GXD Schema
Annotated Overview

This document serves as a bridge between the document describing the high-level conceptual model of GXD database and the detailed schema-level view available through the online MGI schema browser. It shows a series of views of the schema. Each view shows a small set of logically related tables and their join-key connections. The accompanying text provides an overview of how the data in the tables are connected, notes data encoding of special interest, and generally orients the reader to how the data are organized. The intended audience are those who wish to compose SQL queries against the GXD.

In each view, a box represents a database table and contains the table's name. Arrows indicate foreign-key pointers between records; they point from the table conatining the foreign key to the table containing the referenced object. The arrows are labelled with the name of the foreign key column, which is in most cases the same as the primary key column that it references. Tables that are the focus of the current view are white; those that are connected to, but not in the focus of the current view, are gray.

GXD is an integrated part of the larger MGI system. As such, many of the views that follow are not specific to GXD, but are shared with the rest of MGI. This document does NOT describe all of MGI - that is beyond its scope. As well, it does not cover every detail of the GXD schema - those are available via the schema browser. Rather, this document is somewhat like a roadmap that shows how the various "neighborhoods" are structured and interrelated.

The reader should be aware of a few common MGI idioms. First, all database tables contain creation and modification date columns (creation_date, modification_date). Many tables also include columns specifying the MGI editor or process that created and last modified the record (_CreatedBy_key and _ModifiedBy_key). Next, MGI uses naming conventions to aid in understanding. Table names have three-letter prefixes that serve as a loose grouping mechanism. E.g., the tables concerned with accession number handling have the prefix "ACC". Also, columns that are primary or foreign keys usually have names of the form "_Type_key", where "Type" is the object's type. E.g., "_Marker_key", "_Assay_key", etc.

The database is never finished; it is always in a state of transition. Right now, for example, there is one neighborhood that contains the mouse developmental anatomy, and another (newer) neighborhood containing general structured vocabularies. While moving the mouse developmental anatomy into the new neighborhood is planned, it has not yet happened. Therefore, this document contains both views.

Accession IDs

All accession IDs associated with database objects are stored in a single table, ACC_Accession. This table contains all MGI#s assigned to objects like Genes, References, Sequences, Assays, etc. It also contains all foreign database accession IDs, such as GenBank numbers associated with Sequences or MEDLINE numbers associated with references. Since accession IDs come from different databases, we represent those databases as objects (ACC_LogicalDB). Each accession number refers to the logical database that 'owns' it. For example, all MGI#s refer to the MGI logical database record (key=1), and all GenBank IDs refer to the Sequence DB logical database (key=9).

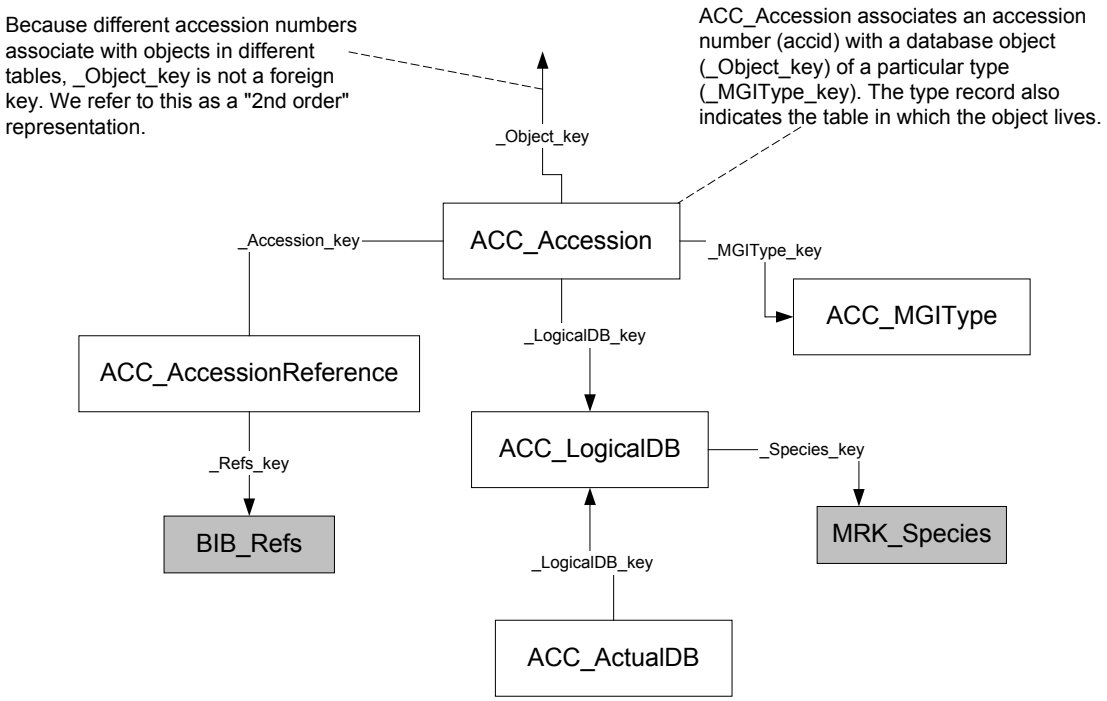
Each accession number refers to the object it is attached to (_Object_key). Since different accession numbers associated with objects in different tables, each accession record also stores the type (_MGIType_key) of the object referred to by _Object_key. So, to get all the accession ids associated with a gene object whose primary key is 1234, you would select from the accession table where _Object_key = 1234 and _MGIType_key = 1 (the key for type Marker).

