations with redundant locations (easier for querying) and management instantiations with no redundant locations.

3.1.18.12 Rank

The Rank column differentiates between multiple locations when features have multiple locations and the semantics of the locations are different. This can best be explained by way of example: Some features (e.g. blast hits and HSPs) have two locations - one on the query and one on the subject. Rank is used to differentiate these. A Rank value of 0 is always used for the query, a Rank value of 1 for the subject. For multiple alignments, assignment of rank is arbitrary. Rank is also used for sequence_variant features, such as SNPs. In this usage a Rank value of 0 indicates the wildtype (or baseline) feature, a Rank value of 1 indicates the mutant (or compared) feature.

The use of FEATURELOC.Rank in Babase

Features on the reference genome located on the reference genome have a Rank value of 0. Features of individual baboons which are located on the reference genome have a Rank value of 1.

This column may not be NULL. The default value of this column is 0. In the typical case where a feature has a single location the Rank value is always 0, the default.
### 3.1.19 FEATUREPROP (Feature Properties)

The FEATUREPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the FEATURE table. FEATUREPROP contains one row for every (non-NULL) additional column value, per FEATUREPROP row, per Rank value.

The combination of Feature_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per feature per value type.

FEATUREPROP rows are automatically deleted when the related FEATURE row is deleted. FEATUREPROP rows are automatically deleted when the related CVTERM row is deleted.

#### 3.1.19.1 Featureprop_Id

A unique number identifying the FEATUREPROP row. This column may not be NULL.

#### 3.1.19.2 Feature_Id

The feature having the property -- the Feature_Id value of the related FEATURE row. This column may not be NULL.

#### 3.1.19.3 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

Types are expected to come from the Sequence Ontology’s feature property ontology but this is not enforced.

#### 3.1.19.4 Value

The feature property value -- as text.

#### 3.1.19.5 Rank

A number which gives the value an ordinal position among the other FEATUREPROP values, per the row’s Type_Id value, related to the FEATURE row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

### 3.1.20 GENOTYPE (Genotypes)

The GENOTYPE table contains one row per genotype of interest.

Genetic context. A genotype is defined by a collection of features, mutations, balancers, deficiencies, haplotype blocks, or engineered constructs.

---

**Note**

Because Babase’s focus is on individual baboons genotypes are not particularly of interest. They are, however, the way Chado links natural diversity experimental result sets (ND_EXPERIMENT rows) to features and so are typically part of the join path from experiment to feature.

In Babase the GENOTYPE table is not used. Instead the ND_EXPERIMENT_FEATURE table, a Babase extension, is used to relate ND_EXPERIMENT to FEATURE.

---

GENOTYPE rows are automatically deleted when the related CVTERM row is deleted.
3.1.20.1 Genotype_Id (Genotype Identifier)

A unique number identifying the genotype. This column may not be NULL.

3.1.20.2 Name

A human readable name for the genotype.

3.1.20.3 Uniquename

A unique name for the genotype. In Chado this is typically derived from the features making up the genotype. This column may not be NULL.

3.1.20.4 Description

A textual description of the genotype.

3.1.20.5 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.21 ND_EXPERIMENT (Natural Diversity Experiment Result Set)

ND_EXPERIMENT contains one row for every “set” of experimental results. A “set” of experimental results is typically not all results of what would normally be considered an experiment. Rather the ND_EXPERIMENT row represents disjoint portions of the entire collection of results partitioned according to one of the following schemes:

PARTITIONING OF EXPERIMENTAL RESULTS INTO ND_EXPERIMENT RESULT SETS

• Per genotype analyzed. Depending on how (and whether) the genotype is related to genomic features there can be a single ND_EXPERIMENT row per genomic feature or multiple ND_EXPERIMENT rows per genomic feature. Regardless, in this partitioning of experimental results there is always a 1-to-1 relationship (per any given experiment) between the ND_EXPERIMENT and the GENOTYPE tables.

Using this partitioning scheme per genotype results are stored in ND_EXPERIMENTPROP rows. (Should the genotype be uninteresting and there therefore is a 1-to-1 relationship between genotype and genomic features this system amounts to storing per genomic feature results in ND_EXPERIMENTPROP rows.) Per stock results are stored in ND_EXPERIMENT_STOCKPROP rows.

• Per stock analyzed. In this partitioning of experimental results there is always a 1-to-1 relationship (per any given experiment) between the STOCK and the ND_EXPERIMENT tables.

Using this partitioning scheme per stock results could be stored in ND_EXPERIMENTPROP rather than ND_EXPERIMENT_STOCKPROP.

Given the current Chado schema\(^3\) the “per stock analyzed” partitioning scheme is a degenerate case of the “per genotype analyzed” partitioning scheme because when partitioning experimental results in this fashion the schema design presently has no place\(^4\) to store per genotype (per any given experiment) results.

The use of ND_EXPERIMENT within Babase

Babase uses the “per genotype analyzed” experimental result partitioning scheme.

ND_EXPERIMENT rows are automatically deleted whenever any related ND_GEOLOCATION row is deleted.

---

\(^3\) Chado version 1.21.

\(^4\) A ND_EXPERIMENT_GENOTYPEPROP table perhaps?
3.1.21 ND_Experiment_Id (Natural Diversity Experiment Result Set Identifier)

A unique number identifying the natural diversity experiment result set. This column may not be NULL.

3.1.21.2 ND_Geolocation_Id (Natural Diversity Geolocation Identifier)

Identifier of the ND_GEOLOCATION row which geolocates the experimental result set. This column may not be NULL.

The use of ND_EXPERIMENT.ND_Geolocation_Id within Babase

This value is hugely uninteresting to Babase.

3.1.21.3 Type_Id

The type of experimental results the row holds. This column may not be NULL.

The use of ND_EXPERIMENT.Type_Id within Babase

In Babase it is usually more useful to use ANALYSIS_ND_EXPERIMENT.Type_Id to classify the experimental results. The value of this column tends to be uninteresting.

3.1.22 ND_EXPERIMENT_GENOTYPE (Natural Diversity Experiment Result Set to Genotype Relationships)

The ND_EXPERIMENT_GENOTYPE table contains one row per every relationship between a natural diversity result set and a genotype -- that is, one row per pairing of ND_EXPERIMENT row with a GENOTYPE row.

ND_EXPERIMENT_GENOTYPE relates genotype to natural diversity experiment result sets in, potentially, a many-to-many fashion. The table contains one row for every pairing of genotype to natural diversity experiment result set.

Tip

Using ND_EXPERIMENT_GENOTYPE it is possible to construct many-to-many relationships between GENOTYPE rows and ND_EXPERIMENT rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable, at least not per analysis\. It is probably more appropriate to, per analysis, construct one-to-many relationships between GENOTYPE and ND_EXPERIMENT rows, or the reverse, to construct many-to-one relationships between GENOTYPE and ND_EXPERIMENT rows. In some cases it is appropriate to construct one-to-one relationships between GENOTYPE and ND_EXPERIMENT rows.

\[\] Where an analysis is an analysis of an experiment; in effect, where an analysis is an actual experiment.

The use of ND_EXPERIMENT_GENOTYPE within Babase

Babase uses its Chado extension, ND_EXPERIMENT_FEATURE, instead of ND_EXPERIMENT_GENOTYPE. ND_EXPERIMENT_GENOTYPE is not used.

The combination of ND_Experiment_Id and Genotype_Id must be unique.

ND_EXPERIMENT_GENOTYPE rows are automatically deleted whenever any related ND_EXPERIMENT row is deleted. ND_EXPERIMENT_GENOTYPE rows are automatically deleted whenever any related GENOTYPE row is deleted. ND_EXPERIMENT rows are automatically deleted whenever any related CVTERM row is deleted.

3.1.22.1 ND_Experiment_Genotype_Id

A unique number identifying the pairing of GENOTYPE row to ND_EXPERIMENT row. This column may not be NULL.
3.1.22.2 Genotype_Id

The genotype, typically the input material, related to the natural diversity experiment result set. An GENOTYPE.Genotype_Id value. This column may not be NULL.

3.1.22.3 ND_Experiment_Id

The natural diversity experiment result set related to the genotype. A ND_EXPERIMENT.ND_Experiment_Id value. This column may not be NULL.

3.1.23 ND_EXPERIMENT_STOCK (Natural Diversity Experiment Result Set to Stock Relationships)

The ND_EXPERIMENT_STOCK table contains one row per every relationship between a natural diversity result set and a stock -- that is, one row per pairing of ND_EXPERIMENT row with a STOCK row.

ND_EXPERIMENT_STOCK relates stock to natural diversity experiment result sets in, potentially, a many-to-many fashion. The table contains one row for every pairing of stock to natural diversity experiment result set.

Tip

Using ND_EXPERIMENT_STOCK it is possible to construct many-to-many relationships between STOCK rows and ND_EXPERIMENT rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable, at least not per analysis\(^a\). It is probably more appropriate to, per analysis, construct one-to-many relationships between STOCK and ND_EXPERIMENT rows, or the reverse, to construct many-to-one relationships between STOCK and ND_EXPERIMENT rows.

\(^a\) Where an analysis is an analysis of an experiment; in effect, where an analysis is an actual experiment.

The use of ND_EXPERIMENT_STOCK within Babase

When doing analyses in Babase, whether of SNV sites or functional genomic analysis, an ND_EXPERIMENT row represents some set of results of an analysis regarding a genomic feature. Therefore the typical Babase use-case has multiple ND_EXPERIMENT rows per STOCK row per analysis.

ND_EXPERIMENT_STOCK rows are automatically deleted whenever any related ND_EXPERIMENT row is deleted. ND_EXPERIMENT_STOCK rows are automatically deleted whenever any related STOCK row is deleted. ND_EXPERIMENT_STOCK rows are automatically deleted whenever any related CVTERM row is deleted.

3.1.23.1 ND_Experiment_Stock_Id

A unique number identifying the pairing of STOCK row to ND_EXPERIMENT row. This column may not be NULL.

3.1.23.2 Stock_Id

The stock, typically the input material, related to the natural diversity experiment result set. An STOCK.Stock_Id value. This column may not be NULL.

3.1.23.3 ND_Experiment_Id

The natural diversity experiment result set related to the stock. A ND_EXPERIMENT.ND_Experiment_Id value. This column may not be NULL.
3.1.23.4 Type_Id

The type of relationship between the stock and the natural diversity experiment result set -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.24 ND_EXPERIMENT_STOCKPROP (Natural Diversity Experiment Result Set to Stock Relationship Properties)

The ND_EXPERIMENT_STOCKPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the ND_EXPERIMENT_STOCK table. ND_EXPERIMENT_STOCKPROP contains one row for every (non-NULL) additional column value, per ND_EXPERIMENT_STOCK row, per Rank value.

The use of ND_EXPERIMENT_STOCKPROP within Babase

In some cases this table may contain information on treatment of the stock. In Babase at least some of the rows hold values that represent experimental results.

The combination of ND_Experiment_Stock_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per natural diversity experiment result set per value type.

ND_EXPERIMENT_STOCKPROP rows are automatically deleted when the related ND_EXPERIMENT_STOCK row is deleted. ND_EXPERIMENT_STOCKPROP rows are automatically deleted when the related CVTERM row is deleted.

3.1.24.1 ND_Experiment_Stockprop_Id

A unique number identifying the ND_EXPERIMENT_STOCKPROP row. This column may not be NULL.

3.1.24.2 ND_Experiment_Stock_Id

The natural diversity result set to stock relationship having the property -- the ND_Experiment_Stock_Id value of the related ND_EXPERIMENT_STOCK row. This column may not be NULL.

3.1.24.3 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.24.4 Value

The natural diversity experimental result set to stock relationship property value -- as text.

3.1.24.5 Rank

A number which gives the value an ordinal position among the other ND_EXPERIMENT_STOCKPROP values, per the row’s Type_Id value, related to the ND_EXPERIMENT_STOCK row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].
3.1.25 ND_EXPERIMENTPROP (Natural Diversity Experiment Result Set Properties)

The ND_EXPERIMENTPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the ND_EXPERIMENT table. ND_EXPERIMENTPROP contains one row for every (non-NULL) additional column value, per ND_EXPERIMENTPROP row, per Rank value.

The combination of ND_Experiment_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per natural diversity experiment result set per value type.

ND_EXPERIMENTPROP rows are automatically deleted when the related ND_EXPERIMENT row is deleted. ND_EXPERIMENTPROP rows are automatically deleted when the related CVTERM row is deleted.

3.1.25.1 ND_Experimentprop_Id

A unique number identifying the ND_EXPERIMENTPROP row. This column may not be NULL.

3.1.25.2 ND_Experiment_Id

The natural diversity result set having the property -- the ND_Experiment_Id value of the related ND_EXPERIMENT row. This column may not be NULL.

3.1.25.3 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.25.4 Value

The natural diversity result set property value -- as text.

3.1.25.5 Rank

A number which gives the value an ordinal position among the other ND_EXPERIMENTPROP values, per the row’s Type_Id value, related to the ND_EXPERIMENT row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.26 ND_GEOLOCATION (Natural Diversity Geolocation)

The ND_GEOLOCATION table contains one row for every location of a natural diversity experiment.

<table>
<thead>
<tr>
<th>The use of ND_GEOLOCATION within Babase</th>
</tr>
</thead>
<tbody>
<tr>
<td>This table is not of interest to Babase.</td>
</tr>
</tbody>
</table>

3.1.26.1 ND_Geolocation_Id (Natural Diversity Geolocation Identifier)

A unique number identifying the row. This column may not be NULL.

3.1.26.2 Description

A textual representation of the location, if this is the original georeference.
3.1.26  Latitude

The decimal latitude coordinate of the georeference, using positive and negative sign to indicate N and S, respectively.

3.1.26  Longitude

The decimal longitude coordinate of the georeference, using positive and negative sign to indicate E and W, respectively.

3.1.26  Geodetic

The geodetic system on which the geo-reference coordinates are based.

3.1.26  Altitude

The altitude (elevation) of the location in meters.

3.1.27  ORGANISM

The ORGANISM table contains one row per organismal taxonomic classification.

Note that phylogenies are represented using the phylogeny module, and taxonomies can be represented using the cvterm module or the phylogeny module.

The combination of Genus and Species must be unique. When mapping from the NCBI taxonomy names.dmp file, this column must be used where it is present, as the common_name column is not always unique (e.g. environmental samples). If a particular strain or subspecies is to be represented, this is appended onto the species name. Follows standard NCBI taxonomy pattern.

3.1.27.1  Organism_Id (Organism Identifier)

A unique number representing the organism. This column may not be NULL.

3.1.27.2  Abbreviation

The abbreviated name.

3.1.27.3  Genus

The genus portion of the organism’s taxonomic classification. This column may not be NULL.

3.1.27.4  Species

The species portion of the organism’s taxonomic classification with additional information appended to provide further specification of sub-species, strain, environment, individual and so forth. This column may not be NULL.

3.1.27.5  Common_Name

The common name of the organism.

Should Babase decide to represent individuals as organisms then this column will contain the individual’s Sname in Babase:.

3.1.27.6  Comment

Additional remarks regards the organism.
3.1.28 ORGANISM_DBXREF (Organism to External Database Object Cross-References)

ORGANISM_DBXREF contains one row per external database object related to an organism.

The use of ORGANISM_DBXREF within Babase

In the case of Babase, ORGANISM_DBXREF is used to equate individual baboons (rows in the BIOGRAPH in Babase: table) to organisms.

The combination of Organism_Id and DBXref_Id must be unique.

ORGANISM_DBXREF rows are automatically deleted when the related ORGANISM row is deleted. ORGANISM_DBXREF rows are automatically deleted when the related DBXREF row is deleted.

3.1.28.1 Organism_DBxref_Id

A unique number identifying the row. This column may not be NULL.

3.1.28.2 Organism_Id

Identifier of the ORGANISM row with which the external database object is related. An ORGANISM.Organism_Id value. This column may not be NULL.

3.1.28.3 DBXref_Id (External Database Object Identifier)

Identifier of the external database object related to the ORGANISM row. A DBXREF.DBXref_Id value. This column may not be NULL.

3.1.29 PUB (Publication or Other Documentation of Provenance)

The PUB table contains one row for each document which somehow records the provenance of something represented in the Chado database. The documents can be publications, personal communications, or anything else of import.

The use of PUB within Babase

Babase is not interested in the rows of this table.

PUB rows are automatically deleted when the related CVTERM row is deleted.

3.1.29.1 Pub_Id (Publication Identifier)

A unique number representing the publication. This column may not be NULL.

3.1.29.2 Title

The title of the publication. A descriptive general heading.

3.1.29.3 Volumetitle (Title of Volume)

The title of the volume, if the publication is part of a series.
3.1.29.4 Volume
Volume number or other identifier of the volume.

