

Introduction to RDBMS

1. Introduction & Installation
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

Introduction to RDBMS

1. **Introduction** & Installation
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

Database Management System (DBMS) provides ...

...efficient, reliable, convenient, and safe
multi-user storage of and access to massive
amounts of persistent data.

- Massive Terabytes
- Persistent
- Safe Hardware, software, power, users
- multi-user Concurrency control
- Convenient Physical data independence
High-level query languages declarative
- Efficient Thousands of queries/updates per sec.
- Reliable 99,99999% up-time

Databases

- A **database** is a collection of data
 - numbers
 - dates
 - text or labels
 - ...
- A **Database Management System**
 - Data storage
 - Data retrieval
 - Data manipulation
 - Authentication & Authorization

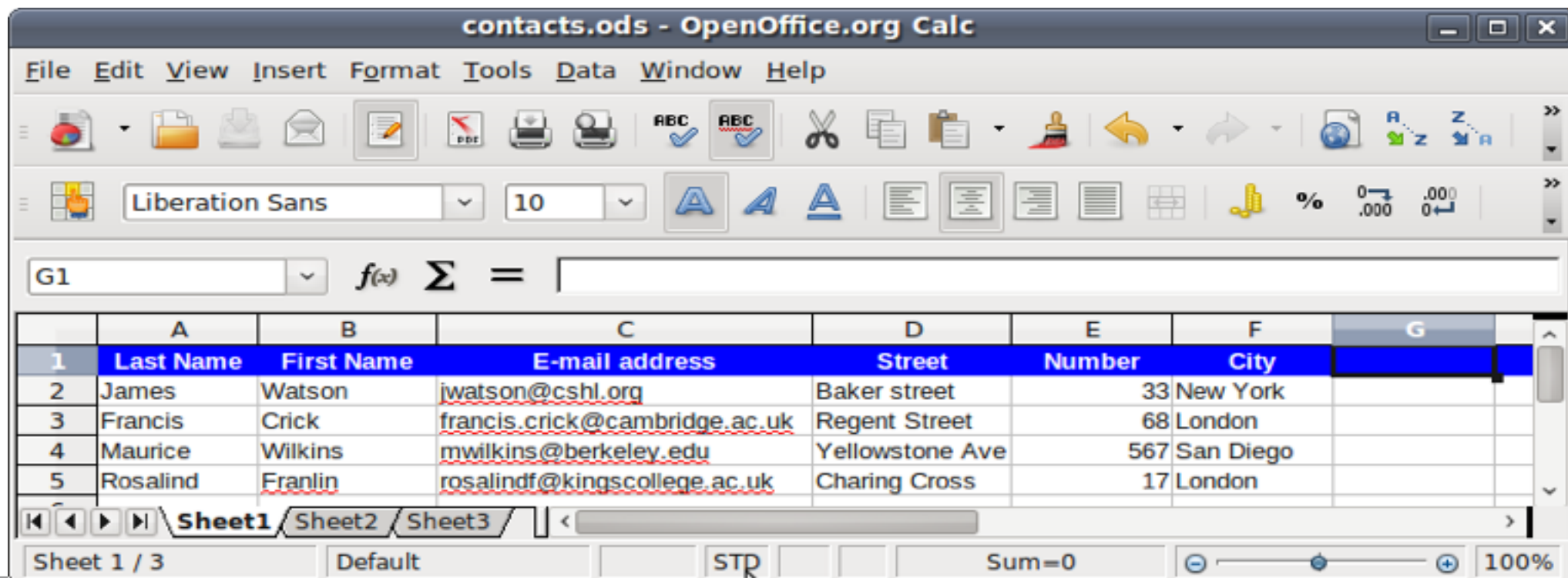


Relational Databases

Rigid structure

2 dimensional tables:

- columns (fields)
- rows (records)



The screenshot shows the OpenOffice.org Calc application window titled "contacts.ods". The spreadsheet has a table with 7 columns: Last Name, First Name, E-mail address, Street, Number, City, and an empty column G. The data is as follows:

	A	B	C	D	E	F	G
1	Last Name	First Name	E-mail address	Street	Number	City	
2	James	Watson	jwatson@cshl.org	Baker street	33	New York	
3	Francis	Crick	francis.crick@cambridge.ac.uk	Regent Street	68	London	
4	Maurice	Wilkins	mwilkins@berkeley.edu	Yellowstone Ave	567	San Diego	
5	Rosalind	Franlin	rosalindf@kingscollege.ac.uk	Charing Cross	17	London	

The interface includes a menu bar (File, Edit, View, Insert, Format, Tools, Data, Window, Help), a toolbar with various icons, and a status bar at the bottom showing "Sheet 1 / 3", "Default", "STP", "Sum=0", and "100%".

Relational Databases

Model **objects (entities)** and their **relationships**

Example of a store that sells products to customers

Entities:

Customers

—>Attributes: name, address, telephone number...

Products

—>Attributes: name, price...

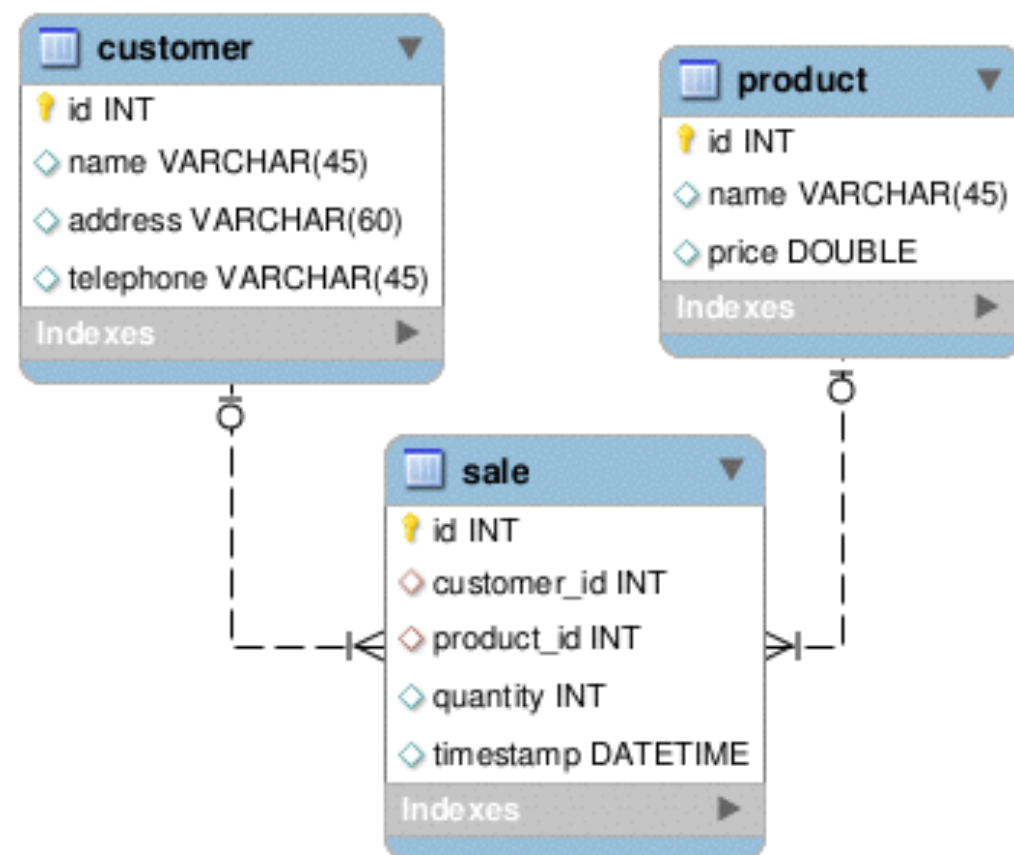
Relationships:

Sale

—>Attributes: quantity, timestamp...

Relational Databases

- MySQL Workbench:
 - graphical representation of entities and relationships
 - generates SQL statements to create database & tables



Relational Database Management Systems (RDBMS)

- Enforce data integrity:
- Honors constraints on columns
- Enforce referential integrity:
Honors constraints on relations
- See also: the 12 rules of Edgar Codd
http://en.wikipedia.org/wiki/Codd%27s_12_rules

RDBMS

Commercial products:

Oracle

DB2 (IBM)

MS SQL Server (Microsoft)

Open-source offerings:

MySQL (Oracle)

PostgreSQL

SQLite

NoSQL

Key-value stores

Berkeley DB

Document databases – unstructured data

CouchDB

MongoDB

Cassandra (FaceBook)

See also: <http://en.wikipedia.org/wiki/Nosql>

Introduction to RDBMS

1. Introduction & **Installation**
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

Installing MySQL on Linux

- For DEB based Linux distributions
(Debian, Ubuntu, ...)

```
> apt-get install mysql-server
```

- For RPM based Linux distributions
(RHEL, Fedora, CentOS, ...)

```
> yum install mysql-server
```

Installing MySQL on Windows/Mac

An installable (MSI) package is available on the MySQL site:

<http://www.mysql.com/>

Follow the 'Downloads (GA)' link

Choose 'MySQL Community Edition', 'MySQL Community Server'

Select 'Microsoft Windows' or 'Mac OS X' as platform

Running MySQL – Unix/Mac

To start / stop / restart the MySQL service UNIX:

```
> service mysql start  
> service mysql stop  
> service mysql restart
```

When starting MySQL for the first time, the system administrator is reminded that the MySQL setup is not yet secured

To start / stop / restart the MySQL service Mac:

```
> cd /usr/local/mysql-xxversion/support-files/  
> sudo ./mysql.server start|stop|restart|reload|force-  
reload|status
```

Notes on Mac installation:

<https://dev.mysql.com/doc/refman/5.1/en/osx-installation-notes.html>

Running MySQL – Unix/Mac

- To check whether or not mysql is running correctly:

```
> service mysql status (UNIX)  
mysql start/running, process 3394
```

```
> ps -ef | grep mysql  
mysql 3394 1 0 12:09 ? 00:00:00 /usr/sbin/mysqld
```

```
> netstat | grep mysql  
tcp 0 0 0.0.0.0:3306 0.0.0.0:* LISTEN 3394/mysqld
```


Running MySQL – Windows

- To check whether or not mysql is running correctly:

Check services running:

- Windows XP: Start/Control Panel/Administrative Tools/Services
- Windows 7: Start/ConfiguratieScherm/System & Beveiliging/
- Windows 10: Start/Settings (Look for services)

Exercises

- Install MySQL
- Add executable to PATH environment
- Start the service
- Check whether or not the service has been started

The MySQL monitor

- To connect or log on to a MySQL database service:

```
> mysql
```

- The MySQL monitor has many options, you can review them using:

```
> man mysql
```

or

```
> mysql --help
```

The MySQL monitor

- The most important options are:

> `mysql [options] [database]`

`-u uname | --user=uname`

default: UNIX account

`-p [pwd] | --password[=pwd]`

default: <none>

if *pwd* not given, prompt for password

`-h hname | --host=hname`

default: localhost

`-P prt | --port=prt`

default: 3306

Exercises

Connect to the database and execute the following SQL statements:

```
mysql> select current_user;
```

```
mysql> show databases;
```

For Mac (initial password reset necessary for root account):

```
mysql> set password =  
password( 'new_password' );
```

The MySQL monitor

Once connected to the database server, you can execute SQL statements:

```
mysql> statement;
```

Every SQL statement should end with a semi-colon (;)

Exercises

Create a database user:

Choose database account, you are free to choose the password,
the hostname is localhost
try to remember userid/password throughout training :-)

```
mysql> create user 'new_user';  
mysql> quit;  
> mysql -u 'new_user';  
mysql> set password = password('new_password');  
mysql> quit;  
> mysql -u 'new_user' -p;
```

Try to connect as this user and execute the following SQL
statements:

```
mysql> select current_user;  
mysql> show databases;
```

Getting Help

An extensive help system is available in the MySQL monitor:

```
mysql> help
```

This gives an overview of commands you can use to customise the output

You can get help on any function or statement:

```
mysql> help contents
```

This shows you the broad topics of available help. You can drill down into any of these topics

Database User Privileges

The created user has very limited privileges. To grant privileges *prv* on table *tbl* in database *db*, you need to execute the following statement:

```
mysql> grant prv on db.tbl  
      to user@host;
```

Some convenient wild cards:

To grant all privileges, specify *all as prv*

To include all databases, specify ** as db*

To include all tables, specify ** as tbl*

The given database and table names do not have to exist (yet)

Getting Help - Demo

How to get help for creating users:

```
mysql> help
```

```
mysql> help contents
```

```
mysql> help account management
```

```
mysql> help create user
```

How to use less as a pager:

```
$ export PAGER=/usr/bin/less
```

```
$ mysql
```

```
mysql> pager
```

```
PAGER set to '/usr/bin/less'
```

Introduction to RDBMS

1. Introduction & Installation
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

SQL

SQL (Structured Query Language) is the language a RDBMS provides for:

Data definition ([DDL](#))

CREATE TABLE, DROP DATABASE

Data manipulation ([DML](#))