MGI separates the concept of logical database (ACC_LogicalDB) from specific incarnations of it (ACC_ActualDB), which provide specific URLs. This could be used to represent different mirror sites for the same database. For example, the SwissProt logical db has actual dbs EBI and at Expasy. Similarly, the "Sequence DB" logical db has actual dbs GenBank, EMBL, and DDBJ. The URL for an actual db may contain "@@@@", which is replaced by the accession id. For example, the actual db for SwissProt at EBI has the URL:

http://www.ebi.ac.uk/htbin/swissfetch?@@@@

So, the link to P22893 (Zfp36) would be:

http://www.ebi.ac.uk/htbin/swissfetch?P22893

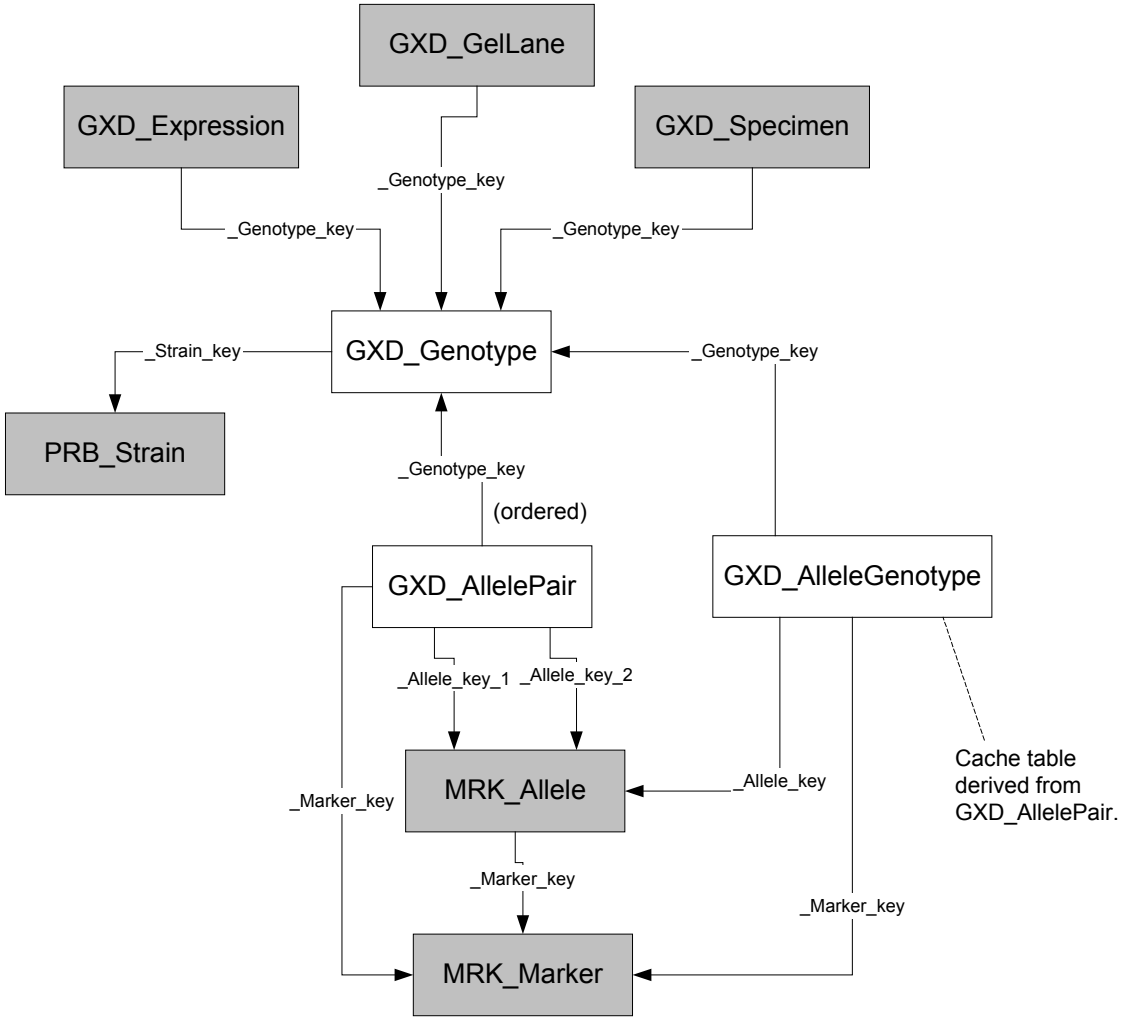


Genotypes

GXD expression data are associated with genotypes. Each lane sample in a gel assay (GXD_GelLane) and each specimen in an In Situ assay (GXD_Specimen) is associated with a genotype (_Genotype_key).

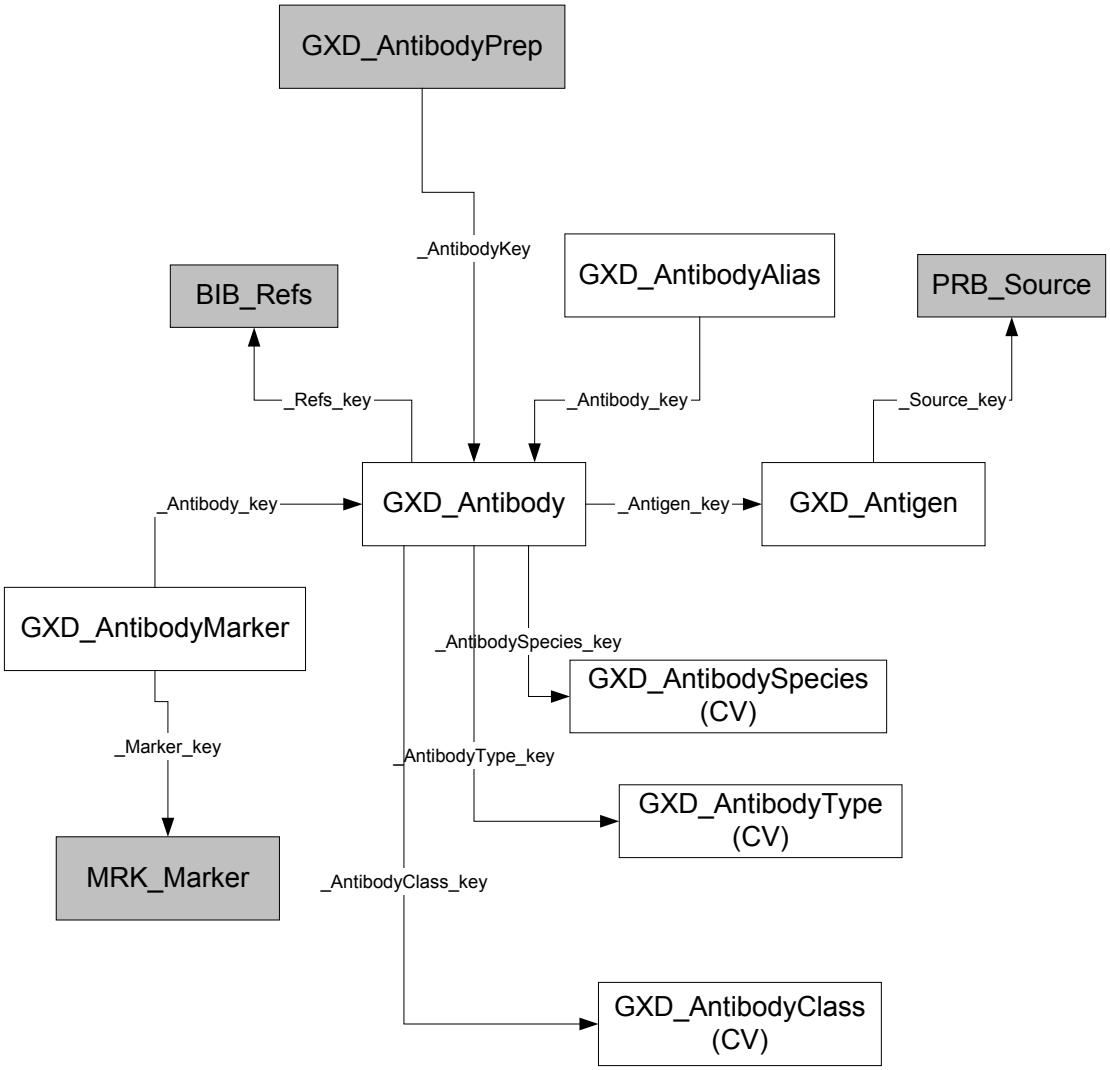
A Genotype is represented as a background strain (_Strain_key) plus some number (zero or more) of pairs of alleles (GXD_AllelePair). Each allele pair refers to one or two alleles of the same gene (_Allele_key_1, _Allele_key_2, _Marker_key) and to the genotype is belongs to (_Genotype_key). For hemizygous genes, _Allele_key_2 is null.