3.1.29.5 Series_Name
Full name of (journal) series.

3.1.29.6 Issue
Issue number or other identifier of the issue.

3.1.29.7 PYear (Publication Year)
Year of the publication.

3.1.29.8 Pages
Page number range[s], e.g. 457--459, viii + 664pp, lv--lvii.

3.1.29.9 Miniref
The intended content of this column is not clear.

3.1.29.10 Uniquename
A unique name for the publication. This column may not be NULL.

3.1.29.11 Type_Id
The type of the publication -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.29.12 Is_Obsolete
A Boolean value indicating whether or not the publication is appropriate to reference in future work. The default is FALSE, future reference to the row is appropriate. This column may not be NULL.

3.1.29.13 Publisher
The publisher of the publication. Note that this column is not validated. Storing this data in another publication related Chado table may result in better data integrity.

3.1.29.14 Pubplace (Publication Place)
The place of publication. Note that this column is not validated. Storing this data in another publication related Chado table may result in better data integrity.
3.1.30 STOCK (Stocks)

The STOCK table contains one row per physical item, whether stored or living. Generally stocks are physical things preserved over time. Typically, stocks are maintained in collections and used as input to genetic experiments.

STOCK rows are assigned primary and secondary types in the usual Chado fashion. There is a STOCK.Type_Id column to denote the stock’s primary type, and a STOCK_CVTERM table to record other types/tags associated with the stock.

STOCK rows can be related to objects in external databases by way of the DBXREF table’s external database object representation. STOCK rows may be representations of external database objects, i.e. be related to a primary external database object by way of STOCK.DBXref_Id, and may also be related to multiple secondary external database objects via the STOCK_DBXREF table.

The use of STOCK within Babase

Babase uses STOCK rows to represent tissue samples, DNA extracts, and derivatives and sub-divisions of these; including the library components that are mixed to produce the libraries which are inputs to the high-throughput DNA sequencing process.

When a STOCK row represents a tissue sample or a DNA extract it’s STOCK.DBXref_Id contains the appropriate value which denotes this.

The STOCK.Uniquename value is unique per STOCK.Organism_Id per STOCK.Type_Id -- to uniquely identify a stock all three values must be specified.

STOCK rows are automatically deleted when the related DBXREF row is deleted. STOCK rows are automatically deleted when the related ORGANISM row is deleted. STOCK rows are automatically deleted when the related CVTERM row is deleted.

3.1.30.1 Stock_Id (Stock Identifier)

A unique number identifying the stock. This column may not be NULL.

3.1.30.2 DBXref_Id (External Database Object Cross-Reference Identifier)

The primary external database object represented by the row -- the related DBXREF.DBXref_Id value.

3.1.30.3 Organism_Id (Organism Identifier)

The organism which is the source of the stock -- the Organism_Id value of the related ORGANISM row. This value should be NULL only when the organism cannot be determined.

3.1.30.4 Name

The human-readable name of the stock.

3.1.30.5 Uniquename

A name that is unique per organism per type. See above. This column may not be NULL.

3.1.30.6 Description

A textual description of the stock.

3.1.30.7 Type_Id (Type)

The kind of stock that the row holds -- a CVTERM.CVTerm_Id value. Typical types would be “living stock”, “genomic DNA”, or “preserved specimen”. This column may not be NULL.
3.1.30.8  IsObsolete

A Boolean value indicating whether or not the stock is no longer in use. This column may not be NULL. Consider recording the date the stock became obsolete in the STOCKPROP table.

3.1.31  STOCK_DBXREF (Stock to External Database Object Cross-References)

STOCK_DBXREF contains one row per external database object related to an stock.

When used within Babase

In the case of Babase, STOCK_DBXREF could be used to relate stock to other parts of Babase in undetermined ways. A possible use is to relate stock to documents which describe their preparation etc.

The combination of Stock_Id and DBXref_Id must be unique.

STOCK_DBXREF rows are automatically deleted when the related STOCK row is deleted. STOCK_DBXREF rows are automatically deleted when the related DBXREF row is deleted.

3.1.31.1  Stock_DBxref_Id

A unique number identifying the row. This column may not be NULL.

3.1.31.2  Stock_Id (Stock Identifier)

Identifier of the STOCK row with which the external database object is related. An STOCK.Stock_Id value. This column may not be NULL.

3.1.31.3  DBXref_Id (External Database Object Identifier)

Identifier of the external database object related to the STOCK row. A DBXREF.DBXref_Id value. This column may not be NULL.

3.1.31.4  Is_Current

A Boolean. When TRUE the relationship between the stock and the external database object is in current use. When FALSE the relationship is outdated. This column may not be NULL. The default value for this column is TRUE.

3.1.32  STOCK_DBXREFPROP (Stock External Database Entity Properties)

The STOCK_DBXREFPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the STOCK_DBXREF table. STOCK_DBXREFPROP contains one row for every (non-NULL) additional column value, per STOCK_DBXREF row, per Rank value.

The combination of Stock_DBXref_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per stock_dbxref per value type.

STOCK_DBXREFPROP rows are automatically deleted when the related STOCK_DBXREF row is deleted. STOCK_DBXREFPROP rows are automatically deleted when the related CVTERM row is deleted.

3.1.32.1  Stock_DBxrefprop_Id

A unique number identifying the STOCK_DBXREFPROP row. This column may not be NULL.
3.1.32.2 Stock_DBXref_Id

The external database entry/stock combination having the property -- the Stock_DBXref_Id value of the related STOCK_DBXREF row. This column may not be NULL.

3.1.32.3 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.32.4 Value

The external database entry/stock combination’s property value -- as text.

3.1.32.5 Rank

A number which gives the value an ordinal position among the other STOCK_DBXREFPROP values, per the row’s Type_Id value, related to the STOCK_DBXREF row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.33 STOCK_CVTERM (Stock Typing/Tagging)

STOCK_CVTERM contains one row per stock, per publication, per each ancillary type or tag used to classify the stock. As usual in Chado, the CVTERM table is used to supply the vocabulary (or vocabularies) used in classification.

The use of STOCK_CVTERM within Babase
Babase is uninterested in publication. Every row has an identical Pub_Id value which may be safely ignored.

The combination of Stock_id, CVTerm_Id, Pub_Id, and Rank must be unique.

STOCK_CVTERM rows are automatically deleted when the related STOCK row is deleted. STOCK_CVTERM rows are automatically deleted when the related PUB row is deleted. STOCK_CVTERM rows are automatically deleted when the related CVTERM row is deleted.

3.1.33.1 Stock_CVTerm_Id

A unique number identifying the row. This column may not be NULL.

3.1.33.2 Stock_Id

The stock assigned an ancillary type or tag. An STOCK.Stock_Id value. This column may not be NULL.

3.1.33.3 CVTerm_Id (Controlled Vocabulary Term Identifier)

The ancillary type or tag applied to the stock. A CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.33.4 Pub_Id (Publication Identifier)

The publication associated with the typing or tagging of the stock. A PUB.Pub_Id value. This column may not be NULL.
3.1.33.5 **Is_Not**

A Boolean value which serves to negate the type or tag (to negate the controlled vocabulary term). This column may not be NULL.

3.1.33.6 **Rank**

A number which gives the value an ordinal position among the other STOCK_CVTERM values, per the row’s Pub_Id value, related to the STOCK row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.34 **STOCK_RELATIONSHIP (Stock Inter-Relationships)**

STOCK_RELATIONSHIP contains one row per subject/verb/object or parent/child relationship between 2 stocks. This table allows organization of stock into acyclic directed graphs to record things like which stock was divided into which sub-stocks, which stock derived from which other stocks, and so forth. It is one of the many “relationship” tables in Chado.

The combination of Subject_Id, Object_Id, Type_Id, and Rank must be unique.

STOCK_RELATIONSHIP rows are automatically deleted when either the related Subject_Id STOCK row is deleted or when the related Object_Id STOCK row is deleted. STOCK_RELATIONSHIP rows are automatically deleted when the related CVTERM row is deleted.

3.1.34.1 **Stock_Relationship_Id**

A unique number identifying the row. This column may not be NULL.

3.1.34.2 **Subject_Id**

The subject of the subject-predicate-object sentence. A STOCK.Stock_Id value. This column may not be NULL.

3.1.34.3 **Object_Id**

The object of the subject-predicate-object sentence. A STOCK.Stock_Id value. This column may not be NULL.

3.1.34.4 **Type_Id**

The relationship between the subject stock and the object stock. A CVTERM.CVTerm_Id value that has a non-0 Is_Relationship_type value. This column may not be NULL.

3.1.34.5 **Value (Additional Notes)**

Additional textual notes on the relationship between the analyses.

3.1.34.6 **Rank**

A number which gives the value an ordinal position among the other STOCK_RELATIONSHIP values, per Subject_Id/Object_Id pairing, related to the CVTERM row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].
3.1.35 STOCKPROP (Stock Properties)

The STOCKPROP table is a Chado property table, it allows for the (effective) addition of arbitrary columns to the STOCK table. STOCKPROP contains one row for every (non-NULL) additional column value, per STOCK row, per Rank value.

The combination of Stock_Id, Type_Id, and Rank must be unique. Note that this arrangement allows for duplicate values per stock per value type.

STOCKPROP rows are automatically deleted when the related STOCK row is deleted. STOCKPROP rows are automatically deleted when the related CVTERM row is deleted.

3.1.35.1 Stockprop_Id

A unique number identifying the STOCKPROP row. This column may not be NULL.

3.1.35.2 Stock_Id

The stock having the property -- the Stock_Id value of the related STOCK row. This column may not be NULL.

3.1.35.3 Type_Id

The kind of value that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL.

3.1.35.4 Value

The stock’s property value -- as text.

3.1.35.5 Rank

A number which gives the value an ordinal position among the other STOCKPROP values, per the row’s Type_Id value, related to the STOCK row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.1.36 SYNONYM (Alternate Names)

The SYNONYM table contains one row for every alternate name of a feature. One feature can have multiple synonyms, and the same synonym can apply to multiple features.

The combination of Name and Type_Id must be unique.

SYNONYM rows are automatically deleted when related CVTERM row is deleted.

Caution
There is no validation ensuring correspondence between Name and Synonym-SGML.

3.1.36.1 Synonym_Id (Synonym Identifier)

A unique number which identifies the synonym. This column may not be NULL.
3.1.36.2 Name (Synonym)

The synonym. Human readable ASCII text. This column may not be NULL.

3.1.36.3 Type_Id (Type)

The kind of synonym that the row holds -- a CVTERM.CVTerm_Id value. This column may not be NULL. Current types equate to “symbol” and “fullname”.

3.1.36.4 Synonym_SGML

The synonym with non-ASCII characters encoded as SGML entities.

3.2 Babase Chado Extensions

Babase has extended Chado. The table related changes are documented in this section.

3.2.1 ANALYSIS (Analyses)

The ANALYSIS table contains one row per computational analysis.

The extension of the ANALYSIS table within Babase

The ANALYSIS table has been extended by Babase with the addition of the Type_Id column and a requirement that ANALYSIS.Name be unique. ANALYSIS is therefore documented in this section with the new tables Babase has added to Chado.

Note

In Babase computational analyses are the result of high throughput genetic sequencing. There are two broad classes of results, individual baboon SNV sites and functional genomic analysis of individual baboons. SNV analysis yields annotations of the genome, the locations of SNV sites. The SNV sites are in turn related to genetic samples and hence to individual baboons. Functional genomic analysis is also related to features on the genome but does not necessarily in and of itself result in new feature annotations. Both classes of analysis produce per individual baboon per feature metrics. Functional genomic analysis tends to produce a single metric per feature per baboon. SNV site analysis produces 3 metrics per feature per baboon, one each for homozygous reference, heterozygous, and homozygous alternate. SNV analysis also produces an additional metric per SNV site.

* This is not to say that the individual baboon is the unit of analysis. Rather that the result of analyses yields results which pertain to individual baboons.

Computational analysis can be related to each other to capture the many steps and combinations of analytical results that may go into a final analysis. (See ANALYSIS_RELATIONSHIP.)

3.2.1.1 Analysis_Id

Unique numeric identifier of the ANALYSIS row. This column may not be NULL.

3.2.1.2 Name

A short name for the analysis. This need not be unique.
3.2.1.3 Description

Textual description of the analysis.

3.2.1.4 Program

The program used to run the analysis. This column may not be NULL.

_Note_
This column is not validated. It may be better to use the ANALYSISPROP table, supplying the program version as the value, or even the ANALYSIS_CVTERM table, each version of each program having a separate controlled vocabulary term. Further, a Babase analysis may be the result of more than one program. A decision is required as to how to use this column in Babase, especially as it may not be NULL.

3.2.1.5 Programversion

The version of the program used to run the analysis. This column may not be NULL. (But see the note regards Program above.)

3.2.1.6 Algorithm

The algorithm used in the analysis. (But see the note regards Program above.)

3.2.1.7 Sourcename

Where the data came from that was used in the analysis.

_Note_
Since Babase analyses tend to use samples collected from individual baboons this column is probably uninteresting and is unlikely to be used.

3.2.1.8 Sourceversion

Version associated with Sourcename.

3.2.1.9 SourceURI

The URL associated with Sourcename.

_Note_
While this column is potentially useful, the ANALYSIS_DBXREF table provides a more powerful alternative.

3.2.1.10 Timeexecuted

The date and time the analysis was executed. This column may not be NULL.

_Note_
Because Babase analyses may be complex and require many steps that take place over time this column is less useful. None the less, it is probably a good idea to put a value in this column to record when the analysis was complete. This column, being an actual timestamp datatype, is readily searchable, sortable, and so forth. Other dates related to the analysis may be optionally recorded in the ANALYSISPROP table.
3.2.1.11  Term_Id

The primary “type” of the analysis. Ancillary “types” may be recorded in ANALYSIS_CVTERM.

Note
In Babase the primary types would be things like “gene expression” or “DNA methylation”.

3.2.2  ANALYSIS_DBXREF (Analysis to External Database Object Cross-References)

ANALYSIS_DBXREF contains one row per external database object related to an analysis. Possible external database objects are inputs into or outputs of the analysis.

Note
In the case of Babase, ANALYSIS_DBXREF is used to relate the FASTQ files which went into the analysis to the analysis.

The combination of Analysis_Id and DBXref_Id must be unique.
ANALYSIS_DBXREF rows are automatically deleted when the related ANALYSIS row is deleted. ANALYSIS_DBXREF rows are automatically deleted when the related DBXREF row is deleted.

3.2.2.1  Analysis_DBXref_Id

A unique number identifying the row. This column may not be NULL.

3.2.2.2  Analysis_Id

Identifier of the ANALYSIS row with which the external database object is related. An ANALYSIS.Analysis_Id value. This column may not be NULL.

3.2.2.3  DBXref_Id (External Database Object Identifier)

Identifier of the external database object related to the ANALYSIS row. A DBXREF.DBXref_Id value. This column may not be NULL.

3.2.2.4  Is_Current

A Boolean. When TRUE the relationship between the analysis and the external database object is a current, official, relationship. When FALSE the relationship is outdated. This column may not be NULL. The default value for this column is TRUE

3.2.3  ANALYSIS_CVTERM (Analysis Typing/Tagging)

ANALYSIS_CVTERM contains one row per analysis per each ancillary type or tag used to classify the analysis. As usual in Chado, the CVTERM table is used to supply the vocabulary (or vocabularies) used in classification.

Note
In Babase ancillary types could be things like “preliminary result” or “final result”.

The combination of Analysis_id and CVTerm_Id must be unique.
ANALYSIS_CVTERM rows are automatically deleted when the related ANALYSIS row is deleted. ANALYSIS_CVTERM rows are automatically deleted when the related CVTERM row is deleted.
3.2.3.1 Analysis_CVTerm_Id

A unique number identifying the row. This column may not be NULL.

3.2.3.2 Analysis_Id

The analysis assigned an ancillary type or tag. An ANALYSIS.Analysis_Id value. This column may not be NULL.

3.2.3.3 CVTerm_Id (Controlled Vocabulary Term Identifier)

The ancillary type or tag applied to the analysis. A CVTERM.CVTerm_Id value. This column may not be NULL.

3.2.3.4 Is_Not

A Boolean value which serves to negate the type or tag (to negate the controlled vocabulary term). This column may not be NULL.

3.2.4 ANALYSIS_RELATIONSHIP (Analysis Inter-Relationships)

ANALYSIS_RELATIONSHIP contains one row per subject/verb/object or parent/child relationship between 2 analyses. This table allows organization of analyses into acyclic directed graphs to record things like which preliminary analysis went into a final analysis, or even which set of analytic results are a subset of some larger result. It is one of the many “relationship” tables in Chado.

The combination of Subject_Id, Object_Id, Type_Id, and Rank must be unique.

ANALYSIS_RELATIONSHIP rows are automatically deleted when either the related Subject_Id ANALYSIS row is deleted or when the related Object_Id ANALYSIS row is deleted. ANALYSIS_RELATIONSHIP rows are automatically deleted when the related CVTERM row is deleted.

