Building Biological Databases

Sheldon McKay

What is a database?
Outline

• Existing databases
• Types of databases
• SQL and relational database basics
• Normalization
• Denormalization
• Dealing with very dense data
• Factors in database choice

Why do I need a database?

• Keeping data in flat files has limits
  – requires parsers
  – little or no organizing principles
  – inefficient and awkward to query
Existing Biological Databases

• Don’t reinvent the wheel; has someone already solved your problem?
  • Genome annotation
    – Ensembl, Bio::DB::GFF, etc.
  • Model organism and general purpose
    – Chado
  • Maps
    – CMap, ArkDB
  • Pathways
    – Reactome, Panther, BioCyc
  • Warehousing/data mining
    – Biomart

However there is still a need for…

– Create lab databases, such as LIMS

– Design databases for the special and unique data that only your group possesses

– Extend and improve existing models and schemas to suit your needs.
Selected Data Models

- Flat Model
- Hierarchical Model
- Object Model
- Relational Model

Flat (Table) Model

- A single, two-dimensional array of data elements
- All members of a column are assumed to be similar values
- All members of a row are assumed to be related to one another
- Example: spreadsheet
Hierarchical Model

- Data organized into a tree-like structure
- Represents relationships like parent/child

<table>
<thead>
<tr>
<th>System_id</th>
<th>System</th>
<th>Parent_system</th>
<th>part</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>electrical</td>
<td>10</td>
<td>ABS relay</td>
</tr>
<tr>
<td>20</td>
<td>drive-train</td>
<td>10</td>
<td>flasher</td>
</tr>
<tr>
<td>30</td>
<td>brakes</td>
<td>10</td>
<td>body control module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>transmission</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>head gasket</td>
</tr>
</tbody>
</table>

Object Model

- Attempt to apply object-oriented programming principles, such as encapsulation and polymorphism, to database organization
- Reached its peak of popularity in the 1990s
- Successfully applied to genome annotation database AceDB
Relational Model

- A mathematical model, originally proposed by E. F. Codd in 1970, defined in terms of predicate logic and set theory

- Relational databases implement a model that is an approximation to Codd's mathematical model

Relational Model

- A relational database contains multiple tables, each similar to the one in the "flat" database model

- Relationships between tables are not defined explicitly; instead, keys are used to match up rows of data in different tables

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene</td>
<td>gene_process</td>
<td></td>
</tr>
<tr>
<td>gene_id</td>
<td>gene_id</td>
<td>process</td>
</tr>
</tbody>
</table>

11
DataBase Management Systems

• A complex set of software programs that controls the organization, storage and retrieval of data in a database

• In principal, this term is not interchangeable with the database itself.
Common Relational DBMSs

- Free/open source
  - MySQL
  - PostgreSQL

- Commercial
  - Oracle
  - Sybase
  - DB2
  - Microsoft SQL Server

Person

Command Line Interface

Application

Programmatic Interface (eg DBI)

DATABASE MANAGEMENT SYSTEM
Relational database basics

- SQL
- Tables and Schemas
- Data Description Language
- Adding data
- Queries

Structured Query Language

- Structured English Query Language ("SEQUEL") was designed to manipulate and retrieve data stored in IBM’s original relation database “System R”.

- The acronym SEQUEL was later condensed to SQL due to trademark issues.

- SQL was adopted as a standard by ANSI (American National Standards Institute) in 1986.

- According to ANSI, the official pronunciation for SQL is /ɛs kjuː ɛl/ (but it also common to pronounce it sequel).

- Although SQL is a defined standard, there are many flavors, some proprietary
What can you do with SQL?