The table, GXD_AlleleGenotype, is a cache table in which each allele pair record is broken into one or two records, each representing a single allele. This table is generated to make querying easier. (We should perhaps make GXD_AlleleGenotype the primary table, and eliminate GXD_AllelePair.)



Antibodies

Antibodies used as probes in protein expression assays are represented in GXD_Antibody and associated tables. An antibody has simple attributes such as a name, and refers by key to several controlled vocabularies: the antigen from which it was raised (GXD_Antigen), the species it was raised in (GXD_AntibodySpecies), and the type and class of the antibody (GXD_AntibodyType and GXD_AntibodyClass, respectively. GXD_AntibodyMarker is a join table associating antibodies with the genes whose products they recognize.

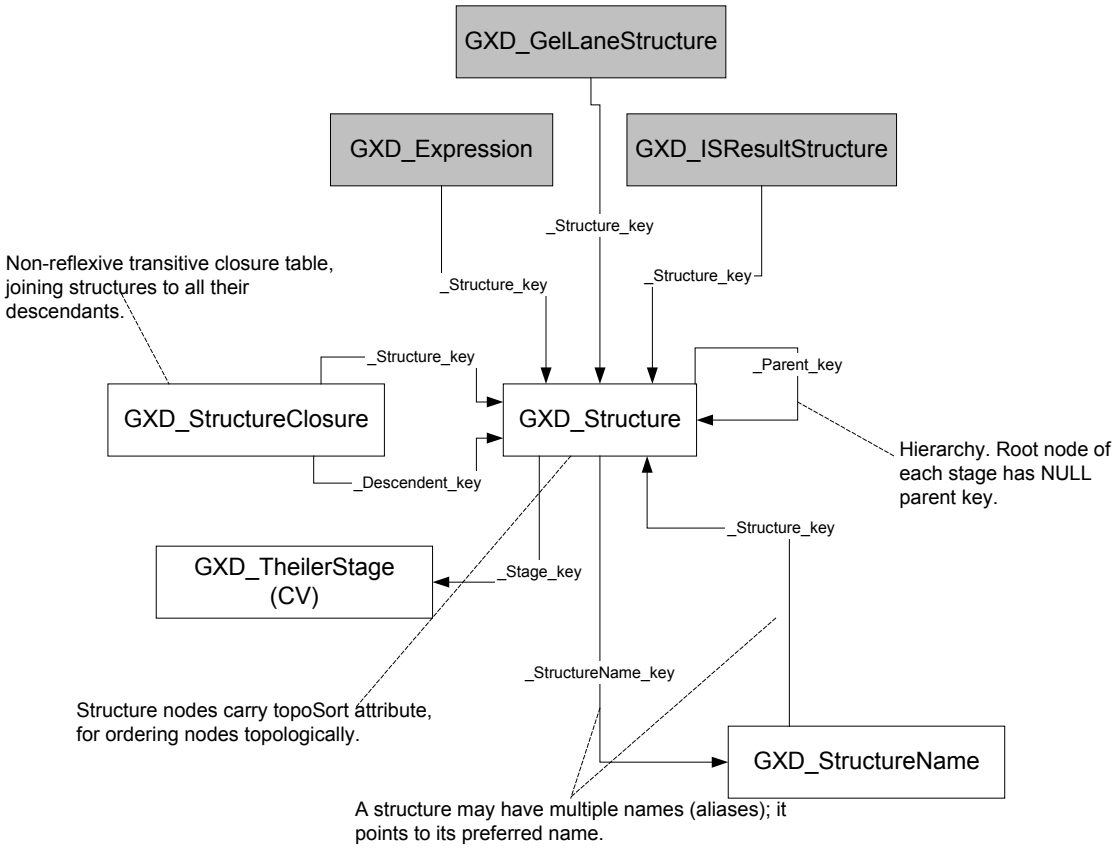


Mouse Developmental Anatomy

Each Theiler stage of development is represented by a separate tree of anatomical terms. A record in GXD_Structure represents one anatomical structure at a specific stage. Each structure points to the parent structure (_Parent_key) that contains it. The root node of each stage has a null parent key. The cache table, GXD_StructureClosure, contains the transitive closure; each record connects one object (_Structure_key) with one descendant (_Descendent_key) [sic].