3.2.4.1 Analysis_Relationship_Id

A unique number identifying the row. This column may not be NULL.

3.2.4.2 Subject_Id

The subject of the subject-predicate-object sentence. An ANALYSIS.Analysis_Id value. This column may not be NULL.

3.2.4.3 Object_Id

The object of the subject-predicate-object sentence. An ANALYSIS.Analysis_Id value. This column may not be NULL.

3.2.4.4 Type_Id

The relationship between the subject analysis and the object analysis. A CVTERM.CVTerm_Id value that has a non-0 Is_Relationship_type value. This column may not be NULL.

3.2.4.5 Value (Additional Notes)

Additional textual notes on the relationship between the analyses.
3.2.4.6 Rank

A number which gives the value an ordinal position among the other ANALYSIS_RELATIONSHIP values, per Subject_Id/Object_Id pairing, related to the CVTERM row. This column may not be NULL.

The default Rank value is 0.

For more on Rank values see The Chado Rank Column [1].

3.2.5 ANALYSIS_ND_EXPERIMENT (Analysis to Natural Diversity Experiment Result Set Relationships)

ANALYSIS_ND_EXPERIMENT relates analyses to natural diversity experiment result sets in, potentially, a many-to-many fashion. The table contains one row for every pairing of analysis to natural diversity experiment result set.

---

**Caution**

Care should be taken when using this table and the Chado genetics module. Use of both can result in multiple links between the ND_EXPERIMENT and the FEATURE table, by way of different join paths. The Chado genetics module links the FEATURE and ND_EXPERIMENT tables by way of the GENOTYPE table. The Chado Companalysis module links FEATURE to ND_EXPERIMENT, using this table, by way of the ANALYSIS table. It is therefore possible to introduce inconsistent relationships into the database should the 2 different join paths end at different FEATURE rows when this is inappropriate.

---

**Tip**

Using ANALYSIS_ND_EXPERIMENT it is possible to construct many-to-many relationships between ANALYSIS rows and ND_EXPERIMENT rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable. It is probably more appropriate to construct one-to-many relationships between ANALYSIS and ND_EXPERIMENT rows, or the reverse, to construct many-to-one relationships between ANALYSIS and ND_EXPERIMENT rows.

---

**Note**

When doing analyses in Babase, whether of SNV sites or functional genomic analysis, an ND_EXPERIMENT row represents the result set of an analysis of a genomic feature. Therefore the typical Babase use-case has multiple ND_EXPERIMENT rows per ANALYSIS row.

---

3.2.5.1 Analysis_ND_Experiment_Id

A unique number identifying the pairing of ANALYSIS row to ND_EXPERIMENT row. This column may not be NULL.

3.2.5.2 Analysis_Id

The analysis related to the natural diversity experiment result set. An ANALYSIS.Analysis_Id value. This column may not be NULL.

3.2.5.3 ND_Experiment_Id

The natural diversity experiment result set related to the analysis. A ND_EXPERIMENT.ND_Experiment_Id value. This column may not be NULL.

3.2.5.4 Type_Id

The type of relationship between the analysis and the natural diversity experiment result set -- a CVTERM.CVTerm_Id value. This column may not be NULL.
3.2.6 ND_EXPERIMENT_FEATURE (Natural Diversity Experiment Result Set to Feature Relationships)

The ND_EXPERIMENT_FEATURE table contains one row per every relationship between a natural diversity result set and a feature -- that is, one row per pairing of ND_EXPERIMENT row with a FEATURE row.

ND_EXPERIMENT_FEATURE relates feature to natural diversity experiment result sets in, potentially, a many-to-many fashion. The table contains one row for every pairing of feature to natural diversity experiment result set.

---

Tip

Using ND_EXPERIMENT_FEATURE it is possible to construct many-to-many relationships between FEATURE rows and ND_EXPERIMENT rows, but care should be taken since it is unlikely that such many-to-many relationships are desirable, at least not per analysis\(^a\). It is probably more appropriate to, per analysis, construct one-to-many relationships between FEATURE and ND_EXPERIMENT rows, or the reverse, to construct many-to-one relationships between FEATURE and ND_EXPERIMENT rows. In some cases it is appropriate to construct one-to-one relationships between FEATURE and ND_EXPERIMENT rows.

\(^a\) Where an analysis is an analysis of an experiment; in effect, where an analysis is an actual experiment.

The combination of ND_Experiment_Id and Feature_Id must be unique.

ND_EXPERIMENT_FEATURE rows are automatically deleted whenever any related ND_EXPERIMENT row is deleted. ND_EXPERIMENT rows are automatically deleted whenever any related FEATURE row is deleted. ND_EXPERIMENT_FEATURE rows are automatically deleted whenever any related CVTERM row is deleted.

3.2.6.1 ND_Experiment_Feature_Id

A unique number identifying the pairing of FEATURE row to ND_EXPERIMENT row. This column may not be NULL.

3.2.6.2 Feature_Id

The feature, typically an analyzed site in some input material, related to the natural diversity experiment result set. An FEATURE.Feature_Id value. This column may not be NULL.

3.2.6.3 ND_Experiment_Id

The natural diversity experiment result set related to the feature. A ND_EXPERIMENT.ND_Experiment_Id value. This column may not be NULL.
Chapter 4

Chado Programs

4.1 Command Line Programs

These are Unix command line programs.

All commands take the -h and --help options to display a short usage summary and take the options -v --version to display the program’s version.

Boolean values are specified from the command line as follows: false, 0, f, no, n and the empty string are false, as are any various in the case of these values. All other value (such as True, 1, Yes, y, etc.) are true.

4.1.1 abrp-genomics-update-testdir

abrp-genomics-update-testdir — update symlinks in the test portions of the /mnt/abrp_genomics hierarchy

Synopsis

abrp-genomics-update-testdir

Description

abrp-genomics-update-testdir puts symlinks to all the “live” files in the /mnt/abrp_genomics hierarchy into the test portions of the hierarchy. This gives the test website access to “live” data files, in addition to whatever test data files may also be present and accessible via the content of the test database.

abrp-genomics-update-testdir must be manually run after changes are made to “live” set of data files in order that the collection of test data files reflect the collection of “live” data files.\(^1\)

4.1.2 chado-funcgen-dump

chado-funcgen-dump — produce a zip file containing files of functional genomic analysis results from Babase/Chado; command line and web interfaces exist

\(^1\) Because the test data files are symlinks to the actual “live” data files, updating the content within a “live” data file is automatically reflected in the content of the corresponding test data file. It is the creation of a new file or the deletion of an old file that abrp-genomics-update-testdir synchronizes.
Synopsis

[--dbname=dbname] [--host=hostname] [--port=port] [--username=username]
[--analysis-query=aquery] [--stock-query=squery] [--anonymize]

Description

chado-funcgen-dump produces a zipfile, filepath, containing multiple files. Each of the files in the zipfile is all or selected columns of files previously uploaded with the chado-funcgen-load command.

Note

A .zip suffix is not automatically added to the end of the supplied filepath.

The chado-funcgen-dump program has an interface which allows it to be executed from the web. See: the section called “The Web Interface”.

An output file is created within the zip file for each selected analysis, if that analysis contains any columns from any of the selected stock. If an analysis matches the analysis selection criteria but contains no columns matching the selected stock then nothing is output for the analysis. Each output file contains those columns matching the stock selection criteria. Thus, the output contains the intersection of the matched analysis with the matched stock. If no matching analysis contain matching stock columns then the zip file produced is empty.

Optionally, the analysis files may be anonymized. See the --anonymize option.

If filepath exists before chado-funcgen-dump is run, it is deleted. The supplied filepath must be an actual file, stdout cannot be used.

Unless otherwise noted all matching is done in a case-sensitive fashion.

Arguments

Caution

CVTERM rows specified by command line arguments are not checked for obsolescence. The CVTERM.Is_Obsolete value is not examined. It is assumed that the user knows whether or not an obsolete term should be used.

SQL queries need not (but may) end in a semi-colon.

DATABASE OPTIONS

--dbname=dbname The (optional) name of the database to which to connect or a conninfo connection string. Defaults to the user’s Unix username.

--host=hostname The (optional) host name of the machine on which the server is running. If the value begins with a slash it is used as the directory for the Unix-domain socket.

--port=port The (optional) TCP port or the local Unix-domain socket file extension on which the server is listening for connections.

--username=username (Optional) Username to use to connect to the database. Defaults to the user’s Unix username.

DATA CHOICE

2 A sample conninfo connection string might look like user='scott', host='localhost', dbname='babase_test'.
--analysis-query=

An (optional) query which selects the analysis for which results are returned. The query must return a single column containing ANALYSIS.Analysis_Id values.

The supplied query must return only those analysis which have exactly one related file stored in the filesystem. (See the specification of the default query below for more information.)

Duplicate ANALYSIS.Analysis_Id values returned by the query are silently ignored.

When this argument is omitted all analysis are selected which have exactly one related file stored in the filesystem -- which have exactly one related ANALYSISPROP row having an ANALYSISPROP.Type_Id value equal to that designated by the --file-cv and --file-term options.

--stock-query=

An (optional) query which selects the stock for which results are returned. The query must return a single column containing STOCK.Stock_Id values.

Duplicate STOCK.Stock_Id values returned by the query are silently ignored.

When this argument is omitted all columns of all selected analysis are returned.

--anonymize

An (optional) option which causes the column headings in the returned files to be replaced with values based on STOCK.Stock_Id. When used, the 2nd and subsequent column headings appearing in the first line of the dumped files are replaced with the STOCK.Stock_Id value, prefaced with the string STK_. of the stock that was analyzed to produce the results contained within the column.

FILE INFORMATION

--nd-experiment-cv=

The name of the set of controlled vocabulary terms used to designate the type of the experimental data -- the CV.Name value identifying the set of CVTERM rows in which the ND_EXPERIMENT.Type_Id values appears.

--nd-experiment-term=

The controlled vocabulary term used to designate the experimental data in the db as information regards columns of data in the previously uploaded files -- the CVTERM.Name values of the ND_EXPERIMENT.Type_Id which represent the columns of the analysis.

--column-cv=

The set of controlled vocabulary terms which designates the types of the columns of the input file -- the CV.Name value identifying the set of CVTERM rows which classify columnar data.

--column-term=

The vocabulary terms describing the type of columnar data . The term is used to associate the input columns with a category. The term must be a CVTERM.Name value in CVTERM rows which belong to the --column-cv cvname controlled vocabulary. The related CVTERM.CVTerm_Id value is placed in the ANALYSIS.ND_EXPERIMENT_PROP.Type_Id column.

--column-number-cv=

The name of the set of controlled vocabulary terms containing the type used to designate column numbers -- the CV.Name value identifying the set of CVTERM rows in which the ND_EXPERIMENT_PROP.Type_Id value appears in the row which stores the column number of the input file.

Defaults to null.

--column-number-term=

The controlled vocabulary term used to designate the column number within the input file -- the CVTERM.Name value of the ND_EXPERIMENT_PROP.Type_Id row where the column number of the input file is stored.

Defaults to column_number.

--file-cv=

The name of the set of controlled vocabulary terms containing the type used to classify the ANALYSISPROP row which stores the pathname of the previously uploaded analysis files -- the CV.Name value identifying the set of CVTERM rows in which the ANALYSISPROP.Type_Id value appears in the row which stores the pathname of the previously uploaded analysis file.

Defaults to null.

--file-term=

The controlled vocabulary term used to designate the pathname of the previously uploaded analysis file -- the CVTERM.Name value of the ANALYSISPROP.Type_Id row where the pathname of the previously uploaded analysis file is stored.

Defaults to pathname.
Examples

Specifying queries as shell command line arguments can be tricky due to quoting issues. The best approach is to use single quotes in the shell and write your sql using the PostgreSQL dollar quoting syntax. This approach uses $$ in SQL statements in place of single quotes.

Example 4.1 Using Dollar Quoting to Get the Results of a Specific Analysis

```shell
chado-funcgen-dump \
  --dbname=babase \
  --analysis-query='SELECT analysis.analysis_id FROM analysis 
                      WHERE analysis.name = $$myfuncgenanalysis$$'
```

The Web Interface

chado-funcgen-dump can also be invoked from a web interface. The web interface provides a method of selecting analysis and stock -- it provides a substitute for the --analysis-query and --stock-query options. The result is always anonymized, as if --anonymize was used.

The following parameters always take the value indicated:

- `--nd-experiment-cv` babase_chado_terms
- `--nd-experiment-term` column_in_analysis
- `--column-cv` babase_chado_terms
- `--column-term` column_in_analysis
- `--column-number-cv` babase_chado_terms
- `--column-number-term` column_number
- `--file--cv` babase_chado_terms
- `--file-term` pathname

In addition, the web interface adds user-selected demographic information to the resultant zip file.

Environment Variables

**CHADO_FUNCGEN_DUMP** Options which are passed to chado-funcgen-dump automatically. This can be used to establish per-user or sitewide defaults.

**PGDATABASE, PGHOST, PGPORT, PGUSER** Default PostgreSQL connection parameters. For further information see the PostgreSQL documentation.

**FILES**

`~/.pgpass` The PostgreSQL per-user password file is consulted to obtain passwords. For further information see the PostgreSQL documentation.
4.1.3 chado-funcgen-load

chado-funcgen-load — load information regarding a file containing functional genomic analysis results, and, optionally, information on the stock analyzed, into Babase/Chado

Synopsis

```
chado-funcgen-load [-h] [-v] [-V] [-dry-run]
[-dbname=dbname] [-host=hostname] [-port=port] [-username=username]
[-analysis-name=name] [-analysis-description=descr]
[-analysis-program=program] [-analysis-programversion=version]
[-analysis-source=sourcename] [-analysis-sourceversion=sourceversion]
[-analysis-algorithm=algorithm]
[-analysis-timeexecuted=timestamp]
[-analysis-geolocation-id=geolocid]
[-analysis-cvterm-cv=cvname --analysis-cvterm-term=termname]
[-stock-file=pathname]
/pathname
```

Description

`chado-funcgen-load` takes the headings in a tab delimited file, `pathname`, and loads information about the file into the database. The information saved includes not only meta-information regarding the origin of the file but also the location of the file and what STOCK were analyzed to produce the file. The data content of the file is not stored, but the information stored can be used to locate data within and among files.

Since the location of the input file is stored in the database a pathname to the file must be supplied on the command line. stdin cannot be used. The supplied `pathname` may be a relative path. It is converted by the `chado-funcgen-load` into a fully qualified pathname before storage into the database.

Optionally, `chado-funcgen-load` can run `chado-stock-load` to ensure that the needed information on the stock provided as input to the functional genomic analysis is in the database and to obtain the stock’s database identifiers. This is done when the stock related options are supplied on the command line instead of the `--stock-id-map` option. For further information see the documentation of the `chado-stock-load` command.

Tip

Supplying `chado-funcgen-load` with a stock file (to pass on to `chado-stock-load`) is usually the easiest way to give `chado-funcgen-load` the necessary stock identifiers. This is true even when the stock information is already in the database.

Note that the fully qualified path name to `pathname` is stored in the database.

The input file is tab-delimited. The first line contains column headings, each of which should map to a row in STOCK. (The first line should begin with a tab, so that the first cell, the cell over the row headings, is empty.) The first cell of the 2nd and all subsequent lines is expected to contain a row heading, something that maps to an analyzed feature of the genome. The remaining cells, those that form the body of the table, contain values representing the result of the analysis, of the given stock preparation of the given portion of the genome.

---

It is not a requirement that the first cell begin with a tab. `chado-funcgen-load` ignores the content of the first cell.
Note
Information on the STOCK, and hence the individuals sampled, is stored. Because information regarding the specifics of what loci were analyzed varies widely per analysis, and because the labeling of such varies widely, is non-standardized among analysis, and is voluminous and consequently labor intensive to map to any sort of standardized naming scheme, this information is not stored in the database.
Consequently, the database tracks only the set of information collected per functional genomic analysis per stock preparation, not per analysis site. Information is not retrievable per genetic sequence, nor per single result value. Information is retrievable per analysis per stock. E.g., per column.

The meta-information about the analysis is stored as a row in the ANALYSIS table (and related property tables, etc.) The resulting ANALYSIS row is therefore the database representation of the analytical results -- of the designated file.
All database modifications take place within a single transaction. Upon error the transaction is rolled back and the database returned to it's original state.
Unless otherwise noted all matching is done in a case-sensitive fashion.

Arguments

⚠️ Caution
CVTERM rows specified by command line arguments are not checked for obsolescence. The CVTERM.Is_Obsolete value is not examined. It is assumed that the user knows whether or not an obsolete term should be used.