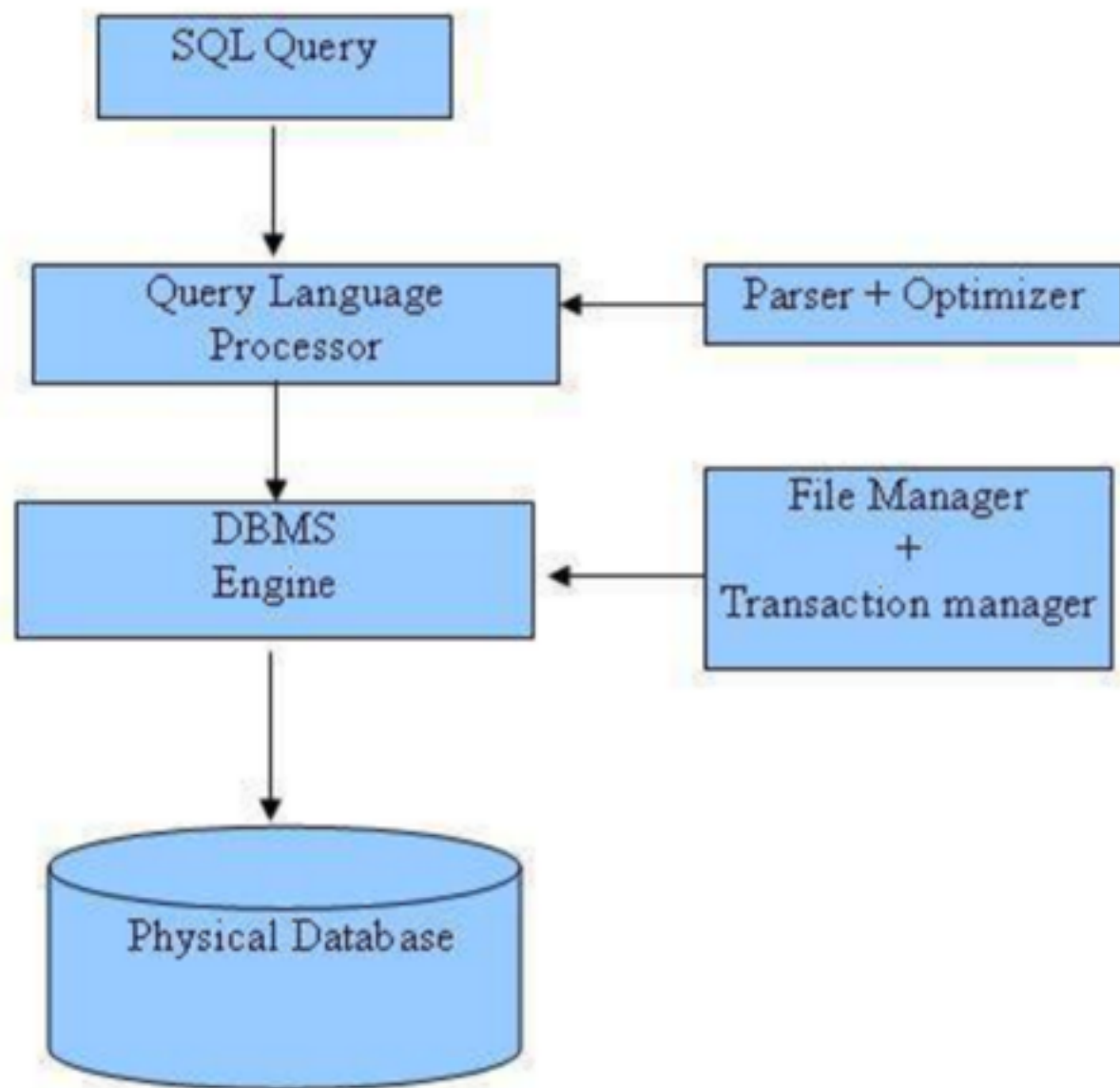
SELECT, INSERT, DELETE, UPDATE

Data control ([DCL](#))

GRANT, REVOKE

SQL is a ANSI/ISO standard - MySQL implements a broad subset of ANSI SQL 99

SQL Architecture



DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
SELECT	Retrieves certain records from one or more tables
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

Table - Row (tuple) - Column

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Constraints: Not NULL, Default, Unique, Primary Key, Foreign Key, Check

The MySQL monitor (again)

There are a number of ways to execute SQL statements using the MySQL monitor:

Interactively

```
$ mysql [database]  
mysql> stmt;
```

From the command line

```
$ mysql [database] -e 'stmt'
```

From a file or a pipe (stdin)

```
$ mysql [database] < stmt_file  
$ cat stmt_file | mysql [database]
```

Creating a Database

Only database users with sufficient privileges can create databases (eg root@localhost)

From the command line:

```
$ mysqladmin [opt] create dbname
```

mysqladmin shares the same command line options for user, password, host and port as mysql

From within the MySQL monitor:

```
mysql> create database dbname
```


Exercises

As root@localhost, create database '*biodb*'

Grant all privileges to the database user you created before:

```
mysql> grant all on biodb.*  
       to user@localhost;
```

Download the file *biodb1.sql* - you might want to take a look at the contents (!)

Execute all SQL statements in this file

Hierarchy

A single MySQL service can have multiple databases

```
mysql> show databases;
```

A particular database db can have multiple tables

```
mysql> use db;
```

```
mysql> show tables;
```

A particular table tbl can have multiple columns or fields

```
mysql> show columns from tbl;
```

```
mysql> show create table tbl;
```

Exercises

- Connect to the database service as a normal user
- What databases do you see?
- What tables are defined in *biodb*?
- What are the column names?

Retrieving rows

- To read data from the database, you use the select SQL statement:

```
select columnlist from src;
```

- With

- *columnlist*:

- list of columns separated with a comma
 - use * to retrieve all columns

- *src*:

- single table or multiple tables joined together
 - subquery
 - view

Retrieving rows - examples

- To show all data from table modorg:

```
select * from modorg;
```

- To show all model organisms:

```
select genus, species  
from modorg;
```

Sorting rows

When using select statements, the data is displayed in no particular order

To sort on one or more columns, use the

```
order by col1 [asc|desc]  
       [, col2 [asc|desc] ...]
```

clause

With

colX: a column or column alias

asc: ascending (default) order

desc: descending order

Exercises

- Show the names (genus & species) of all model organisms in the order of the publishing date of the draft
- Show the names (genus & species) of all model organisms sorted by the number of chromosomes (most chromosomes on top) and then alphabetically by name

Calculated rows

You can add columns in a query that calculate some value using other columns of the same row

A multitude of functions and operators are available: see help in the MySQL monitor

Examples:

```
mysql> select 6 * 7;
```

```
mysql> select  
    concat(class, " ", genus)  
    from modorg;
```

```
mysql> select now();
```


Calculated rows - numbers

See **help numeric functions**

Operators:

$+$, $-$, $*$, $/$, $\%$

Functions:

`sqrt(x)`, `power(x, y)`, ...