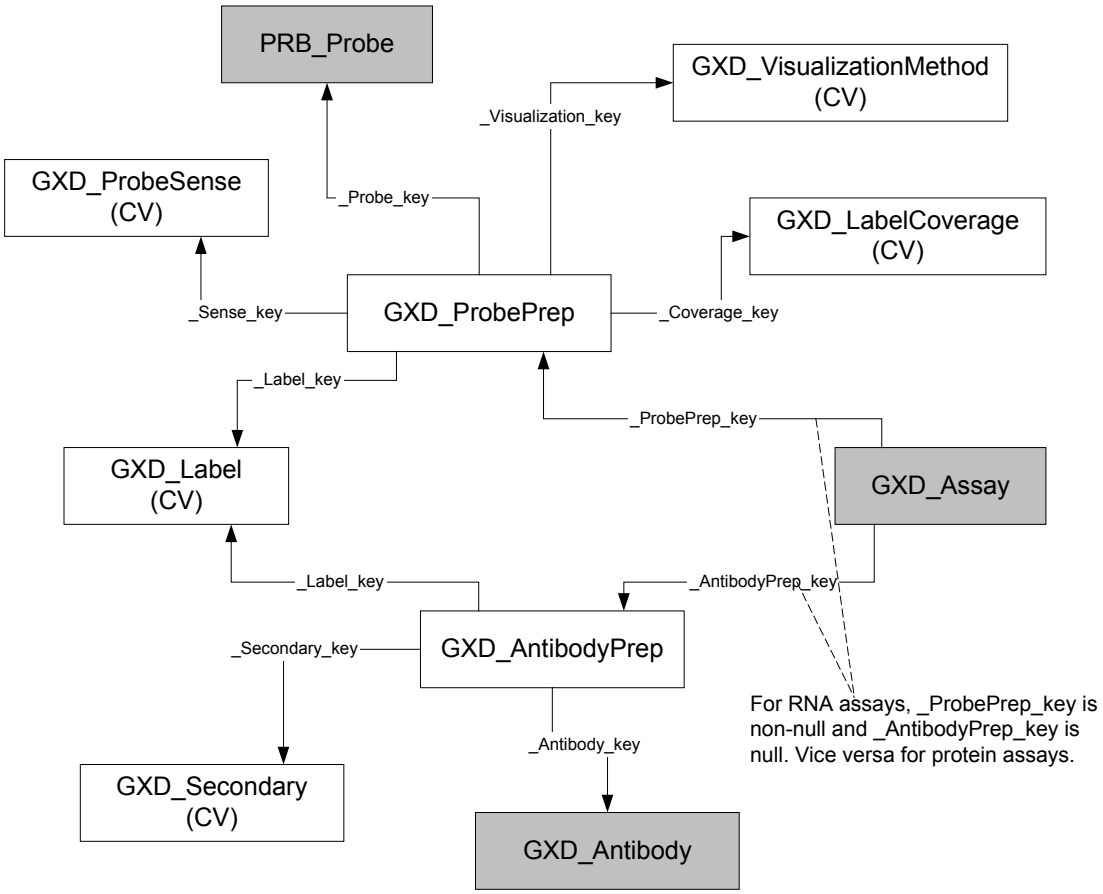
GXD_StructureName contains official names and aliases of a structure; each name points to the structure (_Structure_key) it belongs to. Each structure points to its preferred name (_StructureName_key). Printing the name of a structure usually requires additional context in order to be meaningful. This context consists of printing the names of ancestors up to some selected level. Certain nodes are marked (printStop=1) as being "high enough". These printable names are computed and cached (printName) for performance.

(Since the introduction of GXD, MGI has subsequently developed generalized support for structured vocabularies. See "Vocabs", later in this document. Although planned, the mouse developmental anatomy has not yet been migrated to these newer structures.)