The --analysis-cvterm-cv, --analysis-cvterm-term, and --analysis-cvterm-value options may be repeated as many times as desired, so long as each of these options appears on the command line as many times as each of the others. A row is created in ANALYSIS_CVTERM for each occurrence of this set of options.

DATABASE OPTIONS

--dbname=dbname The (optional) name of the database to which to connect or a conninfo connection string. Defaults to the user’s Unix username.

--host=hostname The (optional) host name of the machine on which the server is running. If the value begins with a slash it is used as the directory for the Unix-domain socket.

--port=port The (optional) TCP port or the local Unix-domain socket file extension on which the server is listening for connections.

--username=username (Optional) Username to use to connect to the database. Defaults to the user’s Unix username.

--dry-run When the (optional) --dry-run option is used a trial run is performed that does not permanently insert any data into the database, the transaction is rolled back when the program completes. --dry-run allows an upload to be tested while minimally affecting database content.

ANALYSIS AND ANALYSIS_CVTERM ROW CONSTRUCTION

--analysis-name=name The (optional) ANALYSIS.Name value to use in the ANALYSIS row representing the input file. When not supplied the default value is the file name portion of the file parameter.

--analysis-description=description The (optional) ANALYSIS.Description value to use in the ANALYSIS row representing the input file. When not supplied the ANALYSIS.Description value is set to NULL.

--- A sample conninfo connection string might look like user='scott', host='localhost', dbname='babase_test'.
--- With the notable exclusion of the testing of constraints and triggers that are deferred until transaction commit.
--- Automatically generated ids and other values produced by PostgreSQL sequence generators are consumed when --dry-run is used.
```
--analysis-program=program The ANALYSIS.Program value to use in the ANALYSIS row representing the input file.
--analysis-programversion=version  The ANALYSIS.Programversion value to use in the ANALYSIS row representing the input file.
--analysis-algorithm=algorithm The (optional) ANALYSIS.Algorithm value to use in the ANALYSIS row representing the input file.
When not supplied the ANALYSIS.Algorithm value is set to NULL.
--analysis-sourcename=sourcename The (optional) ANALYSIS.Sourcename value to use in the ANALYSIS row representing the input file.

Note
This argument is unlikely to be used in Babase.

When not supplied the ANALYSIS.Sourcename value is set to NULL.
--analysis-sourceversion=sourceversion The (optional) ANALYSIS.Sourceversion value to use in the ANALYSIS row representing the input file.

Note
This argument is unlikely to be used in Babase.

When not supplied the ANALYSIS.Sourceversion value is set to NULL.
--analysis-sourceuri=sourceuri The (optional) ANALYSIS.SourceURI value to use in the ANALYSIS row representing the input file.

Note
This argument is unlikely to be used in Babase.

When not supplied the ANALYSIS.SourceURI value is set to NULL.
--analysis-timeexecuted=timestamp The ANALYSIS.Timeexecuted value to use in the ANALYSIS row representing the input file. Any PostgreSQL timestamp representation is accepted.
--analysis-cv=cvname The (optional) name of the set of controlled vocabulary terms containing the type of the analysis -- the CV.Name value identifying the set of CVTERM rows in which the ANALYSIS.Type_Id value appears.
--analysis-term=termname The (optional) controlled vocabulary term used to describe the analysis -- the CVTERM.Name value of the ANALYSIS.Type_Id.
--nd-geolocation-id=geolocid The location where the stock originated. A ND_GEOLOCATION.ND_Geolocation_Id value to use for the ND_Geolocation_Id value of the inserted ND_EXPERIMENT rows.

Note
Babase has no use for the ND_EXPERIMENT.ND_Geolocation_Id column and so will likely always use an arbitrary value like 0 here.

--analysis-cvterm-cv=cvname The set of controlled vocabulary terms to use to classify the analysis into types, a form of meta-information on the analysis and the input file -- the CV.Name value identifying the set of CVTERM rows used as ANALYSIS_CVTERM.CVTerm_Id values in those rows created to store the input file and related analysis.

Note
This should really be a vector, so that each stock can come from a different location. But Babase does not care.
```
--analysis-cvterm-term=termname The classification of the analysis or its input file -- the CVTERM.CVTerm_Id value identifying the CVTERM.CVTerm_Id which classifies the analysis or the input file.

--analysis-cvterm-is-not=boolean Whether or not the --analysis-cvterm-term termname type is to be negated -- the ANALYSIS_CVTERM.Is_Not value.

See the paragraph on boolean representation in the command overview above for how to specify boolean values.

STOCK INFORMATION

--stock-id-map=map A (optional) colon separated list of stock ids (STOCK.Stock_Id values). There must be a one-to-one mapping between the supplied stock ids and the data columns; there must be a stock id supplied for each column of data in the input file, and a column of data for each supplied stock id.

If this option is not supplied then the arguments listed in the Stock Upload Arguments section must all be supplied.

--column-cv=cvname The set of controlled vocabulary terms which describe the types of the columns of the input file -- the CV.Name value identifying the set of CVTERM rows which classify columnar data.

--column-term=termname The vocabulary terms describing the data columns. The term is used to associate the input columns with a category. The term must be a CVTERM.Name value in CVTERM rows which belong to the --column-cv cvname controlled vocabulary. The related CVTERM.CVTerm_Id value is placed in the ANALYSIS_ND_EXPERIMENT.Type_Id column.

ND_EXPERIMENT INFORMATION

Note
The information supplied by these arguments are probably not particularly useful in Babase.

--nd-experiment-cv=cvname The name of the set of controlled vocabulary terms containing the type of the experiment -- the CV.Name value identifying the set of CVTERM rows in which the ND_EXPERIMENT.Type_Id values appears.

--nd-experiment-term=termname The controlled vocabulary term used to describe the experiment -- the CVTERM.Name values of the ND_EXPERIMENT.Type_Id which represent the columns of the analysis.

ND_EXPERIMENT_STOCK INFORMATION

Note
The information supplied by these arguments are almost surely not useful in Babase.

--nd-experiment-stock-cv=cvname The name of the set of controlled vocabulary terms containing the type of relationship between the experiment and the stock -- the CV.Name value identifying the set of CVTERM rows in which the ND_EXPERIMENT_STOCK.Type_Id values appears.

--nd-experiment-stock-term=termname The controlled vocabulary term used to describe the relationship between the experiment and the stock -- the CVTERM.Name values of the ND_EXPERIMENT_STOCK.Type_Id which represent the relationship between the columns of the analysis and its stock.

FILE INFORMATION

--file-cv=cvname The name of the set of controlled vocabulary terms containing the type used to classify the ANALYSISPROP row which stores the pathname of the input file -- the CV.Name value identifying the set of CVTERM rows in which the ANALYSISPROP.Type_Id value appears in the row which stores the pathname of the input file.

Defaults to null.
--file-term=termname The controlled vocabulary term used to designate the path name of the input file -- the CVTERM.Name value of the ANALYSISPROP.Type_Id row where the path name of the input file is stored.
Defaults to pathname.

--column-number-cv=cvname The name of the set of controlled vocabulary terms containing the type used to designate column numbers -- the CV.Name value identifying the set of CVTERM rows in which the ND_EXPERIMENTPROP.Type_Id value appears in the row which stores the column number of the input file.
Defaults to null.

--column-number-term=termname The controlled vocabulary term used to designate the column number within the input file -- the CVTERM.Name value of the ND_EXPERIMENTPROP.Type_Id row where the column number of the input file is stored.
Defaults to column_number.

STOCK UPLOAD ARGUMENTS

--organism-id=oid, --pub-id=pubid, --primary-db=name, --allow-duplicates, --stock-file=pathname
The --stock-file causes chado-funcgen-load to invoke chado-stock-load to load stock information found in the specified --stock-file pathname file. --allow-duplicates is the only optional argument, when --stock-file is used.
For further information see the documentation of the chado-stock-load command.
Either the --stock-file option or the --stock-id-map option must be used.
The stock upload arguments are silently ignored unless --stock-file is used. This allows defaults to be established for them in the CHADO_FUNCGEN_LOAD environment variable.

The ANALYSIS and ANALYSISPROP information

In addition to the values supplied by the command line options beginning with --analysis- the fully qualified path name of the input file is stored in an ANALYSISPROP row.
If the --analysis-cv and --analysis-term arguments are not supplied then the ANALYSIS.Type_Id value is NULL.
The ANALYSIS.Analysis_Id value is automatically generated by PostgreSQL.
Analysis may be typed (or, equivalently, tagged) by repeated use of the --analysis-cvterm-cv, --analysis-cvterm-term, and --analysis-cvterm-is-not options. These result in creation of ANALYSIS_CVTERM rows.
An ANALYSISPROP row is created to store the fully qualified path name of the input file. The cvname value supplied by --file-cv and the termname value supplied by --file-term are used to look up the CVTERM.CVTerm_Id value for the row storing the input path name.

Processing The Input File

The processing of the input file consists solely of matching the input file’s data columns with the command line options and storing the result. For each data column of the input file the following occurs:

• An ND_EXPERIMENT row is created to represent the input file column. The row is created as follows:
    THE ND_EXPERIMENT ROW

    ND_Experiment_Id The next available automatically generated id.

    ND_Geolocation_Id The supplied --nd-geolocation-id geolocid value.

    Type_Id The supplied --nd-experiment-cv cvname and --nd-experiment-term termname values are used to look up the CVTERM.CVTerm_Id used for this value.
• An **ND_EXPERIMENT_STOCK** row is created to relate the stock used as input for the column with the column -- to relate the **STOCK** with the **ND_EXPERIMENT**. This is done as follows:

  **THE ND_EXPERIMENT_STOCK ROW**

  **ND_Experiment_Stock_Id**  The next available automatically generated id.

  **ND_Experiment_Id**  The **ND_EXPERIMENT.ND_Experiment_Id** value.

  **Stock_Id**  The **STOCK.Stock_Id**, as supplied by the **--stock-id-map** map element which corresponds to the column of the input file.

  **Type_Id**  The supplied **--nd-experiment-stock-cv cvname and --nd-experiment-stock-term termname** values are used to look up the **CVTERM.CVTerm_Id** used for this value.

• An **ANALYSIS_ND_EXPERIMENT** row is created to relate the analysis (and hence the input file) with the column -- to relate the **ANALYSIS** with the **ND_EXPERIMENT**. This is done as follows:

  **THE ANALYSIS_ND_EXPERIMENT ROW**

  **Analysis_ND_Experiment_Id**  The next available automatically generated id.

  **Analysis_Id**  The **ANALYSIS.Analysis_Id** of the analysis.

  **ND_Experiment_Id**  The **ND_EXPERIMENT.ND_Experiment_Id** value.

  **Type_Id**  The **CVTERM.CVTerm_Id** value determined by the **--column-cv cvname and --column-term typename**.

• A **ND_EXPERIMENTPROP** row is created to store the column number. This is done as follows:

  **THE ND_EXPERIMENTPROP ROW**

  **ND_Experimentprop_Id**  The next available automatically generated id.

  **ND_Experiment_Id**  The **ND_EXPERIMENT.ND_Experiment_Id** value representing the column of the input file.

  **Type_Id**  The **CVTERM.CVTerm_Id** value used to indicate that the value is a column number in a functional genomic analysis input file, as specified by the **--column-number-cv cv and --column-number-term typename**.

  **Value**  The number of the input file’s column.

  **Rank**  0

### Environment Variables

**CHADO_FUNCGEN_LOAD**  Options which are passed to **chado-funcgen-load** automatically. This can be used to establish per-user or sitewide defaults.

**PGDATABASE, PGHOST, PGPORT, PGUSER**  Default PostgreSQL connection parameters. For further information see the PostgreSQL documentation.

### FILES

```
~/.pgpass  The PostgreSQL per-user password file is consulted to obtain passwords. For further information see the PostgreSQL documentation.
```

### 4.1.4 chado-stock-load

chado-stock-load — load stock information into Babase/Chado
Synopsis

[--dbname=dbname] [--host=hostname] [--port=port] [--username=username]
[--dry-run]
--organism-id=oid [--pub-id=pubid] --primary-db=name
[--allow-duplicates]
[pathname]

Description

chado-stock-load takes information in a tab delimited file, pathname, and loads the information into the STOCK table and related tables if the information is not already in the database. If pathname is omitted then data is taken from standard input.

chado-stock-load outputs a list of the STOCK.Stock_Id values that are in the supplied file.

The input file is tab-delimited. The first two lines contain column headings. The first line of heading describes how the data in the column should be stored in the database. The second line of heading is ignored. The 3rd and subsequent lines contain stock information, one line per stock. The first 3 columns are reserved for specific data and must be present, the remaining columns are optional and their column headings determine how they are stored in the database.

The STOCK rows loaded into the database have as their primary (STOCK.DBXref_Id) DBXREF the individual from which the stock came. They always have a secondary (a related STOCK_DBXREF row) to relate the stock back to a physical sample.

The output of chado-load-stock is suitable for use as the --stock-map value of the chado-funcgen-load command.

All database modifications take place within a single transaction. Upon error the transaction is rolled back and the database returned to it’s original state.

Unless otherwise noted all matching is done in a case-sensitive fashion.

Leading and trailing spaces are silently removed from all data inserted into or compared with values in the STOCKPROP.Value column.

Arguments

DATABASE OPTIONS

--dbname=dbname The (optional) name of the database to which to connect or a conninfo connection string. Defaults to the user’s Unix username.

--host=hostname The (optional) host name of the machine on which the server is running. If the value begins with a slash it is used as the directory for the Unix-domain socket.

--port=port The (optional) TCP port or the local Unix-domain socket file extension on which the server is listening for connections.

--username=username (Optional) Username to use to connect to the database. Defaults to the user’s Unix username.

--dry-run When the (optional) --dry-run option is used a trial run is performed that does not permanently insert any data into the database, the transaction is rolled back when the program completes. --dry-run allows an upload to be tested while minimally affecting database content.

DATA OPTIONS

--organism-id=oid The ORGANISM.Organism_Id value to place in STOCK.Organism_Id.

--- A sample conninfo connection string might look like user=’scott’, host=’localhost’, dbname=’babase_test’.
--- With the notable exclusion of the testing of constraints and triggers that are deferred until transaction commit.
--- Automatically generated ids and other values produced by PostgreSQL sequence generators are consumed when --dry-run is used.
--pub-id=pubid The (optional) PUBPub_Id value to place in the STOCK_CVTERM.CVTerm_Id column of new STOCK_CVTERM rows.

This option is only optional if the input file contains no data which will create rows in STOCK_CVTERM.

--primary-db=name The DB.Name of the database that contains the individuals from which the stock were taken. This is the DB related to the STOCK.DBXref_Id values.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Babase the name will almost always be unique_baboons.</td>
</tr>
</tbody>
</table>

**Processing Options**

--allow-duplicates (Optional) When omitted and the first two columns designate a DBXREF which is already related to a STOCK row via STOCK.DBXref_Id, then all remaining columns must contain values which match with the values in the database which are already related to the existing STOCK row. If present, and a STOCK row exists which has a STOCK.DBXref_Id value which is designated by the value of the first 2 columns, then chado-stock-load will create a new row on STOCK if (and only if) the remaining columns fail to match the data related to the STOCK row which already exists.

Matching is done only on the data in the columns which exist in the supplied file. The database may contain additional information related to the stock and that will remain unexamined if there are no columns in the supplied file which contain such additional information.

<table>
<thead>
<tr>
<th>Caution</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is never advisable to use this option until after running the program without the option and verifying that it is appropriate that the database contain “duplicate” STOCK rows.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Once this option has been used and there is more than one row in STOCK with matching STOCK.Type_Id values and related matching DBXREF.Accession values chado-stock-load will always report the duplicates and --allow-duplicates will have to be used in future when working with these stocks. In the case where duplicates already exist in STOCK chado-stock-load will report when one of the duplicates is an exact match of the uploaded stock line and when the uploaded stock line matches none of the existing duplicates.</td>
</tr>
</tbody>
</table>

**Processing The Input File**

Beginning with the third row, each row of the input file represents a single stock.

The first column is required to contain DB.Name values. These values indicate the external database in which the stock is tracked. The first two rows of the first column are ignored.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Babase the first column will usually contain did to indicate that the stock is from a did.</td>
</tr>
</tbody>
</table>

The second column is required to contain DBXREF.Accession values. These values designate the id number of the stock in the external database. The first two rows of the first column are ignored.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Babase the second column will usually contain did values.</td>
</tr>
</tbody>
</table>
Note
It is the first two columns that are used to identify pre-existing STOCK rows and cause them to be considered as possible matches for the uploaded line. If the entire line matches what is in the database the existing STOCK.Stock_Id is reported and a new STOCK row is not created. If the first two columns match but the remaining columns do not then the use of the --allow-duplicates option determines what happens. See the documentation above on --allow-duplicates for more information.

The third column is required to contain DBXREF.Accession values which are in the DB designated by the --primary-db name.