`exp(x)`, `ln(x)`, ...

`sin(x)`, `cos(x)`, ...

`round(x)`, `ceil(x)`, `floor(x)`, ...

`rand()`, `rand(x)`

Calculated rows - strings

See **help string functions**

Functions:

`length(s)`

`concat(s1, ...)`

`upper(s), lower(s)`

`trim(s), ltrim(s), rtrim(s)`

`substr(s, ...)`

`reverse(s)`

`truncate(s)`

Calculated rows - dates

See **help date and time functions**

Functions:

`currentdate() , now()`

`year(d) , month(d) , week(d)`

`dayofmonth(d) , dayofweek(d)`

`hour(d) , minute(d) , second(d)`

Exercises

Show

- model organism full name (as a single column)

- genome size (as Gb), rounded to 4 digits

- average chromosome size

- publication year

of all rows sorted by average chromosome size
(largest average chromosome size on top)

Column aliases

Columns can be renamed (aliased):

```
select col [as] alias ...
```

The aliases can be used in the order by clause

Exercises

Show

model organism full name (as a single column)

as name

average chromosome size as avgsize

publication year

as pubyear

of all rows sorted by avgsize (largest size on top)

Filtering rows

To select only those rows that meet certain criteria, use the where clause:

```
select columnist  
from sir  
where condition(s)  
[order by sortcol];
```

With **condition(s)**:

- one or more conditions, combined with not, and, or, xor
- only the rows for which the condition(s) evaluate(s) TRUE are selected
- you can not use column aliases in condition(s)

Filtering rows - conditions

See **help comparison operators**

Numerical comparison operations:

`=`

`!=` or `<>`

`<`, `<=`, `>`, `>=`

between `x` and `y` (inclusive)

Example:

select all organisms with more than 10 chromosomes:

```
select genus, species from modorg
where nchr > 10;
```


Filtering rows - conditions

String comparison operations:

=

!= or <>

<, <=, >, >= (lexical)

like "pattern"

matches a pattern:

_ (a single character - cfr shell ?)

% (0 or more characters - cfr shell *)

rlike "regex" [MySQL]

matches a regular expression

Example - select all mammals:

```
select genus, species from modorg
where class = "mammals";
```

Filtering rows - conditions

Dealing with NULL-values

Testing for NULL-ness:

```
select ... where col is null ...  
select ... where col is not null ...
```

Substitution of NULL-values:

```
select ifnull(col, value) ...
```

this function returns:

col if *col* is not NULL

value if *col* is NULL

Example:

```
select genus, species,  
       ifnull(nchr, 0)  
from modorg;
```

Filtering rows - conditions

Boolean logic:

`not x`

evaluates TRUE if *x* is FALSE

`x and y`

evaluates TRUE if both *x* and *y* are TRUE

`x or y`

evaluates TRUE if *x* or *y* is TRUE, or both

`x xor y`

(eXclusive OR)

evaluates TRUE if either *x* or *y* is TRUE, but not both

Exercises

Select all mammals with genomes published after 2005

Select all organisms that have a average chromosome size between 10 and 100 Mbp

Select all organisms whose genus starts with A, B, C, D, or E

Filtering rows - duplicates

To eliminate duplicate rows, use

```
select distinct cols from ...
```

Each combination of *cols* is unique

Filtering rows - limiting output

To limit the number of rows in a result set, use

```
select ... limit n [offset r]
```

The result set is limited to a maximum of *n* rows

If an offset *r* is given, the first *r* rows are skipped

It usually makes little sense to use limit without order by

Exercises

Give an overview of all organism classes in the dataset (sorted alphabetically)

Show the organism names of the top 3 largest genome sizes

Aggregation

So far, queries have concentrated on particular rows

Sometimes, you need to calculate a single result across multiple rows

e.g.: what is the maximum genome size

SQL allows you to

- specify criteria to group rows together
- calculate a single value per group
- filter grouped data

Aggregation - functions

See: help functions and modifiers for use with
GROUP BY

Functions:

```
count(col), count(*),  
count(distinct col)  
sum(col)  
min(col), max(col)  
avg(col), stddev(col),  
variance(col)
```

Exercises

All queries below return a row count.

What is the result? Why?

```
select count(*) from modorg;  
select count(nchr) from modorg;  
select count(class) from modorg;  
select count(distinct class)  
from modorg;
```

How many mammals are in the database?

Aggregation - groups

The group by clause sorts data into groups for the purpose of aggregation:

```
select [col,] aggregatefunctions  
from src  
[where cond]  
group by col  
[order by ...];
```

All rows with the same value in *col* are grouped

For each group, the aggregate function is calculated
It usually makes no sense to select any columns other than *col*

Exercises

How many organisms are present in the dataset for each class?

Note the sort order.

Show the minimum and maximum genome sizes for each class.

Take only those organisms into account for which the genome sizes are known.

Sort the results such that the biggest maximum genome size is on top.

Aggregation - filtering

It is possible to query filter results based on the results of aggregate functions, using the **having** clause:

```
select [col,] aggregatefunctions  
from src  
[where cond1]  
group by col  
having cond2  
[order by ...]
```

Column aliases can be used in *cond2*

Execution order

- 1> Input columns are determined
- 2> where: input columns are filtered
- 3> group by: sorting & grouping
of filtered input
- 4> aggregation functions are calculated
- 5> having: aggregation results are filtered
- 6> order by: output is sorted
- 7> limit / offset: output is chopped

Exercises

For each class with more than 1 organism, show the average number of chromosomes.

Sort the result such that the biggest average is on top.