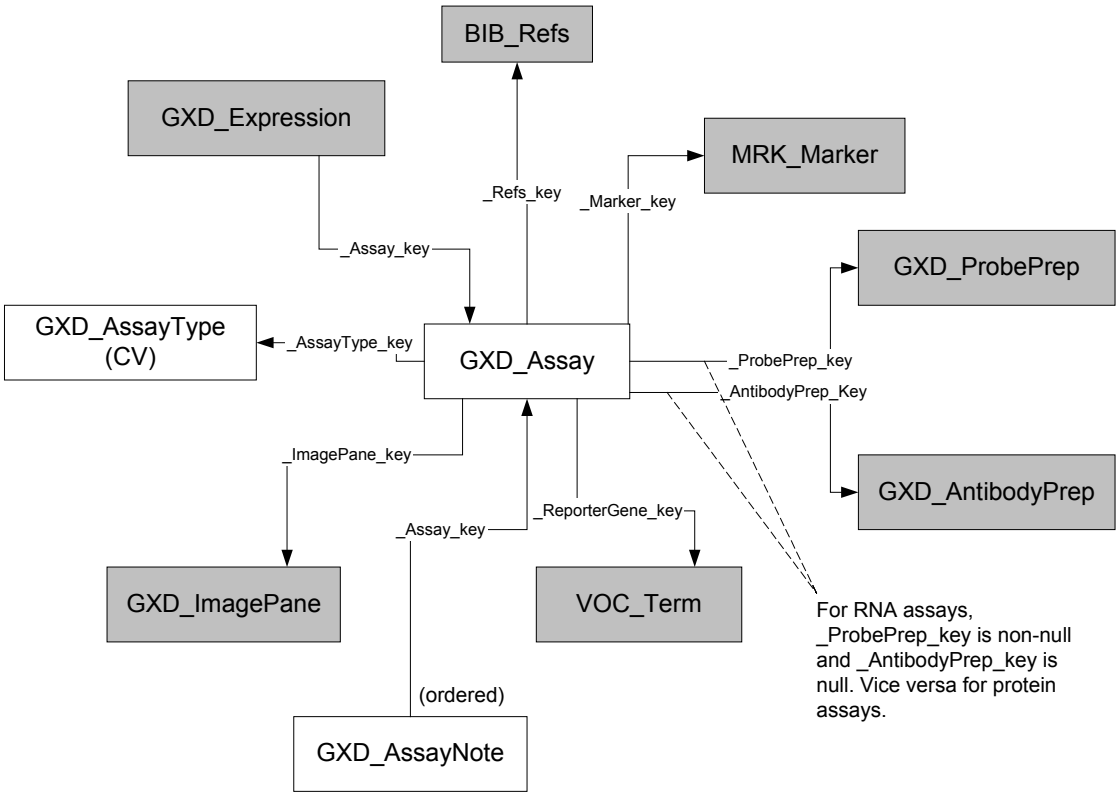
Reagent preparations

These tables capture information about the special preparation of probes used in expression assays. Preparations of nucleotide probes are captured in GXD_ProbePrep, while those for antibody probes are in GXD_AntibodyPrep. Each prep record refers to the actual nucleotide probe (_Probe_key) or antibody probe (_Antibody_key) being used. Each expression assay (GXD_Assay) refers to one prep record (either _ProbePrep_key or _AntibodyPrep_key).



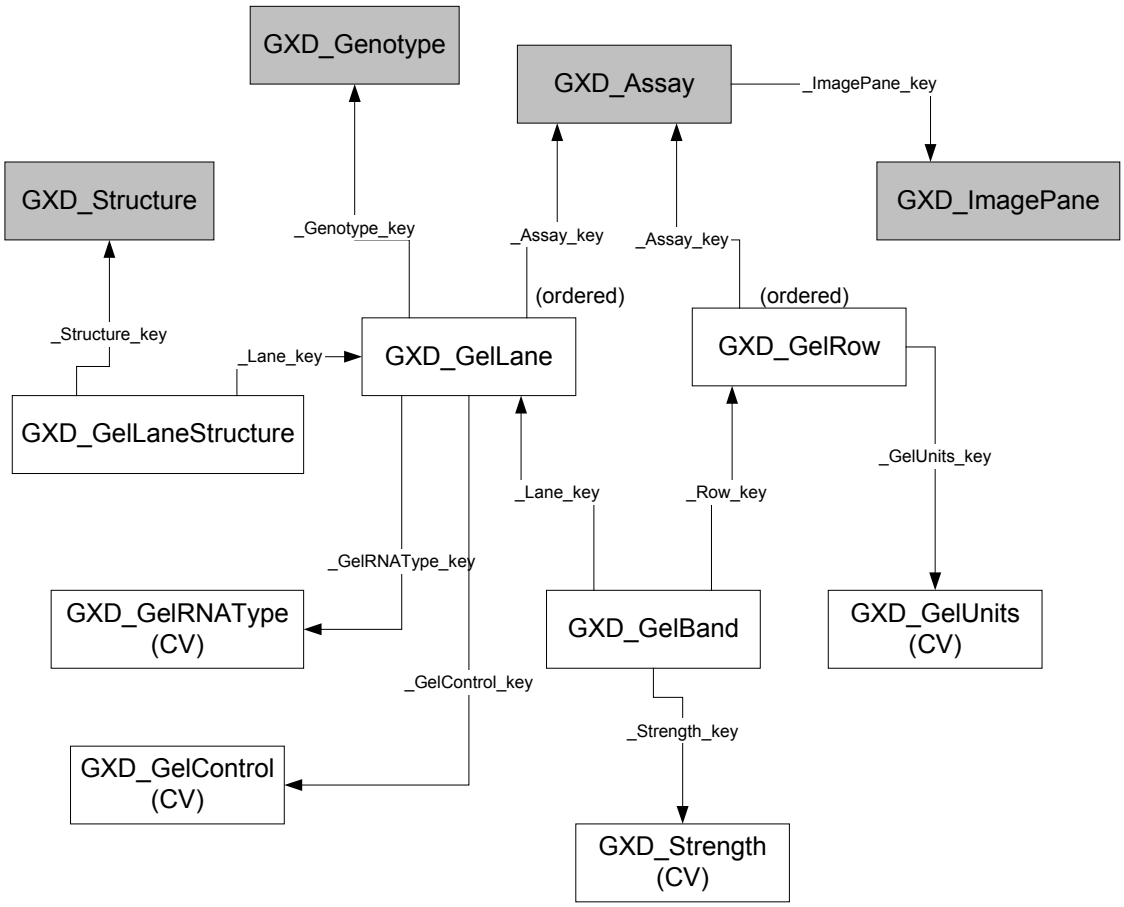
Expression Assays

Each record in GXD_Assay represents on expression assay, which comprises the reagents and results for one probe/one gene. In the schema, assays are distinguished by whether they are gels or insitus and whether they detect transcripts or polypeptides. See descriptions of gel assays and in situ assays for details.
For gel assays, an image of the gel, if available, is referenced (_ImagePane_key).
For knock-ins, the reporter gene is referenced (_ReporterGene_key).