Note
In Babase the third column designates the individual baboon from which the sample was taken to create the stock.

Caution
In the general case, the system does not validate the 3rd column, typically the unique id value, against the first two columns. It is possible to associate a stock with the wrong baboon. However, in the case where the the first column is did the chado-stock-load command itself will check that the value of the 3rd column matches that of the GENETIC_INVENTORY.DNA.Sname column of the GENETIC_INVENTORY.DNA row where the GENETIC_INVENTORY.DNA.DId column's value is the value in column 2. The match is performed as follows: If the value of the 3rd column is the empty string or NULL the values match. If the value of GENETIC_INVENTORY.DNA.Sname is NULL the values match. Otherwise, the two values are compared in a case-sensitive fashion.

The headings of the fourth and subsequent columns, appearing on the first and 2nd line, have the following structure:

The first line is either column_number, or begins with one of type:, or primarytype:, or prop:.

The file must contain exactly one column having a column heading in the first line of column_number. The data in this column records which column in the functional genetic data is associated with the stock in the uploaded file. Since all (but the first) column in a file of functional genomic results must represent the analysis of a stock, each row in the uploaded stock data must contain a unique column number. Further, the uploaded column numbers must, when considered as a collection, have 2 as their lowest value and each successive integer must be contained in the collection, without omission, until the maximal value is reached. The stock rows in the uploaded file need not be ordered by the column number.

The file must contain exactly one column having a column heading in the first line with a prefix of primarytype: When the prefix is primarytype: the following holds true: The column contains CVTERM.Name values and the related CVTERM.CVTerm_Id value is stored in the STOCK.Type_Id column. In the first line of the header primarytype: is followed by the CV.Name of the vocabulary used in the column.

Note
This column is required because the STOCK.Type_Id cannot be NULL. In Babase it is not likely that primarytype: column heading will often read other than primarytype:babase_chado_terms and the column's data values be other than None.

Only one column with a prefix of primarytype: may appear in the input file. If no columns have this prefix then the STOCK.Type_Id is NULL.

When the prefix is type: the following holds true: The column contains CVTERM.Name values and a row is created in STOCK_CVTERM for each data row in the input file. In the first line of the header type: is followed by the CV.Name of the vocabulary used in the column.

When the prefix is prop: the following holds true: A row is created in the STOCKPROP table for each data row in the input file. In the first line of the header the prop: is followed by the CV.Name of the vocabulary used to designate the data values, followed by a colon, followed by the CVTERM.Name used to designate the data value. As rows are inserted into STOCKPROP,
the related CVTERM.CVTerm_Id is used as the STOCKPROP.Type_Id and the value appearing in the column becomes the STOCKPROP.Value.

The value NULL has special meaning to chado-stock-load when appearing in a column which contains STOCKPROP.Value values. It indicates that the PostgreSQL NULL value should be stored in the database.

---

**Note**

There is no way to prevent the creation of either a STOCK_CVTERM row or STOCKPROP row when a STOCK row is created, if there are columns in the input file beyond the first 3. In other words, there is no special value that can be used in the input file to prevent the creation of a STOCK_CVTERM or STOCKPROP row.

---

The details of the creation follows:

- A STOCK row is created for each data row in the input file. The row is created as follows:

  **THE STOCK ROW**

  Stock_Id  The next available automatically generated id.

  DBXref_Id  The DBXREF.DBXref_Id found by lookup of the value in the 3rd column in DBXREF where the DB_Id value is that of the DB with a Name of the *--primary-db name*, and the Accession is the value in the 3rd column.

  Organism_Id  The supplied *--organism-id* value.

  Name  NULL

  Uniquename  The same value as the STOCK.Stock_Id, as text.

  Description  NULL

  Type_Id  The CVTERM.CVTerm_Id value found in the data column headed with a primarytype: prefix as described above, or NULL if there is no such column in the input file.

  IsObsolete  FALSE

- An STOCK_DBXREF row is created to relate the stock to the physical sample from which it was derived. This is done as follows:

  **THE STOCK_DBXREF ROW**

  Stock_DBXref_Id  The next available automatically generated id.

  Stock_Id  The STOCK.Stock_Id of the newly created STOCK row.

  DBXref_Id  The DBXREF.DBXref_Id found by lookup of the DB.Name in the data row’s first column and the DBXREF.Accession in the data row’s second column. It is an error if there is no DBXREF row with a DB_Id related to the DB.Name found in the first data column that has an Accession found in the second data column.

  IsCurrent  TRUE

- A STOCK_CVTERM row is created for each column with a type: prefix in the first header row. This is done as follows:

  **THE STOCK_CVTERM ROW**

  Stock_CVTerm_Id  The next available automatically generated id.

  Stock_id  The STOCK.Stock_Id of the newly created STOCK row.

  CVTerm_Id  The CVTERM.CVTerm_Id value determined by matching the input column value with the CVTERM.Name and likewise matching the CV designated in the column heading as described above.

  Pub_Id  The supplied *--pub-id* value.

  IsNot  FALSE

  Rank  0

- A STOCKPROP row is created for each column with a prop: prefix in the first header row. This is done as follows:

  **THE STOCKPROP ROW**
**Stockprop_Id** The next available automatically generated id.

**Stock_Id** The STOCK.Stock_Id value representing the row of the input file.

**Type_Id** The CVTERM.CVTerm_Id value derived from the column heading as described above.

**Value** The value in the cell of the input file.

**Rank** 0

---

**Environment Variables**

**CHADO_STOCK_LOAD** Options which are passed to `chado-stock-load` automatically. This can be used to establish per-user or sitewide defaults.

**PGDATABASE, PGHOST, PGPORT, PGUSER** Default PostgreSQL connection parameters. For further information see the PostgreSQL documentation.

**FILES**

`~/.pgpass` The PostgreSQL per-user password file is consulted to obtain passwords. For further information see the PostgreSQL documentation.

---

**4.1.5 chado-vcf-load**

chado-vcf-load — load a VCF file into Babase/Chado

**Synopsis**

```
chado-vcf-load [-h] [-v] [--version]
               [--analysis-name=name] [--analysis-description=descr]
               [--analysis-program=program] [--analysis-programversion=version]
               [--analysis-source-name=sourcename] [--analysis-sourceversion=sourceversion]
               [--analysis-sourceuri=sourceuri] [--analysis-timestamp=timestamp]
               file
```

**Environment Variables**

**CHADO_STOCK_LOAD** Options which are passed to `chado-stock-load` automatically. This can be used to establish per-user or sitewide defaults.

**PGDATABASE, PGHOST, PGPORT, PGUSER** Default PostgreSQL connection parameters. For further information see the PostgreSQL documentation.

**FILES**

`~/.pgpass` The PostgreSQL per-user password file is consulted to obtain passwords. For further information see the PostgreSQL documentation.

---

chado-vcf-load could increment STOCKPROP.Rank for those cases where the input file contains more than one column which creates properties and uses the same CV and CVTERM. At the time of this writing there are no such input files.
**Description**

`chado-vcf-load` takes the data in the VCF file *file* and loads it into the database.

---

**Caution**

The user must take care when running `chado-vcf-load` and typically should avoid running this program concurrently with other processes which alter the `FEATURE` or `FEATURELOC` tables. For further information see the documentation of the `--transaction-isolation-level` option.

---

`chado-vcf-load` is designed to load into Chado VCF files which encode SNV analysis, although it may be possible to use it to load VCF files which represent other sorts of analyses. Consequently, `chado-vcf-load` handles files conforming to only a subset of the VCF 4.1 specification. For further information see: the section called “Assumptions and Limitations”

---

**Note**

This document uses the term *source feature* to denote the VCF file `#CHROM` value. This value denotes the genomic feature used as the coordinate system origin within the VCF data line. This “Chado-esque” term is used since there seems to be no vocabulary defined for this in the VCF specification.

Any Chado feature can be used as a source feature, although the typical source feature is a chromosome.

---

There are 2 general categories of information loaded into the database. The first is meta-information about the analysis as a whole. While this includes the meta-information present at the top of the VCF file much of the information regarding the overall analysis which produced the VCF file is supplied to the program via command line arguments.

The meta-information about the analysis is stored as a row in the `ANALYSIS` table (and related property tables, etc.) The resulting `ANALYSIS` row is therefore the database representation of the analytical results -- of the loaded VCF file.

The other category of information is, of course, the genomic data supplied in the VCF file. Command line arguments control not only how the program interprets the data encountered, but also, as with analysis meta-information, supply data values incorporated into the Chado representation of the per-VCF-line data.

The data lines of a VCF file contain, in the REF column, the base pairs found in the reference genome at the location specified by the `CHROM#` and `POS` columns. This sequence information is referred to herein as the *reference base feature*. Like the source feature, the reference base feature is stored in Chado\(^\text{12}\) as a row in the `FEATURE` table. Unlike the source feature, the reference base feature may or may not already exist in the database, it is shared between all analysis of the reference genome at the locus. If it does not exist then the program creates it.

The remaining information in each data line of the VCF file is denoted the genotype data. Each column of each line, after the initial “fixed” columns, consists of a cell of genotype analysis results. The analyzed genomic feature is represented by a row in `FEATURE`. This feature row is shared between all analysis of the genotype at the locus. This *genotype feature* is located within the genome relative to the `CHROM#` source feature, to record it’s relationship with the reference genome. The genotype feature is also related to the reference base feature via `FEATURE_RELATIONSHIP`, to note that the two features represent the same locations on different genotypes. The analytic results regarding the genotype feature is stored in `ANALYSISFEATURE` rows (and related `ANALYSISFEATUREPROP` rows, etc.) Each analytical result presented in a VCF data line is therefore linked to the analysis as a whole (`ANALYSIS`), the `CHROM#` source feature, and to the reference base feature (`FEATURE`).\(^\text{13}\)

It is worth noting that the VCF file’s GQ values are handled differently than the other genotype keyword data. The `ANALYSISFEATURE.Normscore` column is designed to hold GQ-like values, and so this column is used to store GQ values. The other VCF keyword values (GT, PL, and the like) are stored in `ANALYSISFEATUREPROP` rows.

When genotype values, such as PL, contain sub-parts delimited by commas then each sub-part is stored in a separate `ANALYSISFEATUREPROP` row. The first sub-part, the data appearing before first comma, is stored in an `ANALYSISFEATUREPROP` row having a `Rank` value of 1. The second sub-part, the data appearing between the first and second comma, is stored in an `ANALYSISFEATUREPROP` row having a `Rank` value of 2, and so forth.

\(^{12}\) The reference base feature is stored this way in Babase’s use of Chado. This is not the traditional approach, which is to store the reference base feature as a row in the `FEATURELOC` table.

\(^{13}\) The ability to store an arbitrary amount of information related to the analysis of a particular reference base feature (SNP, etc.) is a major advantage of storing SNPs as rows in the `FEATURE` table rather than, as is typical, as rows in the `FEATURELOC` table.
When genotype values, such as GT, do not contain comma delimited data the entire value is stored in a single ANALYSISFEATUREPROP row.

The DBXREF table is relied on extensively to represent the analyzed genotypes.

**Note**

In Babase DBXREF rows are used to represent individual baboons. It is worth noting that analyzed features on the reference genome is related to the “reference baboon” as if the reference baboon had the reference genome sequence.

All database modifications take place within a single transaction. Upon error the transaction is rolled back and the database returned to it’s original state.

Unless otherwise noted all matching is done in a case-sensitive fashion.

**Caution**

To limit the amount of virtual memory consumed during it’s execution, because it loads a large amount of data in a single transaction⁴, the chado-vcf-load changes all deferred constraints to immediate constraints within the transaction that it creates. This has the potential to interfere with data integrity checks that are implemented as deferred constraint triggers, although no such triggers exist at this time within Chado.

⁴ And because Chado defers all foreign key constraints.

**Caution**

chado-vcf-load assumes that the PostgreSQL permission system prevents database users from making unauthorized changes. chado-vcf-load is not secure against SQL injection attacks.⁴ Consequently users may be able to execute arbitrary SQL statements. This is a problem only if the PostgreSQL credentials used to authorize a user’s database connection grant privileges that the user should not have.

⁴ Specifically, the –sample-id-trans-table, –sample-id-trans-keycol, and –sample-id-trans-valuecol options are all vulnerable to SQL injection.

**Arguments**

**Caution**

CVTERM rows specified by command line arguments are not checked for obsolescence. The CVTERM.IsObsolete value is not examined. It is assumed that the user knows whether or not an obsolete term should be used.

**DATABASE OPTIONS**

--dbname=dbname The (optional) name of the database to which to connect or a conninfo connection string.¹⁴ Defaults to the user’s Unix username.

--host=hostname The (optional) host name of the machine on which the server is running. If the value begins with a slash it is used as the directory for the Unix-domain socket.

--port=port The (optional) TCP port or the local Unix-domain socket file extension on which the server is listening for connections.

--username=username (Optional) Username to use to connect to the database. Defaults to the user’s Unix username.

¹⁴ A sample conninfo connection string might look like user=’scott’, host=’localhost’, dbname=’babase_test’.
**--do-pg-analyze** (Optional) Run the PostgreSQL ANALYZE command periodically during data load.

For the most part this option should not be used. However, if there is no existing data in the database or if the amount of data that does exist in the database is minimal compared to the amount of data to be loaded into the database then the query planner may produce non-optimal queries. In this case it can be useful to have the database analyze database content during the data loading process.

---

**Caution**
The **--do-pg-analyze** option will silently fail, doing nothing, if the database user (see **--username** does not have permissions to run **ANALYZE** on the tables into which this program inserts.

---

**Note**
The **repeatable_read** transaction isolation level (see **--transaction-isolation-level**), or a more restrictive level, could be called for when using **--do-pg-analyze**.\(^a\)

---

\(^a\) Otherwise the auto-vacuuming and consequent analysis could produce poor statistics that would be seen by the **chado-vcf-load** process. However in practice this is unlikely to happen. It is probable that the **chado-vcf-load** would acquire locks that would prevent auto-vacuuming.

**--transaction-isolation-level=level** (Optional) The transaction isolation level at which to run the transaction.

The available choices are:

- **read_committed**
  This is the default.
- **repeatable_read**
- **serializable**
- **database_default**
  The default transaction isolation level for the database.

---

**Caution**
Changing the transaction isolation level can have adverse effects.

---

The safest isolation level is **serializable** however at this level the system holds locks for the duration of the transaction.\(^b\) For long running transactions this can result in depletion of the storage available for predicate locks.

---

**Caution**
At the **repeatable_read** transaction isolation level it is possible, although unlikely, that concurrent **chado-vcf-load** commands will create duplicate feature rows.\(^a\) This can also happen, with an even lower probability, at the **read_committed** transaction isolation level.

This is true in general when running **chado-vcf-load** at the same time as any other transaction which alters the content of the **FEATURE** or **FEATURELOC** tables, duplicate features can be created in the database. The **serializable** level checks for and prevents this, but at a cost. The **repeatable_read** isolation level is less likely to produce duplicates when run concurrently with a transaction which updates existing rows whereas the **read_committed** isolation level is less likely to produce duplicates when run concurrently with a transaction which inserts new rows.

The user must take care when running **chado-vcf-load**. It is best to avoid running this program concurrently with other processes which alter the **FEATURE** or **FEATURELOC** tables.\(^b\)

---

\(^a\) A duplicate feature row would be a second row which represents the same thing as a feature already in the database. This could be either a sequence of bases on the reference base genome or a feature which represents a analyzed region of a specific genotype.

\(^b\) At least without assurance that the two different processes will not both alter or attempt to examine a representation of any given feature.

---

\(^{15} \text{See http://git.postgresql.org/gitweb/?p=postgresql.git;a=blb;f=src/backend/storage/lmgr/README-SSI;hb=master or http://vldb.org/pvldb/vol5/p1850_danrkports_vldb2012.pdf.}\)
Note
It is safe\(^a\) to query the FEATURE and FEATURELOC tables while the chado-vcf-load program is executing.\(^b\)

\(^a\) At least in Babase, which has the default transaction isolation level set to serializable.
\(^b\) In Babase, the results of the chado-vcf-load will not be visible until after the successful completion of the command.

--dry-run When the (optional) --dry-run option is used a trial run is performed that does not permanently insert any data into the database, the transaction is rolled back when the program completes. --dry-run allows an upload to be tested\(^{16}\) while minimally affecting database content.\(^{17}\)

ANALYSIS AND ANALYSISPROP ROW CONSTRUCTION

--analysis-name=name The (optional) ANALYSIS.Name value to use in the ANALYSIS row representing the VCF file. When not supplied the default value is the file name portion of the file parameter, with any .vcf suffix (case in-sensitive) removed.

--analysis-description=description The (optional) ANALYSIS.Description value to use in the ANALYSIS row representing the VCF file. When not supplied the ANALYSIS.Description value is set to NULL.