Database upgrade

To prepare for the next part:

Download biodb2.sql

Delete the database biodb:

```
mysql> drop database biodb;
```

Recreate the database biodb:

```
mysql> create database biodb;
```

Create the tables and insert the data

```
$ mysql biodb < biodb2.sql
```


Joins

Relational databases model entities and their relationships

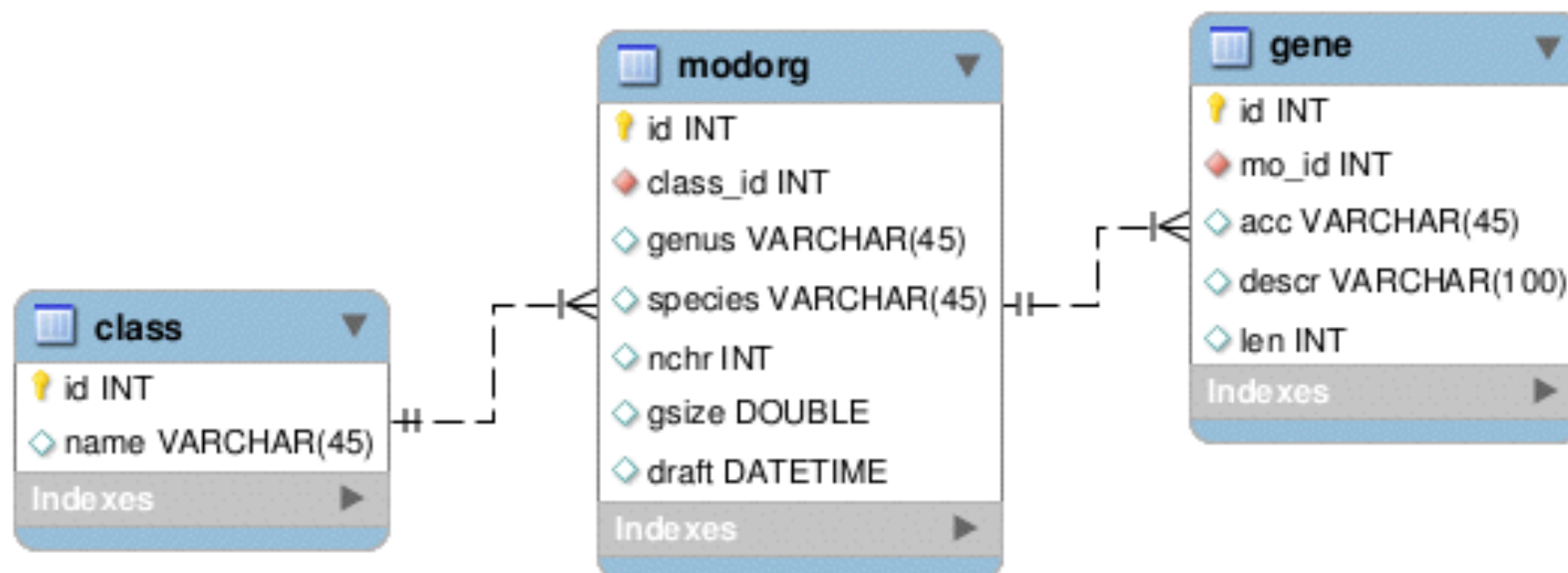
Different entities: different tables

Joins allow you to combine information across different tables

Joins

modorg.class_id is a foreign key that references class.id

gene.mo_id is foreign key that references modorg.id



Joins - the Cartesian product

When you specify multiple tables in a query, the database server will generate all possible combinations: the Cartesian product

Example:

table class has 6 rows

table modorg has 10 rows

query

```
select * from modorg, class
```

has 60 rows

Joins

One way to look at joins is to consider it a filtered (think: where) Cartesian product:

```
select * from modorg, class
where modorg.class_id = class.id
```

Note:

- column name id is present in both tables class and modorg - to avoid ambiguity, the column name is qualified: class.id
- column name modorg.class_id is also qualified, but since there is no ambiguity, this is not strictly necessary

Joins

Alternative syntax:

```
select *  
from modorg [inner] join class  
    on modorg.class_id = class.id;
```

Both syntactical forms are considered completely equivalent - there is no compelling reason to prefer one above the other, it is just a matter of taste.

Joins - column aliases

When specifying columns in the query, care must be taken to avoid ambiguity:

```
mysql> select id, name, genus, species
        from modorg, class
        where modorg.class_id = class.id;
```

ERROR 1052 (23000):

Column 'id' in field list is ambiguous

Ambiguous columns must be qualified. And additionally you can choose an alias:

```
mysql> select modorg.id as mo_id,
        name, genus, species
        from modorg, class
        where modorg.class_id = class.id;
```

Joins - data source aliases

In some cases, you need to join a table with itself or you select data from a subquery

In this case it can be handy (or even necessary) to alias the data source:

```
select a.col, b.col  
from src1 [as] a, src2 [as] b  
where ...
```

Joins - data source aliases

Consider the (at first sight) simple question:

For each class, give the class name, organism name and date of the organism that was sequenced first

A part of the answer is this:

```
select class_id, min(draft) as dr from  
modorg  
group by class_id;
```

To add the class name: join with table class

To add the organism name: join with table modorg

Joins - data source aliases

Add the class name:

```
select name, dr
from
    (select class_id, min(draft) as dr
     from modorg group by class_id) as s,
    class
where s.class_id = class.id;
```

Add the organism name:

```
select name, genus, species, dr
from
    (select class_id, min(draft) as dr
     from modorg group by class_id) as s,
    class, modorg
where s.class_id = class.id
    and s.dr = draft;
```

Exercises

For all rows in table *gene*, show

organism name

class name

accession

length

description of the gene

Outer joins

Inner join:

```
select class.name, genus, species, gene_acc  
from  
  class  
  inner join modorg oclass.id=modorg.class_id  
  inner join gene on gene.mo_id = modorg.id;
```

Left outer join:

```
select class.name, genus, species, gene_acc  
from  
  class  
  inner join modorg oclass.id=modorg.class_id  
  left outer join gene on gene.mo_id = modorg.id;
```

Reusing queries - views

This last exercise is actually very useful as a data source itself:

- It could be used as such (as a subquery) - but this would end up being inconvenient.
- The query as such can be saved as a special table-like object, called a view.

Syntax:

```
create view viewname as  
select ...
```

Reusing queries - views

You can refer to a view by specifying its name in the from clause:

```
select ...  
from viewname  
where ...  
order by ...
```

Although there are circumstances where you can change a view, most views are to be considered as read-only data sources

Exercises

Create a view (*genevw*) from the query in the previous exercise

Select all genes containing hemoglobin in the description and sort the result set by gene length.

Next:

What is the minimum and maximum gene length?

What is the average gene length?

And the standard deviation?