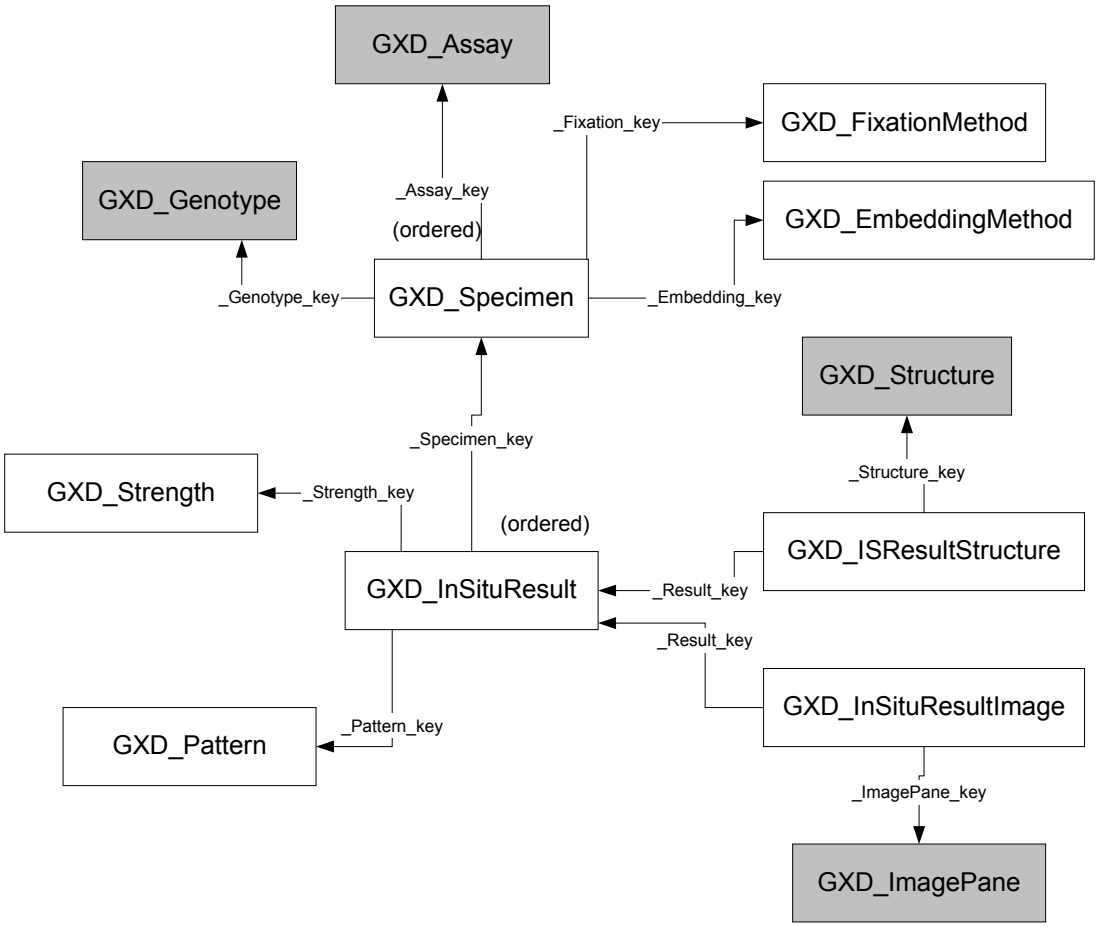
Gel Assays

A gel assay represents the gel as a matrix of columns (GXD_GelLane) representing the different samples, rows (GXD_GelRow) representing all bands (present or not) of a given size, and cells (GXD_GelBand) representing a specific band or lack thereof. Each lane describes the sample by its genotype (_Genotype_key), age, sex, the tissue/structure(s) (via GXD_GelLaneStructure join table), the type of RNA (_GelRNAType_key, if applicable). Various types of control lane (_GelControl_key > 1) are distinguished but otherwise contain no data. Each band in a data lane (_GelControl_key=1) describes its strength (_Strength_key) in qualitative terms (e.g. "absent","present","trace"...).



In Situ Assays

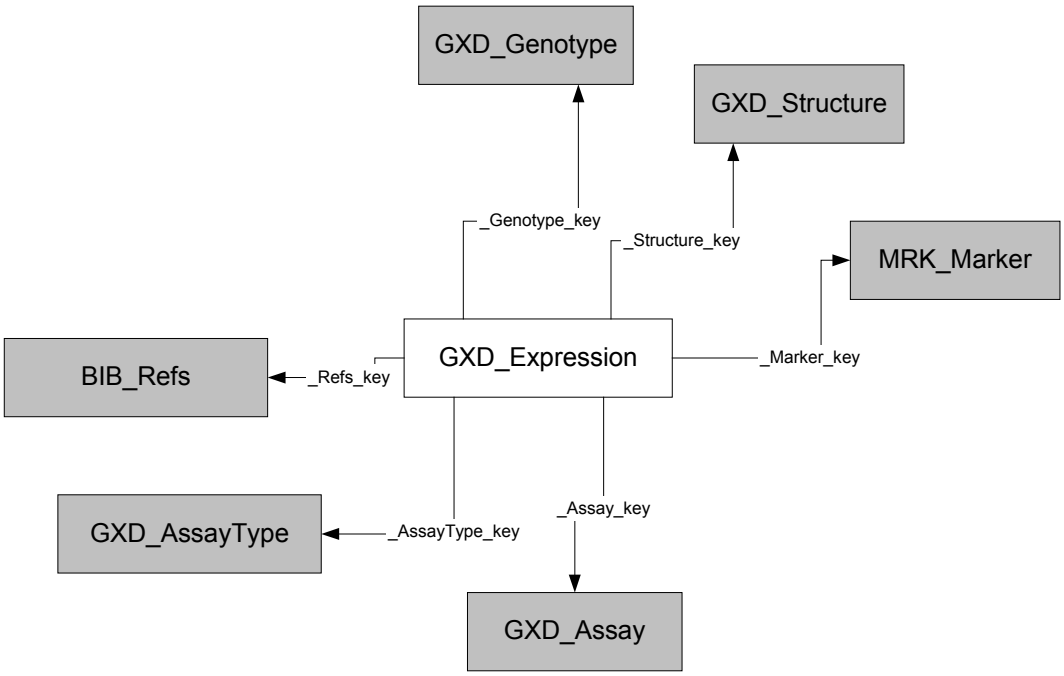
An in situ assay contains data for one or more specimens (GXD_Specimen) associated with the assay record. The specimen is described by its genotype (_Genotype_key), age, and sex, how it was prepared (_Fixation_key and _Embedding_key). Each result (GXD_InsituResult) describes expression in specific tissues/structure(s) of the specimen via GXD_ISResultStructure join table. Expression is qualitatively described by strength (_Strength_key) and spatial pattern (_Pattern_key). Each result may also be depicted in zero or more images (via GXD_ISResultImage join table).



Expression Results
(cache table)

This table caches yes/no statements of expression, gathered from In Situ and Gel expression assays. It is maintained by the system triggers and stored procedures. The vast majority of expression queries are directed against this table.

Each record indicates whether expression of a given gene (_Marker_key) was or was not detected (expressed) in a specific place/time (_Structure_key) in a given biocontext (_Genotype_key, age, sex). Each statement has a key (_Expression_key) and refers back to where it came from (_Assay_key, _AssayType_key, _Refs_key).