--analysis-program=program The ANALYSIS.Program value to use in the ANALYSIS row representing the VCF file.

--analysis-programversion=version The ANALYSIS.Programversion value to use in the ANALYSIS row representing the VCF file.

--analysis-algorithm=algorithm The (optional) ANALYSIS.Algorithm value to use in the ANALYSIS row representing the VCF file. When not supplied the ANALYSIS.Algorithm value is set to NULL.

--analysis-sourcename=sourcename The (optional) ANALYSIS.Sourcename value to use in the ANALYSIS row representing the VCF file. Note
This argument is unlikely to be used in Babase.

When not supplied the ANALYSIS.Sourcename value is set to NULL.

--analysis-sourceversion=sourceversion The (optional) ANALYSIS.Sourceversion value to use in the ANALYSIS row representing the VCF file. Note
This argument is unlikely to be used in Babase.

When not supplied the ANALYSIS.Sourceversion value is set to NULL.

--analysis-sourceuri=sourceuri The (optional) ANALYSIS.SourceURI value to use in the ANALYSIS row representing the VCF file. Note
This argument is unlikely to be used in Babase.

\(^{16}\) With the notable exclusion of the testing of constraints and triggers that are deferred until transaction commit. But see above as regards whether any of these exist.
\(^{17}\) Automatically generated ids and other values produced by PostgreSQL sequence generators are consumed when --dry-run is used.
When not supplied the `ANALYSIS.SourceURI` value is set to `NULL`.

`--analysis-timeexecuted=timestamp` The `ANALYSIS.Timeexecuted` value to use in the `ANALYSIS` row representing the VCF file. Any PostgreSQL timestamp representation is accepted.

`--analysis-cv=cvname` The (optional) name of the set of controlled vocabulary terms containing the type of the analysis -- the `CV.Name` value identifying the set of `CVTERM` rows in which the `ANALYSIS.Type_Id` value appears.

`--analysis-term=termname` The (optional) controlled vocabulary term used to describe the analysis -- the `CVTERM.Name` value of the `ANALYSIS.Type_Id`.

`--analysisprop-cv=cvname` The set of controlled vocabulary terms to use when storing the VCF file’s meta-information -- the `CV.Name` value identifying the set of `CVTERM` rows used as `ANALYSISPROP.Type_Id` values in those rows created to store the VCF file’s meta-information.

**Source Feature Determination**

`--source-feature-cv=cvname` The (optional) set of controlled vocabulary terms in which the VCF data lines’ source feature appears, i.e. the collection of terms that comprise the available list of genomic features -- the `CV.Name` value identifying the set of `CVTERM` rows used to find the source feature identified by the `#CHROM` value of each VCF data line.

The default is `sequence`, the Sequence Ontology.

`--source-feature-term=termname` The (optional) controlled vocabulary term describing the source feature which each data line of the VCF file uses as a coordinate system origin -- the `CVTERM.Name` value used to find the source feature identified by the `#CHROM` value of each VCF data line.

The default is `chromosome`.

`--source-feature-relationship-cv=cvname` The (optional) set of controlled vocabulary terms which encodes the relationships between the terms in the `--source-feature-cv cvname cv`. See the `CVTERM_RELATIONSHIP` documentation for more information.

The default is `sequence`.

`--source-feature-relationship-term=termname` The (optional) controlled vocabulary term in the `--source-feature-relationship-cv cvname` that denotes the relationship between the `--source-feature-term termname` supplied and the `CVTERM.CVTerm_Id` related to the source feature identified by the `#CHROM` value of each VCF data line. For further information see: the section called “Identifying the `#CHROM` source feature”.

The default is `is_a`.

`--source-feature-db=dbname` The (optional) collection of alternate feature names for the source feature used in the VCF file’s data lines -- the `DB.Name` value used when finding the source feature identified by the `#CHROM` value of each VCF data line.

The default is to use all databases.

**Reference Base Feature Typing**

`--reference-base-feature-cv=cvname` The (optional) set of controlled vocabulary terms in which the VCF data lines’ reference base feature appears, e.g. the collection of terms that comprise the available list of genomic features -- the `CV.Name` value identifying the set of `CVTERM` rows used to find or create the reference base feature identified by the POS and REF values of each VCF data line.

The default is `sequence`, the Sequence Ontology.

---

18 In theory, it might be better if `--source-feature-relationship-term` had no default value. All relationships between `CVTERMs` are “is_a” relationships, only some, such as “part_of” are more explicit. Having a default means that the `CVTERM_RELATIONSHIP.Type_Id` value is always compared against something specific. This means it’s impossible to match every sort of “is_a” relationship, as might be possible if there was no `--source-feature-relationship-term` at all -- instead of a default.

In practice, it’s simpler to have a default and we don’t have a need to match every sort of “is_a” relationships at once.
--reference-base-feature-term=termname The controlled vocabulary term, in the --reference-base-feature-cv cvname ontology, describing the reference base feature of each data line in the VCF file -- the CVTERM.Name value used to find or create the reference base feature identified by the POS and REF values of each VCF data line.

**Note**
For SNVs, as is typical in Babase, the *termname* should be `base`.

---

**GENOME ROW CONSTRUCTION**

--feature-db-name=dbname The name of the external database in which are looked up the VCF data line’s non-“fixed” column headings (columns 10 and above, inclusive), for the purposes of identifying the source of the analyzed genome.

**Note**
In Babase the *dbname* will typically be `unique_baboons`.

--reference-base-feature-dbxfref-accession=accession The (optional) DBXREF.Accession value used to provide the reference base feature with a FEATURE.DBXref_Id value. If this argument is not supplied then the reference base feature will be given a NULL DBXref_Id value.

**Note**
In Babase the *accession* value will typically be `reference_baboon`, so that the reference base feature is related to the reference baboon.

--sample-id-ignore-case=boolean (Optional) Whether or not to ignore character case when comparing the non-“fixed” column headings in the VCF header lines, the column headings which identify the genomic samples. See the paragraph on boolean representation in the command overview above for how to specify boolean values. The default is True, to be case-insensitive.

--sample-id-trans-table=table The (optional) table in which to look up the VCF file’s non-“fixed” column headings to obtain DBXREF.Accession values which are related to the the --feature-db-name’s *dbname*. The *table* may be schema qualified.

When this argument is not supplied the non-“fixed” column headings must themselves be such DBXREF.Accession values.

--sample-id-trans-keycol=keycol The (optional) column name of the --sample-id-trans-table *table* in which the VCF file header line’s sample ids are looked up.

--sample-id-trans-valuecol=valuecol The (optional) column name of the --sample-id-trans-table *table* which contains DBXREF.Accession values.

--alternate-base-term=termname The (optional) controlled vocabulary term, in the --reference-base-feature-cv cvname ontology, describing the alternate base sequence found in each data line in the VCF file -- the CVTERM.Name value used to classify the VCF line’s ALT value. The default value is `polymorphic_sequence_variant`.

**Note**
Babase users will typically use SNP.

--feature-relationship-cv=cvname The (optional) set of controlled vocabulary terms in which the term describing the relationship to establish between the genotype and the reference base feature appears, e.g. the collection of terms that comprise the available list of relationships between features -- the CV.Name value identifying the set of CVTERM rows used to define relationships between features.

The default is `relationship`, the OBO Relationship Ontology.

19 In the Sequence Ontology `polymorphic_sequence_variant` has the identifier SO:0001025.
**--reference-base-feature-relationship-term=**<termname> The (optional) controlled vocabulary term, in the–feature-relationship-cv cvname ontology, describing the relationship to establish between the reference base feature and the source feature -- the CVTERM.Name value used in the FEATURE_RELATIONSHIP.Type_Id column.
The default value is part_of.

**--genotype-reference-relationship-term=**<termname> The (optional) controlled vocabulary term, in the--feature-relationship-cv cvname ontology, describing the relationship to establish between the genotype feature and the reference base feature -- the CVTERM.Name value used in the FEATURE_RELATIONSHIP.Type_Id column.
The default value is instance_of.

**--genotype-feature-cv=**<cvname> The (optional) set of controlled vocabulary terms in which the VCF data lines’ genotype feature keyword values appear, i.e. the collection of terms that comprise the available list of genotype field keywords (per section 1.4.2 of the VCF specification version 4.2) -- the CV.Name value identifying the set of CVTERM rows used to designate the different genotype feature keyword values, PL, GT, and the like, present in the cells of each VCF data line.
This option is only optional if the VCF file does not contain any genotype analysis results to be loaded into ANALYSISFEATUREPROP rows.

**--genotype-feature-keywords=**<keywords> The (optional) colon delimited list of genotype feature keywords to be stored in ANALYSISFEATUREPROP rows. If this option is omitted or the keywords consist of the empty string then no genotype data in the VCF file will be stored in ANALYSISFEATUREPROP rows.

**Note**
The order of the supplied VCF keywords does not have to correspond to the order in which the keywords appear in the VCF file data.

**Note**
In Babase the keywords value will typically be PL:GT.

**--genotype-feature-terms=**<termnames> The (optional) colon delimited list of controlled vocabulary terms corresponding to the genotype feature keywords supplied via the --genotype-feature-keyword <keywords>. The supplied terms must be in controlled vocabulary identified by the --genotype-feature-cv <cvname>. Each of the terms identifies the CVTERM.Name value used to classify the respective genotype sub-field data designated in the --genotype-feature-keywords <keywords> -- the ANALYSISFEATUREPROP.Type_Id obtains it’s value from these terms.
This option is only optional if --genotype-feature-keywords is not specified. Otherwise, it must contain exactly one term name for each genotype feature keyword supplied in the --genotype-feature-keyword <keywords>.

**FILE INFORMATION**

**--file-cv=**<cvname> The name of the set of controlled vocabulary terms containing the type used to classify the ANALYSISPROP row which stores the pathname of the input file -- the CV.Name value identifying the set of CVTERM rows in which the ANALYSISPROP.Type_Id value appears in the row which stores the pathname of the input file.
Defaults to null.

**--file-term=**<termname> The controlled vocabulary term used to designate the pathname of the input file -- the CVTERM.Name value of the ANALYSISPROP.Type_Id row where the pathname of the input file is stored.
Defaults to pathname.

**--sample-id-trans-cv=**<cvname> The set of controlled vocabulary terms in which appears the term designating a mapping between the genomic sample ids used in the VCF non-“fixed” column headings and the translated DBXREF.Accession values.

**--sample-id-trans-term=**<termname> The controlled vocabulary term, in the --sample-id-trans-cv <cvname> ontology, used as an ANALYSISPROP.Type_Id when recording a mapping between a non-“fixed” VCF column heading, a genomic sample id, and a DBXREF.Accession.
The ANALYSIS and ANALYSISPROP information

In addition to the values supplied by the command line options beginning with `--analysis-` there are other analysis related data that are loaded into the database.

If the `--analysis-cv` and `--analysis-term` arguments are not supplied then the `ANALYSIS.Type_Id` value is NULL. The `ANALYSIS.Analysis_Id` value is automatically generated by PostgreSQL.

An ANALYSISPROP row is created to store the fully qualified pathname of the input file. The `cvname` value supplied by `--file-cv` and the `termname` value supplied by `--file-term` are used to look up the CVTERM.CVTerm_Id value for the row storing the input pathname.

An ANALYSISPROP row is created for each of the VCF meta-information lines, those lines appearing at the top of the VCF file beginning with `##`. The `cvname` value supplied by `--analysisprop-cv` and the (case-insensitive) string from the VCF file which begins with at the start of the line with `##` (inclusive) and ends with the `=` character (exclusive) are used to determine the ANALYSISPROP.Type_Id value of the new row. The entire line in the VCF file is stored as the ANALYSISPROP.Value. In the case where there are more than 1 lines in the VCF file beginning with the same meta-information key, the ANALYSISPROP.Rank value is incremented and reflects the order in which the keys appear in the file.

Identifying the #CHROM source feature

The #CHROM value in each VCF data line must be associated with a source feature in the FEATURE table. Broadly speaking, this is done first by looking up the value as the feature name. If that fails then by the feature’s primary external database accession value. The final attempt is by the feature’s secondary external database accession values. Each of these lookups first checks that the discovered source feature is of the user-supplied type (typically, `is_a chromosome`). Failing that the discovered source feature is checked to see that it is a related type once removed from the user-supplied type. (Typically, that the discovered source feature has a type that `is_a chromosome`). The detailed process is as follows:

1. The ontology used to classify the source feature is identified.
   A CV.CV_Id value is obtained by matching CV.Name against the supplied `--source-feature-cv cvname` value. It is an error if none is found.

2. The ontology term which classifies the source feature is identified.
   A CVTERM.CVTerm_Id value is obtained by matching the above CV.CV_Id against the CVTERM.CV_Id and the `--source-feature-term termname` value against CVTERM.Name. It is an error if none is found.

3. The ontology used to relate the source feature classifications is identified.
   A CV.CV_Id value is obtained by matching CV.Name against the supplied `--source-feature-relationship-cv cvname` value. It is an error if none is found.

4. The ontology term used to relate the source feature classification to other ontology terms is identified.
   A CVTERM.CVTerm_Id value is obtained by matching the above CV.CV_Id against the CVTERM.CV_Id and the `--source-feature-relationship-term termname` value against CVTERM.Name. It is an error if none is found.

5. If a `--source-feature-db dbname` value was supplied an external database is identified, otherwise all external database cross references are used.
   To identify an external database a DB.DB_Id value is obtained by matching DB.Name against the supplied `dbname`. It is an error if a `dbname` is supplied and no match is found.

6. An attempt is made to identify the source feature by #CHROM value using the feature’s unique name and primary type.
   A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against FEATURE.UniqueName and the `--source-feature-term` ontology term (CVTERM.CVTerm_Id) value against the FEATURE.Type_Id value.

   If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. If no value is found the procedure continues.

---

20 An alternative to lookup of (typically) chromosome identifiers by external database accession value, is to consider the various alternate chromosome names to be synonyms. So, instead of using the DBXREF table, use the SYNONYM table. Although the use of the SYNONYM table is simpler, this is not a significant factor when it comes to data maintenance since there are not that many chromosomes or different naming schemes. Since the various alternatives for chromosomes names are maintained by external authorities using the DBXREF table better fits the model and this is the approach taken.
• An attempt is made to identify the source feature by #CHROM value using the feature’s unique name and primary type; where the the feature’s primary type must be directly related to that which the user supplies with --source-feature-term. Typically, this involves finding a matching source feature having a primary type which is_a chromosome.

A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against FEATURE.Uniquename. The FEATURE.Type_Id must match a CVTERM_RELATIONSHIP.Subject_Id having a CVTERM_RELATIONSHIP.Type_Id identified by the --source-feature-relationship-type option. Further, the the --source-feature-term ontology term (CVTERM.CVTerm_Id) must identify the CVTERM_RELATIONSHIP.Object_Id value.

If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. If no value is found the procedure continues.

• An attempt is made to identify the source feature by #CHROM value using the feature’s primary external database accession value.

A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against DBXREF.Accession and the --source-feature-term ontology term (CVTERM.CVTerm_Id) value against the FEATURE.Type_Id value, where the DBXREF row is related to the the FEATURE row by FEATURE.DBXref_Id.

If a --source-feature-db was supplied then the DBXREF.DB_Id must match the DB.DB_Id value discovered above.

If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. If no value is found the procedure continues.

• An attempt is made to identify the source feature by #CHROM value using the feature’s primary external database accession value and the feature’s primary type; where the feature’s primary type must be directly related to that which the user supplies with --source-feature-term. Typically, this involves finding a matching source feature having a primary type which is_a chromosome.

A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against DBXREF.Accession, where the DBXREF row is related to the the FEATURE row by FEATURE.DBXref_Id.

If a --source-feature-db was supplied then the DBXREF.DB_Id must match the DB.DB_Id value discovered above.

The FEATURE.Type_Id must match a CVTERM_RELATIONSHIP.Subject_Id having a CVTERM_RELATIONSHIP.Type_Id identified by the --source-feature-relationship-type option. Further, the the --source-feature-term ontology term (CVTERM.CVTerm_Id) must identify the CVTERM_RELATIONSHIP.Object_Id value.

If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. If no value is found the procedure continues.

• An attempt is made to identify the source feature by #CHROM value using the feature’s secondary external database accession values.

A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against DBXREF.Accession and the --source-feature-term ontology term (CVTERM.CVTerm_Id) value against the FEATURE.Type_Id value, where the DBXREF row is related to the the FEATURE row by way of the FEATURE_DBXREF table.

If a --source-feature-db was supplied then the DBXREF.DB_Id must match the DB.DB_Id value discovered above.

If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. If no value is found the procedure continues.

• An attempt is made to identify the source feature by #CHROM value using the feature’s secondary external database accession values; where the feature’s primary type must be directly related to that which the user supplies with --source-feature-term. Typically, this involves finding a matching source feature having a primary type which is_a chromosome.