Does the view show up when using

```
mysql> show tables;
```

Database backup

You can dump a complete database into a text file:

```
$ mysqldump [opt] db > db.sql
```

This includes:

- statements for creating the database
if the option `--databases` is used
- statements for creating tables, views, ...
- statements for inserting data

To restore the database (you may need to create it first):

```
$ mysql db < db.sql
```

Introduction to RDBMS

1. Introduction & Installation
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

Database schema

Although you can execute DDL commands from the MySQL monitor directly, this is not often done

There are tools that allow you to design a schema graphically and generate the **CREATE TABLE ...** statements

Examples:

MySQL workbench

HeidiSQL

MySQL workbench

Available as a standard package on some Linux distros (eg Fedora)

Available as Windows MSI, Linux DEB or RPM package or source archive from <http://dev.mysql.com/downloads/workbench/>

To install a DEB package:

```
# dpkg -i package.deb
```

To install a RPM package:

```
# rpm -Uvh package.rpm
```

To install on Windows/Mac:

```
# double click package.msi or package.dmg
```

Database schema

Once the schema is designed, MySQL workbench can generate a 'script' containing all SQL statements to create the tables and other objects:

MySQL Workbench Model

Create Model

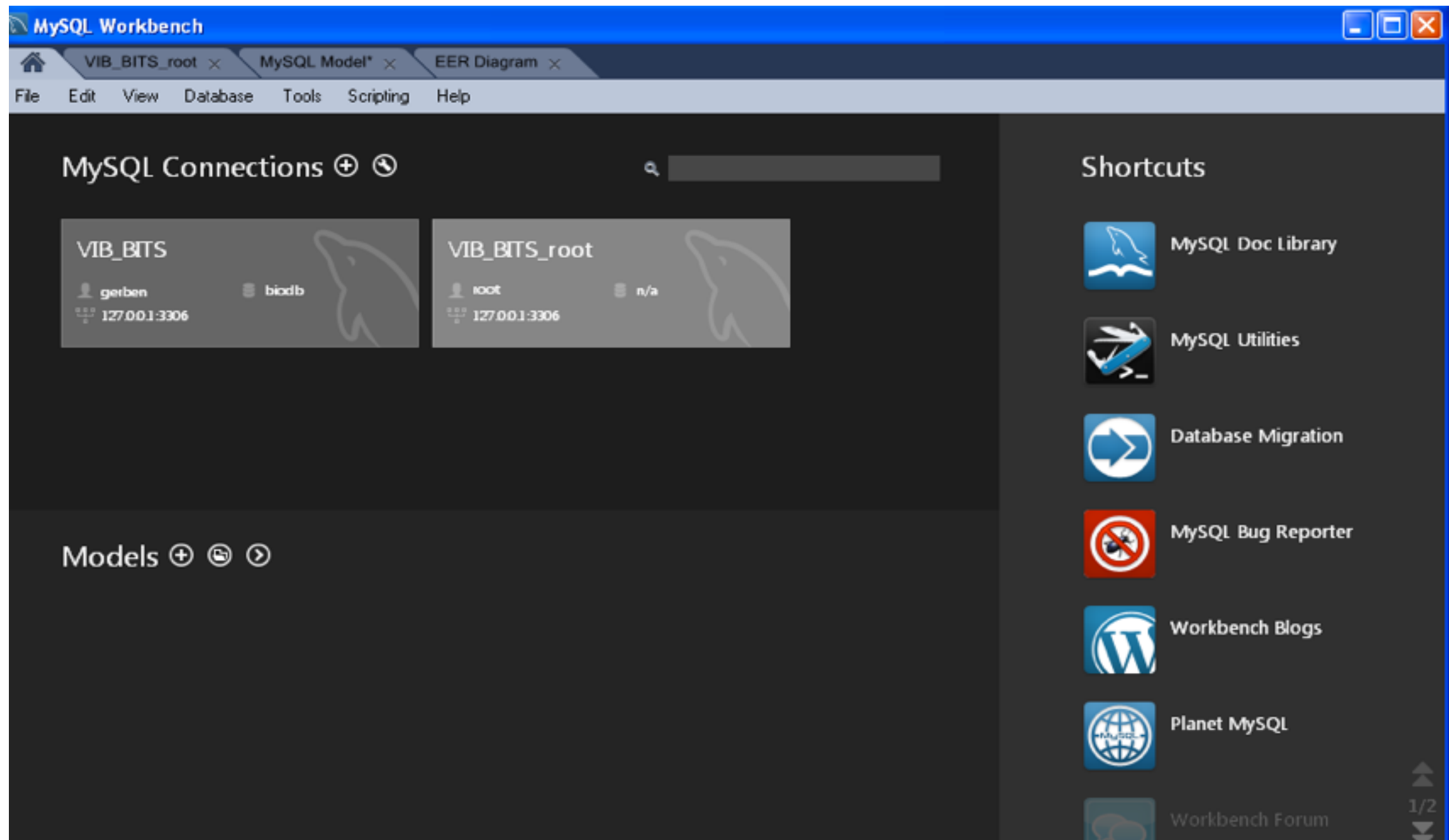
File ->

Export ->

Forward Engineer SQL CREATE Script

This 'script' can be executed as usual from the MySQL monitor

MySQL workbench - demo



Inserting rows

To populate tables, use the INSERT SQL statement:

```
mysql> insert into tbl
      (col1, col2, ...)
values
      (val1, val2, ...) [,
      (valx, valy, ...) , ...]
```

With:

- tbl the name of the table
- col1, col2, ... a list (subset) of column names
- val1 value for col1
- val2 value for col2

Inserting rows - examples

Example (biodb version 1)

```
insert into modorg  
(id, class, genus, species, nchr, gsize,  
draft)  
values  
(1, "Bacteria", "Escherichia", "coli", 1,  
4.639, "1997-09-05 00:00:00")
```

Note that strings and dates have to be properly quoted

Inserting rows

You can leave out the column list if

- a value is given for all columns
- the values are specified in the order dictated by the schema

```
mysql> insert into tbl  
      values  
      (val1, val2, ...) [,  
      (valx, valy, ...) , ...]
```

Changing rows

To change one or more rows, use the UPDATE SQL statement:

```
mysql> update tbl  
set col1=expr1 [, col2=expr2, ...]  
[where cond]
```

With:

- tbl* the name of the table
- col1* the column to change
- expr1* the new value
- cond* the row filter - if unspecified, all rows of the table will be updated

Changing rows - examples

To change the C elegans number of chromosomes to 7:

```
update modorg  
set nchr = 7  
where genus = "caenorhabditis"  
and species = "elegans";
```

To change the draft date to the current date:

```
update modorg  
set draft = now();
```

Deleting rows

To remove rows, you use the DELETE SQL statement:

```
delete from tbl  
[where cond]
```

With:

- tbl* the name of the table
- cond* the row filter - if unspecified, all rows of the table will be deleted
- note: since you can only remove entire rows, there is no need to specify column names

Deleting rows - examples

To remove all in model organisms with a genome publishing date before 2000:

```
delete from modorg  
where year(draft) < 2000;
```

Introduction to RDBMS

1. Introduction & Installation
2. Hello data? - SQL
3. DB Design: MySQLWorkBench
4. Biological DBs using RDBMS

Known examples

Ensembl:

<http://www.ensembl.org/downloads.html>

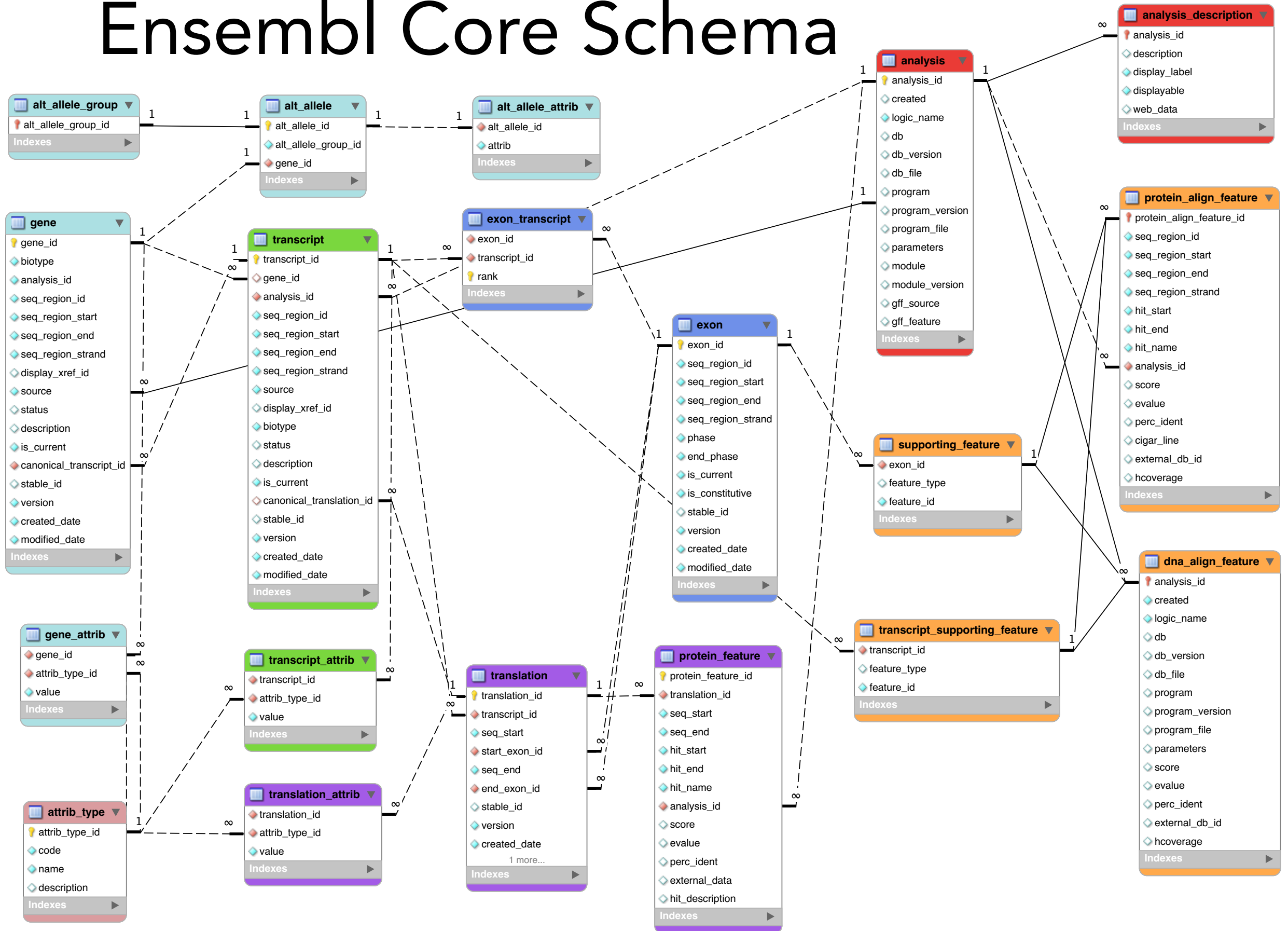
UCSC Table Browser:

<https://genome.ucsc.edu/cgi-bin/hgTables>

BioMart:

<http://www.ensembl.org/biomart/>

Ensembl Core Schema



UCSC Table Browser

Genomes Genome Browser Tools Mirrors Downloads My Data Help About Us

Table Browser

Use this program to retrieve the data associated with a track in text format, to calculate intersections between tracks, this form, the [User Guide](#) for general information and sample queries, and the OpenHelix Table Browser [tutorial](#) for examining the biological meaning of your set through annotation enrichments, send the data to [GREAT](#). Send data to [G](#) downloaded in their entirety from the [Sequence and Annotation Downloads](#) page.

clade: **genome:** **assembly:**

group: **track:**

table:

region: ☒ genome ☐ DE Pilot regions ☐ position

identifiers (names):

filter:

intersection:

correlation:

output format: ☐ [Galaxy](#) ☐ [GREAT](#) ☐ [GenomeSpace](#)

output file: (leave blank to keep output in browser)

file type returned: ☒ text ☐ gzip compressed

To reset all user c (including custom tracks), [click here](#).

Species List:

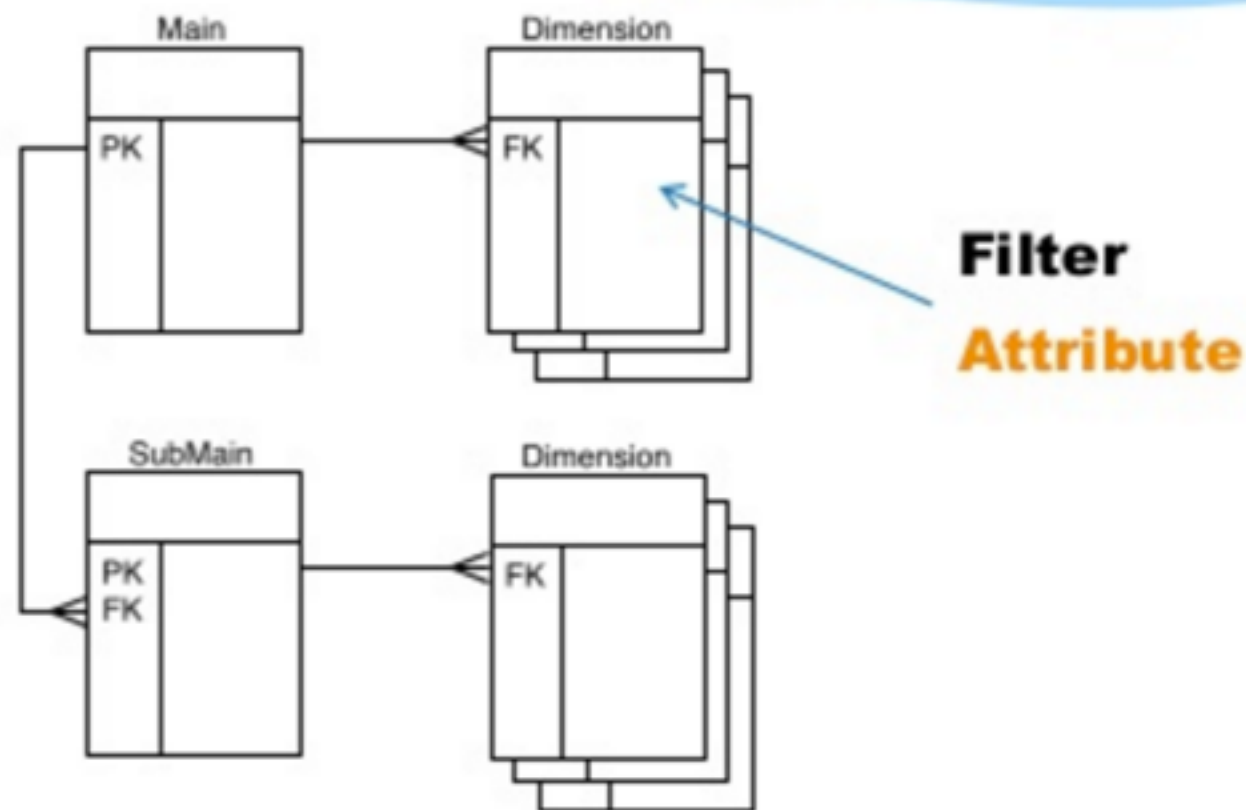
- Human
- Mouse
- Alpaca
- Armadillo
- Baboon
- Bonobo
- Bushbaby
- Cat
- Chimp
- Chinese hamster
- Cow
- Dog
- Dolphin
- Elephant
- Ferret
- Gibbon
- Gorilla
- Guinea pig
- Hedgehog
- Horse
- Kangaroo rat
- Manatee
- Marmoset
- Megabat
- Microbat
- Minke whale
- Mouse lemur
- Naked mole-rat
- Opossum
- Orangutan
- Panda
- Pig
- Pika
- Platypus
- Rabbit
- Rat
- Rhesus
- Rock hyrax
- Sheep
- Shrew
- Sloth
- Squirrel
- Squirrel monkey
- Tarsier
- Tasmanian devil

BioMart Reverse Star Schema

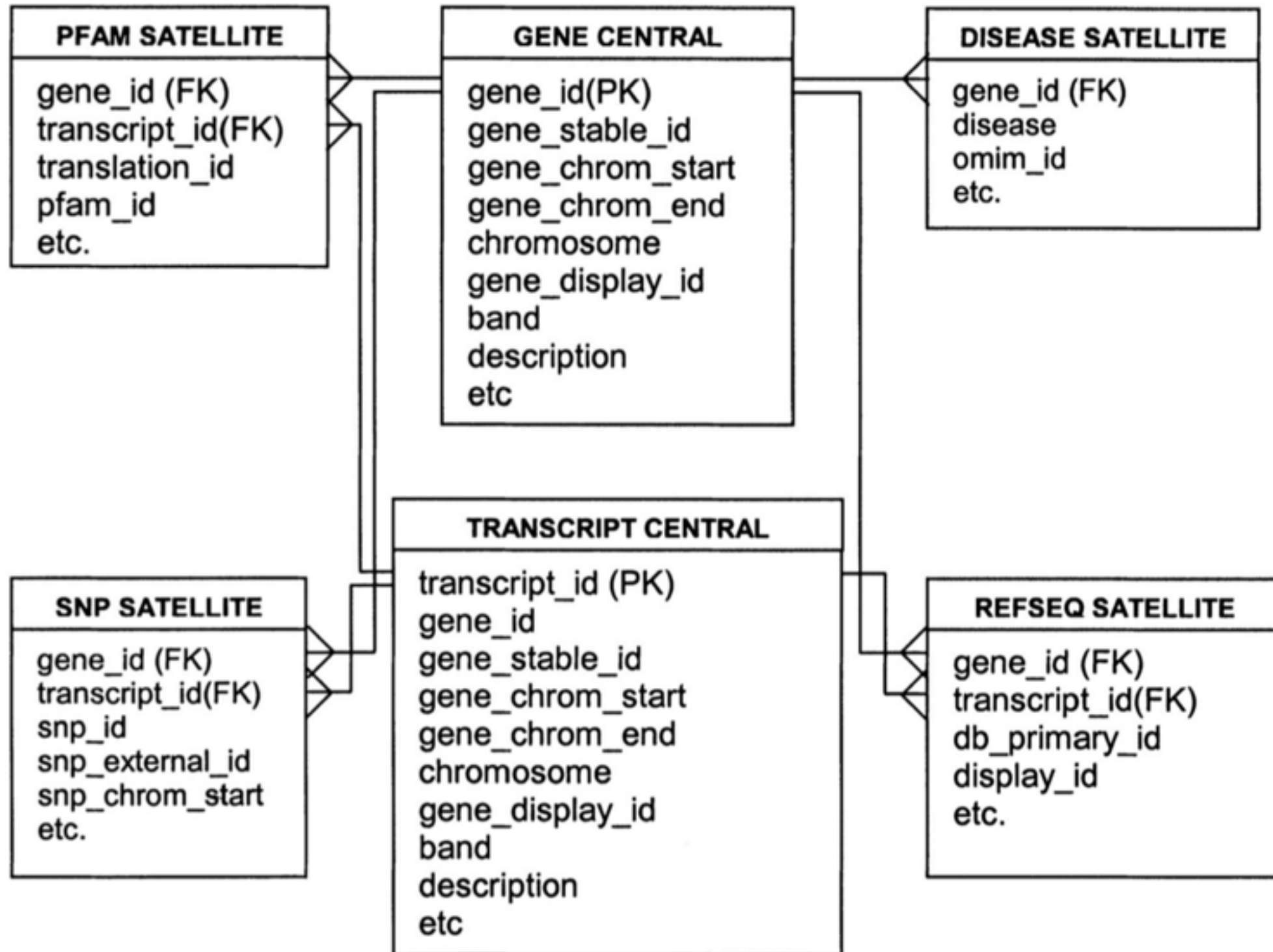
BioMart schema
– “reversed star”

Dataset

- For fast retrieval of large quantities of descriptive data
- Main table columns are mainly query constraints



EnsMart



Useful links for further reading

<http://www.tutorialspoint.com/sql/>

<http://www.sqlcourse.com/>

<http://shop.oreilly.com/product/9780596009762.do>

(O'Reilly: SQL CookBook or MySQL CookBook)