Images

The image tables store information about the figures in a publication. An image (IMG_Image) contains of one or more sub-images called "panes" (IMG_ImagePane). Each pane may have a label, such as "A", "B", "C", ... or "+/+", "+/-", ... Each image (IMG_Image) has a legend (figureLabel), copyright notice (copyrightNote), and citation (_Refs_key).

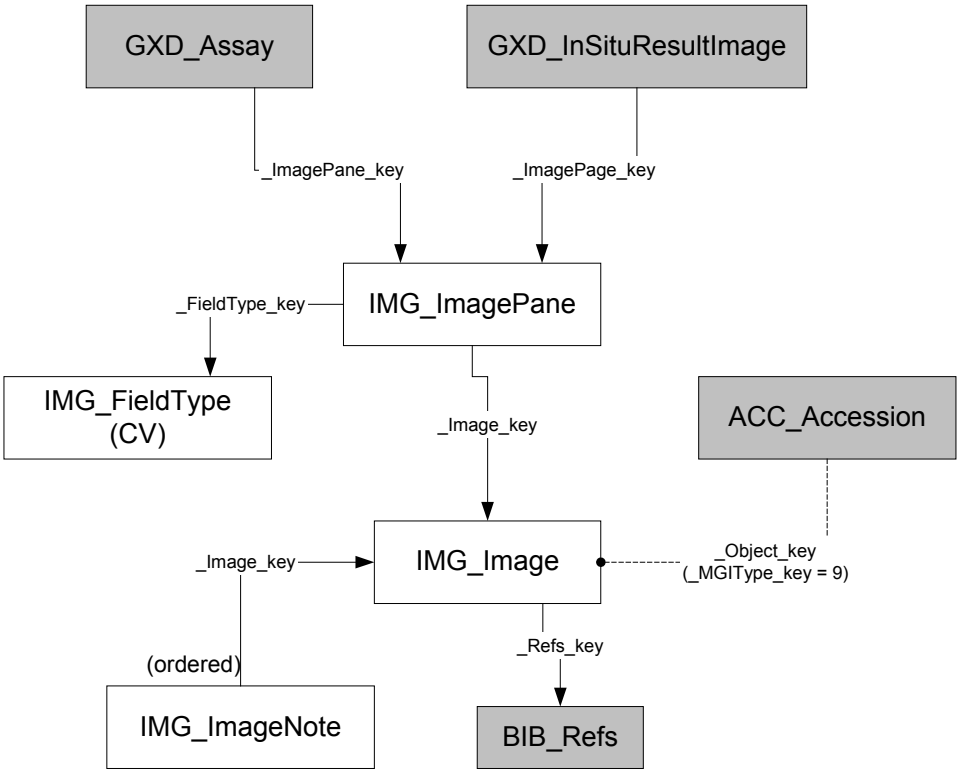
IMG_Image does not contain actual images. Rather, it describes attributes of images, and pointers to an external, web-accessible resource, called the "MGI Image Archive", and the 'pointer' is an Accession ID from this database. Accession ids look like "PIX:12345". IMG_Image records also have MGI#s. For example, the IMG_Image record with key 1009 has MGI# = MGI:1203999 and archive id = PIX:9.

The image archive server provides a CGI interface:

http://www.informatics.jax.org/pix/fetch_pixels.cgi?id=n

For example, to retrieve the image with archive id PIX:9, the URL would be:

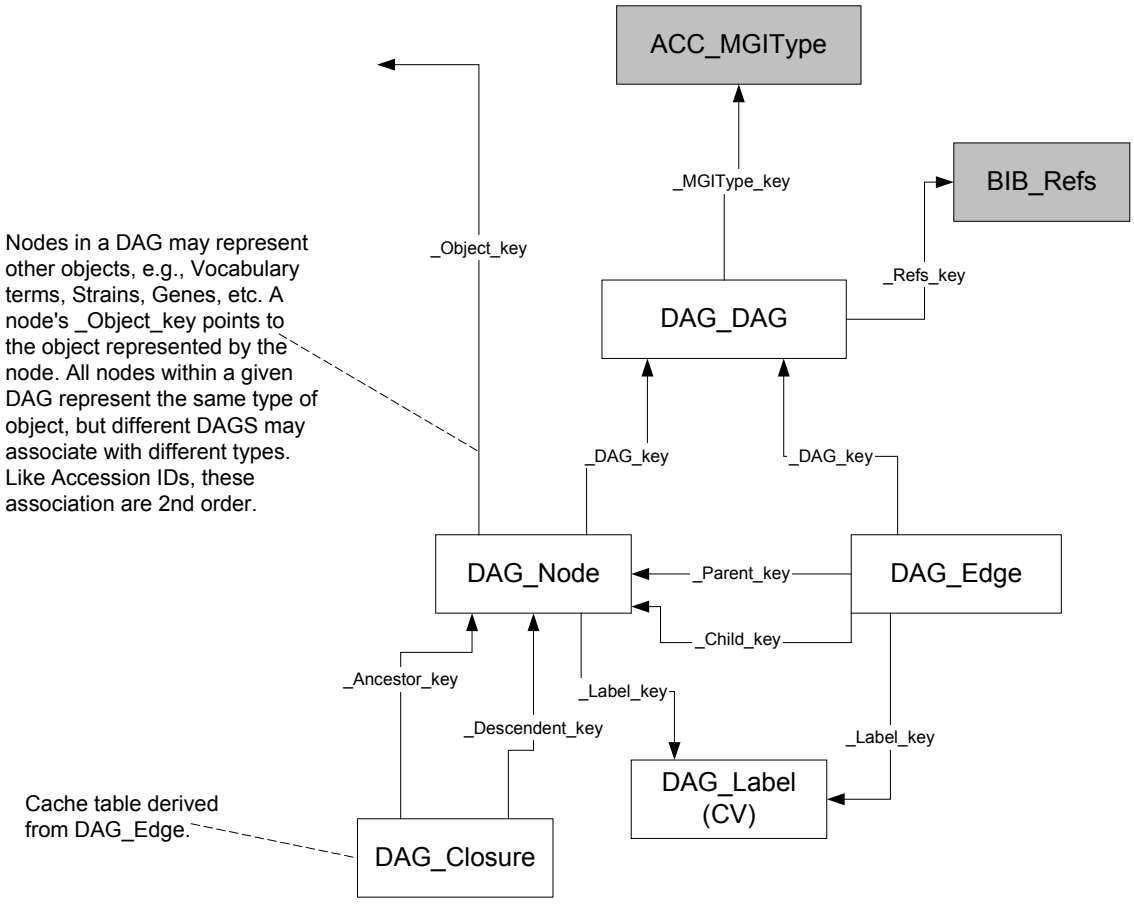
http://www.informatics.jax.org/pix/fetch_pixels.cgi?id=9