A FEATURE.Feature_Id value is found by matching the VCF line’s #CHROM value against DBXREF.Accession, where the DBXREF row is related to the the FEATURE row by way of the FEATURE_DBXREF table.

If a --source-feature-db was supplied then the DBXREF.DB_Id must match the DB.DB_Id value discovered above.

The FEATURE.Type_Id must match a CVTERM_RELATIONSHIP.Subject_Id having a CVTERM_RELATIONSHIP.Type_Id identified by the --source-feature-relationship-type option. Further, the the --source-feature-term ontology term (CVTERM.CVTerm_Id) must identify the CVTERM_RELATIONSHIP.Object_Id value.

If exactly one value is found then the discovered value becomes the source feature. It is an error if more than one value is found. It is an error if no values are found.
The VCF Reference Base Nucleotides

Once the CHROM# source feature is identified it is used to find the database feature (FEATURE row) which represents the VCF data line’s reference base sequence, the \textit{reference base feature}.\footnote{Representing the VCF reference base nucleotides as a FEATURE row, rather than as a FEATURELOC row as appears to be common practice, has a number of advantages. FEATURE rows, unlike FEATURELOC rows, have a “first class” status: Features can be lifted from one version of a reference genome to the next. (If the genome does change, the old feature can be marked obsolete and a new feature created.) This allows analysis to be compared across reference genome versions, avoiding massive duplication of FEATURELOC rows, etc. FEATURES are referenced directly from other tables. And so forth.}

\begin{itemize}
  \item The ontology used to classify the reference base feature is identified. A CV.CV_Id value is obtained by matching CV.Name against the supplied \texttt{--reference-base-feature-cv cvname} value. It is an error if none is found.
  \item The ontology term used to classify the reference base feature is identified. A CVTERM.CVTerm_Id value is obtained by matching the above CV.CV_Id against the CVTERM.CVTerm_Id and the \texttt{--reference-base-feature-term termname} value against CVTERM.Name. It is an error if none is found.
  \item An existing reference base feature is searched for:
    \begin{itemize}
      \item The root of the \texttt{--reference-base-feature-relationship-term termname} (default: \texttt{part_of}) sequence graph containing the reference base feature is found. Typically this is a chromosome so this feature is called a chromosome throughout the remaining steps. The chromosome is found by recursively examining all related FEATURELOC.SrcFeature_Id values, starting at the source feature, and discovering the feature’s containing feature until no containing feature is found. It is an error if multiple roots are found\footnote{As might happen if a feature has multiple related FEATURELOC rows some of which themselves are, for some reason, not rooted at the chromosome. If this is an issue more command line parameters could be introduced which serve to identify the CVTERM row which represents the type (like “chromosome”) of the desired root.}.
      \item The offset of the source feature from the beginning of the chromosome is determined. This is done in the same fashion as the discovery of the chromosome, by recursive examination of FEATURELOC rows. To this is added the VCF line’s POS value, minus 1 to account for ones-based v.s. zero-based coordinate system. The result is the offset of the reference base feature from the beginning of the chromosome.
      \item All FEATURE rows that have a FEATURE.Type_Id matching the \texttt{--reference-base-feature-term CVTERM.CVTerm_Id} value (in the case of an SNV this would be a value meaning “base”), which are not obsolete (FEATURE.Is_Obsolete is \texttt{FALSE}), and which are located on the chromosome, are found. The candidate feature’s offset from the beginning of the chromosome are determined.
      \item The offset of the reference base feature from the beginning of the chromosome is compared with the offset of the candidate features from the beginning of the chromosome and mis-matches are discarded.
      \item The number of character’s in the VCF line’s REF value is compared against the number of bases in the remaining candidate features’. The number of bases is calculated by subtracting the FEATURELOC.FMin column from the FEATURELOC.FMax column. Mis-matches are discarded from the set of candidate features.
    \end{itemize}
\end{itemize}

Note Babase, at the time of this writing, is primarily concerned with SNVs, so the database FEATURE row identified as the reference base feature typically represents a single base nucleotide in the reference genome.

\footnote{Note that a VCF file REF value can contain a “padding base”. The padding base is in some cases optional, leading to multiple ways in which the same REF value is represented. If there is inconsistency in the use of padding bases there will be multiple representations of the same VCF reference base sequence in the database.}
– Candidate features with a NULL FEATURE.Residues value are discarded.
– The candidate features are checked to see if they are related to the specified external database entry. Candidates which do not match are discarded.

**Note**
In Babase the external database entry will typically be that of the Unknown Baboon.

The supplied --feature-db-name dbname value is used to obtain a DB.DB_Id value. It is an error if none is found.
If the --reference-base-feature-dbxref-accession argument was not supplied candidate features match only if their FEATURE.DBXref_Id value is NULL. Otherwise, the DBXREF table is searched for a DBXREF.DBXref_Id value where the DB_Id value matches that associated with the --feature-db-name dbname and the supplied accession value matches that of the DBXREF.Accession. A candidate matches when the discovered DBXref_Id value is that of the reference base feature’s FEATURE.DBXref_Id.

– It is an error if more than one candidate feature remains. It is an error if there is one candidate feature remaining and the FEATURE.Residues value does not match (compared case-insensitively) the VCF line’s REF value. If exactly one candidate feature remains it is the reference base feature.

• If an existing reference base feature is not found in the database then a reference base feature is created, as a new row in FEATURE. The column values of the new row are as follows:

**FEATURE VALUES OF A NEW REFERENCE BASE FEATURE**

**Feature_Id** The next available automatically generated id.

**DBXref_Id** The supplied --feature-db-name dbname value is used to obtain a DB.DB_Id value. It is an error if none is found.

If the --reference-base-feature-dbxref-accession argument was not supplied then the DBXref_Id value is made NULL. Otherwise, the DBXREF table is searched for a DBXREF.DBXref_Id value where the DB_Id value matches that associated with the --feature-db-name dbname and the supplied accession value matches that of the DBXREF.Accession.

If a DBXref_Id value is found it is used as the DBXref_Id value. It is an error if none is found.

**Organism_Id** The Organism_Id value of the source feature.

**Name** The VCF line’s ID value, or NULL when this is the missing value.

**Uniquename** The row’s Feature_Id value.

**Residues** The REF value, converted to upper case.

**Seqlen** NULL

**MD5Checksum** NULL

**Type_Id** The discovered --reference-base-feature-term CVTERM.CVTerm_Id value.

**Is_Analysis** TRUE

**Is_Obsolete** FALSE

**Timeaccessioned** The start time of the current transaction.

**Timelastmodified** Same as Timeaccessioned.

The reference base feature is located relative to the CHROM# value. A new FEATURELOC row is created for this purpose, as follows:

**FEATURELOC VALUES OF A NEW REFERENCE BASE FEATURE**

**Featureloc_Id** The next available automatically generated id.

**Feature_Id** The Feature_Id value of the newly created FEATURE row representing the reference base feature, above.

**Srcfeature_Id** The Feature_Id value of the (CHROM#) source feature.

---

24 This means that annotations which do not include FEATURE.Residues information will have duplicate feature rows created. This is done to avoid altering rows related to specific annotations.
25 This ensures that all rows created from a single VCF file have a uniform timestamp.
**FMin**  The VCF line’s POS value, minus 1.\(^\text{26}\)  
**Is_FMin_Partial** FALSE  
**FMax**  The FMin value, plus the number of characters in the VCF line’s REF value.  
**Is_FMax_Partial** FALSE  
**Strand** +1  
**Phase** NULL  
**Locgroup** 0  
**Rank** 0  

• Regardless of whether a new reference base feature is created or an existing one found, the reference base feature is related to the source feature with a FEATURE_RELATIONSHIP row and related to the analysis with an ANALYSISFEATURE row. Further, the VCF line’s ALT value is stored in one or more ANALYSISFEATUREPROP rows. All this is done as follows:  

The reference base feature is related to the source feature. A new FEATURE_RELATIONSHIP row is created for this purpose, as follows:  

**FEATURE_RELATIONSHIP**  
**Feature_Relationship_Id** The next available automatically generated id.  
**Subject_Id** The Feature_Id value of the newly created FEATURE row representing the reference base feature, above.  
**Object_Id** The Feature_Id value of the source feature.  
**Type_Id** The CVTERM.CVTerm_Id value of the --reference-base-feature-relationship-term termname.  
**Rank** 0  

An ANALYSISFEATURE row is created to relate the reference base sequence to the analysis. The QUAL value of the VCF file’s data line is stored therein. The row is created as follows:  

**ANALYSISFEATURE**  
**Analysisfeature_Id** The next available automatically generated id.  
**Rawscore** NULL  
**Normscore** NULL  
**Significance** The VCF data line’s QUAL value.  
**Analysis_Id** The ANALYSIS.Analysis_Id value of the ANALYSIS row representing the analysis which produced the VCF file.  
**Feature_Id** The Feature_Id value of the FEATURE row representing the reference base feature.  

The VCF data line’s ALT value is the sequence of alternate bases discovered by the analysis. This alternate base sequence is stored in a row in the ANALYSISFEATUREPROP table and thereby related to the reference base feature and the analysis. If there is more than one alternate value, the VCF specification allows for multiple comma-separated values, each value is stored in a separate ANALYSISFEATUREPROP row. The differing values are distinguished by Rank.  

---  

**Note**  
Since Babase is primarily concerned with SNV analysis the alternate base sequence will typically be a single base.  

---  

The new ANALYSISFEATUREPROP row is created as follows:  

**ANALYSISFEATUREPROP**  
**Analysisfeatureprop_Id** The next available automatically generated id.  
**Analysisfeature_Id** The Analysisfeature_Id value relating the reference base feature to the analysis.
**Type_Id** The CVTERM.CVTerm_Id value obtained by matching the CV.CV_Id derived from the `--reference-base-feature-cv cvname` value against the CVTERM.CV_Id and the `--alternate-base-term termname` value against CVTERM.Name.

It is an error if none is found.

**Value** The VCF data line’s ALT value, converted to upper case.

**Rank** 0 when there is a single ALT value, otherwise the Rank value is an integer starting with 1; the first ALT value has a Rank of 1, the second a Rank value of 2, etc. As with other Chado Rank values, when there are multiple values the Rank of 0 is not used.

### The Analyzed Genotype Columns

The non-“fixed” VCF columns, 10 and above (inclusive), must have headings which represent the genotype analyzed. These analyzed genotypes, the column headings, in the simplest case, match DBXREF.Accession values. Regardless, the genotypes analyzed must be represented as external database entities.

When the `--sample-id-ignore-case` option is present the non-“fixed” VCF columns are compared in a case-insensitive fashion, whether compared directly against DBXREF.Accession values or compared against translation table values. The default, when `--sample-id-ignore-case` is absent, is to perform comparisons in a case-sensitive fashion.

The `--feature-db-name dbname` value is used to obtain a DB_DB_Id value. It is an error if none is found.

The non-“fixed” column heading must all either be DBXREF.Accession values or must all be able to be translated to DBXREF.Accession values; values where the DBXREF.DB_Id value corresponds to the supplied `--feature-db-name dbname`. It is an error if this is not the case.

The default is for non-“fixed” column headings to be DBXREF.Accession values. To translate the non-“fixed” column headings to DBXREF.Accession values the `--sample-id-trans-table`, `--sample-id-trans-keycol`, and `--sample-id-trans-valuecol` options are used. When these arguments are used each non-“fixed” column heading is looked up in the `--sample-id-trans-table` column. When there is a match the value discovered in the `--sample-id-trans-valuecol` column is used and required to be a valid DBXREF.DB_Id. It is an error if the value looked up is not a DBXREF.DB_Id value of a DBXREF row having a DBXREF.DB_Id value matching the supplied `--feature-db-name dbname`. It is an error if the column heading is not a `keycol` value on the `--sample-id-trans-table`. It is an error if the column heading matches more than one row on the `--sample-id-trans-table`.

Regardless of whether the non-“fixed” VCF column headings are translated, a mapping between the non-“fixed” column number and the DBXREF row identifying genome is stored in the ANALYSISPROP table. The `--sample-id-trans-term name` is used to obtain a DB.DB_Id value. It is an error if none is found. The resulting CVTERM.CVTerm_Id is used as the ANALYSISPROP.Type_Id and an ANALYSISPROP is created for each non-“fixed” VCF column heading. The ANALYSISPROP.Rank is used to designate the column number of the non-“fixed” column, starting with 1 for the first non-“fixed” column. The ANALYSISPROP.Value is the DB.Name and DBXREF.Accession. This is stored in the canonical Chado external representation for identifiers: it is stored as dbname:accession.

The VCF file FORMAT column is parsed to discover the position (if any) of the GQ value, and the positions of the values of the `--genotype-feature-keywords keywords`. If the GQ sub-field does not exist in the FORMAT column `chado-vcf-load` silently uses a NULL value. If any of the specified keywords do not exist in the FORMAT column `chado-vcf-load` also silently ignores the absence, omitting the insertion of a corresponding ANALYSISFEATUREPROP row.

There need not be any non-“fixed” quote columns, they are optional. When such columns do exist the the content of each cell (column/line combination) is stored as described below.

Each value of each non-“fixed” column of each data line in the VCF file, each analyzed site, is represented as a feature in the database. Note that these “genotype” features do not contain a FEATURE.Residues value (it is NULL). This is because different analysis may produce different results. The “genotype” feature is located relative to the #CHROM source feature. The genotype feature is related to the reference base feature to denote that they represent the same feature on different genotypes.

The program first searches to see if a feature representing the analyzed site in the analyzed genotype exists. It does this by a method equivalent to the following:27

---

27 The discovered genotype feature is not examined to determine whether or not it is the same kind of feature as the reference base feature, the FEATURE.Type_Id value is not checked.

The discovery process also does not examine the FEATURE.Residues value of the discovered feature.

It is the responsibility of the database to ensure such integrity. (Whether or not the database actually does any kind of integrity checking.)
• The ontology used to relate the genotype feature to the reference base feature is identified.

A CV.CV_Id value is obtained by matching CV.Name against the supplied --feature-relationship-cv cvname value. It is an error if none is found.

• The ontology term used to relate the genotype feature to the reference base feature is identified.

A CVTERM.CVTerm_Id value is obtained by matching the above CV.CV_Id against the CVTERM.CV_Id and the --genotype-reference-relationship-term termname value against CVTERM.Name and CVTERM.Is_RelationshipType is non-0. It is an error if none is found.

• Find all features related to the reference base feature in the way genotype features are expected to be related to reference base features -- all FEATURE_RELATIONSHIP.Subject_Id values where FEATURE_RELATIONSHIP.Object_Id is the Feature_Id of the reference source feature and where the FEATURE_RELATIONSHIP.Type_Id value is the CVTERM.CVTerm_Id value of the --genotype-reference-relationship-term termname.

• Discard all matches at the wrong locus -- matches where the is not exactly one related FEATURELOC row relating the match to the #CHROM source feature with an FMin value of the FEATURELOC relating the reference base feature to the #CHROM source feature, with a similar corresponding FMax value, and with a Locgroup of 0 and a Rank of 1.

• Discard all obsolete matches -- matches where FEATURE.Is_Obsolete is TRUE.

• Discard all matches where the genotype is not appropriate -- where the FEATURE.DBXref_Id value of the discovered feature does not match the DBXREF.DBXref_Id of the column heading.

• It is an error if more than one match remains.

If a match remains then it is the feature with which the genotype information will be related. Otherwise a new feature is created as follows:

FEATURE VALUES OF A NEW GENOTYPE FEATURE

Feature_Id The next available automatically generated id.

DBXref_Id The external database identifier of the genotype identified by the column heading -- the DBXREF.DBXref_Id associated with the column heading.

Organism_Id The Organism_Id value of the reference base feature.

Name The VCF line’s ID value, or NULL when this is the missing value. (The Name value of the reference base feature.)

UniqueName The row’s Feature_Id value.

Residues NULL

Seqlen NULL

MD5Checksum NULL

Type_Id The discovered --reference-base-feature-term CVTERM.CVTerm_Id value. (The Type_Id value of the reference base feature.)

Is_Analysis TRUE

Is_Obsolete FALSE

TimeAccessioned The start time of the current transaction.

Timelastmodified Same as TimeAccessioned.

The genotype feature is located relative to the CHROM# value. A new FEATURELOC row is created for this purpose. Genotype feature locations are distinguished from "regular" feature locations by having a FEATURELOC.Rank of 1 instead of the usual 0. This is a duplicate of the FEATURELOC row created to locate the reference base feature, except for the Featureloc_Id value, the Feature_Id value, and the Rank value. The details are as follows:

FEATURELOC VALUES OF A NEW GENOTYPE FEATURE.

28 This ensures that all rows created from a single VCF file have a uniform timestamp.
**Featureloc_Id**  The next available automatically generated id.