Directed Acyclic Graphs

Each record in DAG_DAG represents one directed acyclic graph. Each dag contains a set of nodes (DAG_Node) and a set of edges (DAG_Edge). A dag has a name (name) an abbreviated name (abbreviation) and a citation (_Refs_key). A node represents another MGI object (via the node's _Object_key plus the dag's _MGIType_key). An edge points from the parent node (_Parent_key) to the child node (_Child_key). Both nodes and edges may optionally have a label (_Label_key). Labels (DAG_Label) are arbitrary, uninterpreted strings.

For performance reasons, the transitive closure is maintained (DAG_Closure). Each record joins one ancestor node (_Ancestor_key) and one descendant node (_Descendent_key). The closure is non-reflexive, i.e., a node is not its own ancestor or descendant.

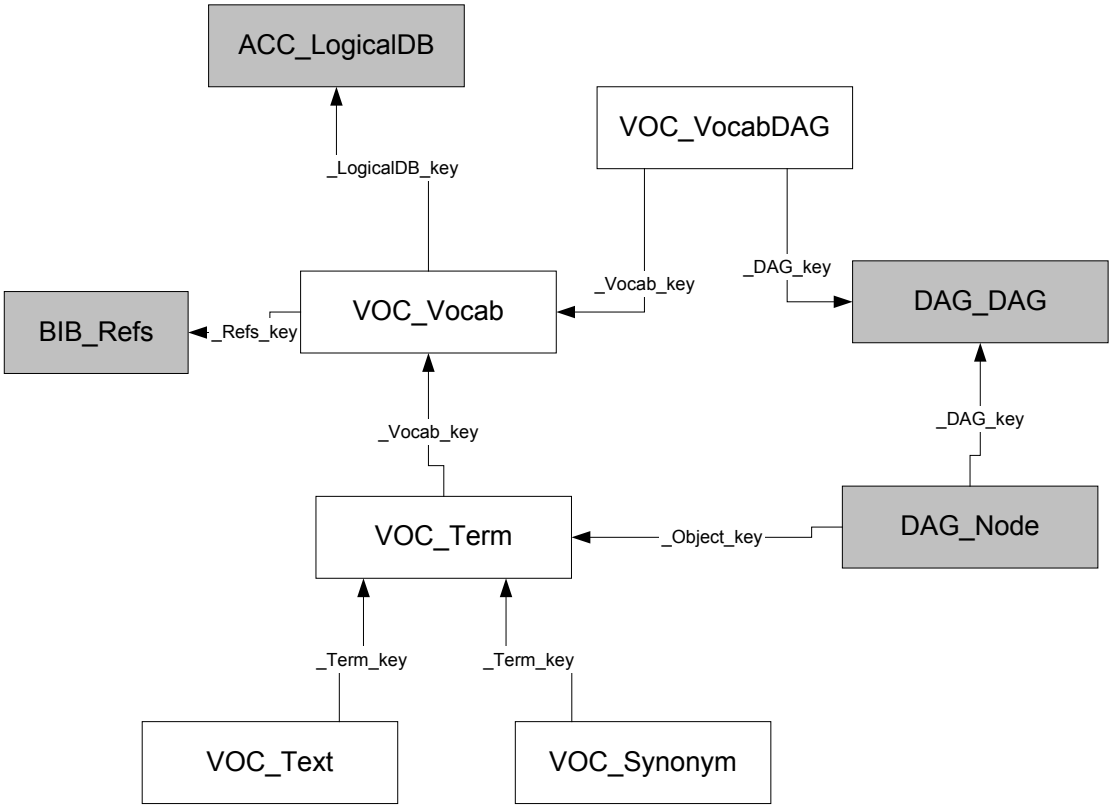


Vocabularies

A vocabulary (VOC_Vocab) is a set of vocabulary terms (VOC_Term) which may (or may not) be structured into one or more dags (DAG_DAG). The terms in a vocabulary may have accession numbers (like GO:000132). The vocabulary may have a citation (_Refs_key).

A simple vocabulary (isSimple == 1) is just a list of terms with no associated DAGs. The terms are ordered by their (sequenceNum) attribute. A structured vocabulary (isSimple==0) includes one of more DAGs (via the join table VOC_VocabDAG) that impose a structure on the terms. The dag nodes refer to the term via _Object_key, and the dag itself refers to the MGIType object for vocabulary terms.

A vocabulary term (VOC_Term) has a key (_Term_key) and label (term), and may have any number of synonyms (VOC_Synonym). A term may have a definition or other informational text (VOC_Text); the records in this table referring to the same term (_Term_key) are ordered by (sequenceNum) and concatenated into a single arbitrary-length note. A term may be marked as having been retired (isObsolete).



Annotations

An annotation (VOC_Annot) is an evidence-backed assertion of correspondence between a vocabulary term (_Term_key) and some database object (_Object_key). An annotation may either be either a positive (isNot==0) or negative (isNot==1) statement. An evidence record (VOC_Evidence) ties an annotation (_Annot_key) to an evidence code (_EvidenceTerm_key) and a citation (_Refs_key). The evidence codes used for annotations are themselves a (simple) vocabulary.

Annotations are grouped into types. An annotation type (VOC_AnnotType) has a name (name), specifies the type of object being annotated (_MGIType_key), the vocabulary whose terms are used (_Vocab_key), and the vocabulary whose terms serve as evidence codes (_EvidenceVocab_key). An annotation record indicates its annotation type (_AnnotType_key).