**Feature_Id**  The Feature_Id value of the newly created FEATURE row representing the genotype feature, above.

**Srcfeature_Id**  The Feature_Id value of the (CHROM#) source feature.

**FMin**  The VCF line’s POS value, minus 1.\(^{29}\)

**Is_FMin_Partial**  FALSE

**FMax**  The FMin value, plus the number of characters in the VCF line’s REF value.

**Is_FMax_Partial**  FALSE

**Strand**  +1

**Phase**  NULL

**Locgroup**  0

**Rank**  1

The genotype feature is related to the reference base feature. A new FEATURE_RELATIONSHIP row is created for this purpose, as follows:

THE FEATURE_RELATIONSHIP VALUES WHICH RELATE A NEW GENOTYPE FEATURE TO THE REFERENCE BASE FEATURE.

**Feature_Relationship_Id**  The next available automatically generated id.

**Subject_Id**  The Feature_Id value of the newly created FEATURE row representing the genotype feature, above.

**Object_Id**  The Feature_Id value of the reference base feature.

**Type_Id**  The CVTERM.CVTerm_Id value of the --genotype-reference-relationship termname.

**Rank**  0

The genotype analysis data is then related to the genotype feature, and to the analysis. An ANALYSISFEATURE row is created to relate the genotype feature to the analysis. The GQ value, if one exists, of the VCF file’s data line/genotype column is stored therein. The row is created as follows:

THE ANALYSISFEATURE ROW RELATING THE GENOTYPE FEATURE TO THE ANALYSIS

**Analysisfeature_Id**  The next available automatically generated id.

**Rawscore**  NULL

**Normscore**  The GQ value of the line/column, or NULL if there is no GQ keyword in the VCF data or the GQ value is “.”.

**Significance**  NULL

**Analysis_Id**  The ANALYSIS.Analysis_Id value of the ANALYSIS row representing the analysis which produced the VCF file.

**Feature_Id**  The Feature_Id value of the FEATURE row representing the genotype feature.

\(^{29}\) To adjust for Chado’s interbase coordinate system.
The VCF genotype sub-fields designated by the `keywords` of `--genotype-feature-keywords` (if any) are stored in new `ANALYSISFEATUREPROP` rows.

The content of the new `ANALYSISFEATUREPROP` rows are as follows:

**STORING THE VCF GENOTYPE KEYWORD VALUES IN ANALYSISFEATUREPROP ROWS**

No row is stored if the genotype data column in the VCF file omits one of the `--genotype-feature-keywords` keywords.\(^{30}\) A VCF "omitted value", i.e. ".", is stored in row as NULL.

The `--genotype-feature-cv cv` value is compared against CV.Name to discover a CV.CV_Id value. The `--genotype-feature-terms termmames` values are used to discover CVTERM.CVTerm_Id values, where the CVTERM.Name value matches the one of the `termmames` values and the CVTERM.CV_Id value matches the discovered value for `cv`, above.

**Analysisfeatureprop_Id** The next available automatically generated id.

**Analysisfeature_Id** The `ANALYSISFEATURE.Analysisfeature_Id` value of the `ANALYSISFEATURE` row representing the genotype analysis result.

**Type_Id** The, respective, discovered `--genotype-feature-term CVTERM.CVTerm_Id` value, per above, classifying the genotype keyword value.

**Value** The genotype feature keyword data as it appears in the VCF file, with one exception: the "." value is stored as NULL.

**Rank** 0 for data that is not comma-delimited. For comma delimited data Rank is the (ones-based) position of the comma delimited sub-field.

**Signals**

The `chado-vcf-load` responds to the SIGUSR1 signal by printing the line number being processed.

The `ps` command is used to find the pid of the process running the `chado-vcf-load` command. The `kill` command is then used to send the USR1 signal to the process. The process responds to the USR1 signal by printing the line number being processed. Note that this is all done in a separate terminal window; the terminal running `chado-vcf-load` is where output occurs.

**Example 4.2 Discovering how many lines have been processed**

```bash
$ ps ax | grep chado-vcf-load
7889 pts/3  R+   6:20 chado-vcf-load --dbname=babase_vcf --analysis←
-<analysis descr=GATKanalysis --analysis-description=GATK_norefDB_samtools.vcf ←
--analysis-program=unknown --analysis-programversion=0 --analysis-←
<timeexecuted=2015-01-05 13:01 GATK_norefDB_samtools.vcf
8141 pts/4  S+   0:00 grep chado-vcf-load
$ kill -s USR1 7889
```

**Tips**

Depending on the size of the VCF file, `chado-vcf-load` can take some time to run. The `screen` command can be used to run `chado-vcf-load" in the background", i.e. detached from an interactive terminal. This technique prevents accidental interruption due to network disconnection and the like.

**Environment Variables**

**CHADO_VCF_LOAD** Options which are passed to `chado-vcf-load` automatically. This can be used to establish per-user or sitewide defaults.

**PGDATABASE, PGHOST, PGPORT, PGUSER** Default PostgreSQL connection parameters. For further information see the PostgreSQL documentation.

---

\(^{30}\) Omitting a keyword would appear to be in violation of the VCF 4.1 specification, although this is not explicit.
FILES

~/.pgpass  The PostgreSQL per-user password file is consulted to obtain passwords. For further information see the PostgreSQL documentation.

Assumptions and Limitations

**chado-vcf-load** makes some assumptions. If these are not met then errors may result.

**ASSUMPTIONS**

- VCF files are assumed to be conformant to both the VCF specification and to the restricted subset of the specification handled by **chado-vcf-load**.

---

**Warning**

Unless otherwise noted data values are accepted as-is and loading a non-conformant data file into the database will produce unspecified results.

---

- “Padding bases” must be used consistently in the loaded VCF files or else a single reference base sequence might be represented by more than one row in **FEATURE**.

- Reference base sequences must be stored in **FEATURE.Residues** for them to be discovered and used by **chado-vcf-load**, otherwise a single reference base sequence might be represented by more than one row in **FEATURE**. This might be a problem for longer sequences or features created by other software.

- There is no provision for adding “secondary tags” (i.e. **FEATURE_CVTERM** rows) to newly created, or existing, features. Such tags might identify features as, e.g., SNVs.

**chado-vcf-load** processes a subset of the VCF 4.1 specification. The following components of the specification are not handled:

- Lines bearing the following tags containing meta-information are not parsed and do not affect processing:
  
  - `##fileversion`
  - `##INFO`
  - `##FILTER`
  - `##FORMAT`
  - `##ALT`
  - `##assembly`
  - `##contig`
  - `##SAMPLE`
  - `##PEDIGREE`
  - `##pedigreeDB`

- The CHROM# value is not parsed for angle brackets (<> and corresponding `##contig` and `##assembly lookup`.

- Telomeres are not handled in POS values. POS values are expected to be >=1 and <= the number of bases in the source feature. POS values are not validated and unexpected values will produce unspecified results.

- The ID value is not correctly interpreted. It probably should be parsed for semi-colons and the tokens treated as external database (DBXREF) accession values. Instead the entire value is treated as a single non-unique identifier and used (only when a row is created) solely for supplying a convenient display name.

---

**31** Instead **chado-vcf-load** looks for alternate ids in DBXREF.Accession values, as documented. It does not strip angle brackets (<> from the CHROM# values. It may be necessary to introduce a command line option to control such stripping.
• Extremely long REF values should be converted into md5 sums and stored in the FEATURE.MD5Checksum column, the length stored in the FEATURE.Seqlen column, and processing adjusted accordingly.

• The REF and ALT columns are not parsed for angle-bracketed ID strings (<ID>).

• The value of the FILTER column is ignored.

• The value of the INFO column is ignored.

• The VCF GQ genotype sub-field is hardcoded and stored in Normscore.

• The parsing of VCF genotype sub-field keywords is hardcoded. This determines whether or not the sub-field itself is parsed and stored with a Rank value greater than 0. If a comma is present in the data the sub-field is parsed and Rank is greater than 0.

• The FEATURELOC.Rank value of the reference base feature is always 0. Should there be multiple versions of the reference genome the Rank value should probably be adjusted accordingly when a feature is located on more than one reference genome. There will probably need to be related adjustments to the Rank values of the genotype features. The issue of liftover to a new reference genome has yet to be fully explored.

4.1.6 copy-to-abrp-genomics

copy-to-abrp-genomics — copies and anonymizes portions of Babase/Chado to the abrp-genomics public portal machine

Synopsis

copy-to-abrp-genomics --host hostname --key keyfile dbname

Description

copy-to-abrp-genomics copies portions of the Babase/Chado database to the abrp-genomics public portal machine. The portion of the database copied is anonymized.

The username abrpdbloader is used to log in to the remote system and load the remote database.

Arguments

--host hostname The name of the computer to which to copy the data.

--identity keyfile The ssh key file used to identify the user on the remote computer.

dbname The name of the database from which to copy data. If the dbname is babase the data is copied to the live public portal website. Otherwise the data is copied to the test portion of the public portal website.

The copy process

In lieu of copying the STOCK table and the tables containing phenotypic information on the individuals a query is used to create the stock_phenotypes table on abrp-genomics. This query is obtained from the /etc/babase/stock_phenotypes.sql file.

The following tables are copied:

• ANALYSIS
• ANALYSISPROP
• ANALYSIS_ND_EXPERIMENT
The copy process anonymizes the identifiers stored in DBXREF.Accession. Indexes are copied but constraints and other referential integrity rules are not. The target database is expected to be read-only.

FILES

/etc/babase/stock_phenotypes.sql The query used to create the stock_phenotypes table on the destination system.

4.1.7 receive-abrp-genomics-schema

receive-abrp-genomics-schema — receives a copy of portions of Babase/Chado and installs it into the database on the public portal machine

Synopsis

receive-abrp-genomics-schema dbname

Description

receive-abrp-genomics-schema receives portions of the Babase/Chado database and restores them into the appropriate database on the public portal machine.

Note

This program is invoked automatically by copy-to-abrp-genomics. It is not expected to be run manually under normal circumstances.

Arguments

dbname The name of the database from which the data was copied. If the dbname is babase the data is copied to the live public portal website. Otherwise the data is copied to the test portion of the public portal website.

The restore process

The program is designed to run on the target machine, the public portal. It receives the output from copy-abrp-genomics-schema on stdin.
4.2 Web Interfaces

4.2.1 VCFDump (Download anonymized VCF files)

The VCFDump web interface delivers a compressed, anonymized VCF file to the public.

4.2.1.1 Description

The interface presents a list of the existent VCF files. These are the files of analysis labeled with the ANALYSIS.Type_Id value indicating a genomic analysis, and having a related stored pathname to a readable file.

The non-“fixed” column headings are anonymized by replacement of the on-disk values with anonymized values which map to the analyzed genomes. The lookup table used is configurable per the DB portion of the DBXREF identifier stored in ANALYSISPROP.

Errors, warnings, and so forth are logged. The application logs an informational message whenever a VCF file is downloaded. This is independent of and in addition to the normal logging of page downloads by the webserver.

4.2.1.2 Configuration

VCFDump gets its configuration from a text file. The comments in the file serve to document the various configuration parameters.

4.3 SQL Utilities

4.3.1 chado_analysis_delete

chado_analysis_delete — undo the loading of an analysis into the database

Synopsis

INT chado_analysis_delete ( INT analysis_id );
INT chado_analysis_delete ( VARCHAR(255) name );

Description

chado_analysis_delete() deletes analysis and related data. It undoes an analysis load. It may be called either with an ANALYSIS.Analysis_Id or with an ANALYSIS.Name to identify the analysis to be deleted. See the examples below.

Caution

In order to ensure that all information related to an analysis is deleted it is almost always necessary to use chado_analysis_delete() to delete ANALYSIS rows. Deleting rows from ANALYSIS with an SQL DELETE statement or similar method will not delete the FEATURE related information put into the database by chado-vcf-load, nor will a DELETE statement remove the ND_EXPERIMENT related information put into the database by chado-funcgen-load.

Note

chado_analysis_delete() never deletes STOCK information.

32 See

33 See the description of how the mapping between the VCF column number and the analyzed genotype is stored.
chado_analysis_delete() takes either one of the following two arguments:

**analysis_id**  An ANALYSIS.Analysis_Id value identifying the analysis to be deleted.

**name**  An ANALYSIS.Name value identifying the analysis to be deleted.

chado_analysis_delete() is designed to delete all the rows added to the database by chado-vcf-load and chado-funcgen-load. Hence, it deletes all rows in the following tables which are related to the deleted analysis row:

- ANALYSISPROP
- ANALYSIS_CVTERM
- ANALYSISFEATURE
- ANALYSISFEATUREPROP
- ANALYSIS_ND_EXPERIMENT
- ND_EXPERIMENT
- ND_EXPERIMENTPROP
- ND_EXPERIMENT_STOCK
- ND_EXPERIMENT_STOCKPROP

It is assumed that none of the above tables are related to an ANALYSIS row unless the relationship was established by one of the upload programs.

Deleting uploaded FEATURE rows is not as straightforward. FEATURE rows can be shared between analysis and can exist independently of analysis so a test is required to determine whether a FEATURE row exists solely to support the analysis being deleted. FEATURE rows related to the deleted analysis are deleted if they meet all of the following properties:

- The feature is not related to another analysis.
- The feature entered the db as the result of an analysis -- FEATURE.Is_Analysis is TRUE.
- The feature is related to one of the genotypes that the analysis analyzed or is related to the reference genome.

To see if the feature is related to one of the genotypes analyzed the FEATURE.DBXref_Id is compared against the database accession value stored in the analysis’ ANALYSISPROP.Value, for that ANALYSISPROP row related to the analysis that has a ANALYSISPROP.Type_Id value matching the CVTERM row with a CVTERM.Name of vcf_column_source related to the CV having a CV.Name of babase_chado_terms. The ANALYSISPROP.Value identifies an external database accession in the format of DB.Name value, colon (:), DBXREF.Accession value. It is this that is dereferenced and so translated to an actual DBXREF.DBXref_Id value to be compared against the FEATURE.DBXref_Id value.

To see if the feature is detail added to the reference baboon sequence by chado-vcf-load the FEATURE.DBXref_Id is compared against the DBXREF row having a DBXREF.Accession value of reference_baboon and a related DB row with a DB.Name of unique_baboons.

- The information about the bases which make up the feature must be appropriate for a feature created by chado-vcf-load.

If the feature is related to one of the genotypes analyzed then the FEATURE.Residues value must be NULL. If the feature represents information about the reference baboon sequence then the value of FEATURE.Residues must be non-NULL.

When deleting rows from the FEATURE table related rows from the following tables are also always deleted:

- FEATURELOC
- FEATUREPROP
- FEATURE_CVTERM
- SYNONYM
- FEATURE_DBXREF
- FEATURE_RELATIONSHIP
EXAMPLES

Example 4.3 Deleting an Analysis by Analysis Id

```
babase=> select chado_analysis_delete(123);
    chado_analysis_delete
----------
       123
(1 row)
```

Example 4.4 Deleting an Analysis by Analysis Id

```
babase=> select chado_analysis_delete('My_Bad_Analysis');
    chado_analysis_delete
----------
       123
(1 row)
```

Example 4.5 Deleting an Analysis by Query

```
babase=> select chado_analysis_delete(analysis.analysis_id)
   from analysis
   where analysis.timeexecuted >= '2015-01-01';
    chado_analysis_delete
----------
       123
       124
       125
(3 rows)
```

RESULT

chado_analysis_delete() returns the ANALYSIS.Analysis_Id of the deleted analysis, or NULL if no ANALYSIS was deleted.
Appendix A

Indexes Added To Chado

Babase has added indexes to Chado to improve performance. These are listed separately in an appendix since they do not directly impact the function of the database. Indexes on multiple keys are in the order given.

A.1 Indexes On ANALYSIS

A.1.1 analysis_idx1b on ANALYSIS.Type_Id

A btree index on ANALYSIS.Type_Id.

A.1.2 analysis_bc1 on ANALYSIS.Name (Unique)

A constraint (and therefore a unique btree index) requiring ANALYSIS.Name be unique.

A.2 Indexes On FEATURE

A.2.1 feature_idx1b on FEATURE.Feature_Id, FEATURE.DBXref_Id

A btree index on FEATURE.Feature_Id, FEATURE.DBXref_Id.

A.3 Indexes On FEATURE_RELATIONSHIP

A.3.1 feature_relationship_idx1b on FEATURE_RELATIONSHIP.Object_Id, Subject_Id, Type_Id

A btree index on FEATURE_RELATIONSHIP.Object_Id, FEATURE_RELATIONSHIP.Subject_Id, FEATURE_RELATIONSHIP.Type_Id.

A.4 Indexes On FEATURELOC

A.4.1 featureloc_idx1b on FEATURELOC Feature_Id, FMin, FMax

A btree index on FEATURELOC.Feature_Id, FEATURELOC.FMin, FEATURELOC.FMax.