• Data retrieval
  SELECT

• Data manipulation
  INSERT, UPDATE, MERGE, TRUNCATE, DELETE

• Data definition
  CREATE, DROP

• Data transaction
  START TRANSACTION, COMMIT, ROLLBACK

• Data Control
  GRANT, REVOKE

Tables and queries

• Data stored in tables
  – Each row is a tuple (ordered list)

• A collection of table definitions is called a schema

• Data are entered and retrieved using SQL statements
A table

Table: ‘favorite_color’
4 columns
2 rows (tuples)

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Tom</td>
<td>Jones</td>
<td>blue</td>
</tr>
<tr>
<td>101</td>
<td>John</td>
<td>Smith</td>
<td>red</td>
</tr>
</tbody>
</table>

SQL Data Description Language

```sql
CREATE TABLE favorite_color (  
id INTEGER,  
first_name VARCHAR(255),  
last_name VARCHAR(255),  
color VARCHAR(255)  
);
```
Some SQL Domains

- INTEGER
- CHAR
- VARCHAR
- FLOAT
- DOUBLE PRECISION
- TEXT
- DATE

Some constraints:
- UNIQUE
- NOT NULL

Adding data using SQL INSERTs

```sql
CREATE TABLE favorite_color (  
id INTEGER,  
first_name VARCHAR(255),  
last_name VARCHAR(255),  
color VARCHAR(255)  
);

INSERT INTO favorite_color VALUES  
(100, 'Tom', 'Jones', 'blue');
INSERT INTO favorite_color VALUES  
(101, 'John', 'Smith', 'red');
```
Retrieving data

- Data can be retrieved using SQL queries
- SQL can be used interactively, or programmatically (e.g., via DBI)
- SQL is based (loosely) on relational algebra
  - A set of operations for manipulating relations
  - Main operations:
    - PROJECT
    - RESTRICT
    - JOIN

Projection

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>T cell differentiation</td>
</tr>
</tbody>
</table>

SELECT symbol FROM gene;
Restriction

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>T cell differentiation</td>
</tr>
</tbody>
</table>

SELECT *  
FROM gene  
WHERE species='Ovis aries'

Combining operators: restrict +project

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>T cell differentiation</td>
</tr>
</tbody>
</table>

SELECT symbol, species  
FROM gene  
WHERE process='Immune response'
Closure

• The result of a relational query is always a relation
• This allows queries to be composed
  – the result of a query can be used in another query

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>T cell differentiation</td>
</tr>
</tbody>
</table>

SELECT process
FROM gene
WHERE id='Q13478'
MySQL command line interface: interactive

```
smckay@brie3 DBI_lecture]$ mysql -usmckay -pcourse genes
mysql> desc favorite_color;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
|------------+--------------+------+-----+---------+-------+
| id         | int(11)      | YES  |     | NULL    |       |
| first_name | varchar(255) | YES  |     | NULL    |       |
| last_name  | varchar(255) | YES  |     | NULL    |       |
| color      | varchar(255) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> select * from favorite_color;
+------+------------+-----------+-------+
<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Tom</td>
<td>Jones</td>
<td>blue</td>
</tr>
<tr>
<td>101</td>
<td>John</td>
<td>Smith</td>
<td>red</td>
</tr>
</tbody>
</table>
+------+------------+-----------+-------+
2 rows in set (0.00 sec)
```

MySQL command line interface: passing SQL via STDIN

```
DROP TABLE favorite_color;
CREATE TABLE favorite_color (id INTEGER, first_name VARCHAR(255), last_name VARCHAR(255), color VARCHAR(255));
INSERT INTO favorite_color VALUES (100, 'Tom', 'Jones', 'blue');
INSERT INTO favorite_color VALUES (101, 'John', 'Smith', 'red');
SELECT * FROM favorite_color;
```

```
smckay@brie3 DBI_lecture]$ mysql -usmckay -pcourse genes < input.txt
id  first_name last_name  color
100  Tom    Jones   blue
101  John   Smith   red
```
Normalization

- process of restructuring the logical data model of a database to eliminate redundancy
- also to help organize data efficiently, and reduce the potential for anomalies during data operations

Consider the data:

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
</tbody>
</table>
| P05542 | CD4_SHEEP | Ovis aries    | “T cell differentiation”,
                                          “positive regulation of interleukin-2 biosynthesis” |

A gene can belong to multiple processes. How do we model this?
What is wrong with this approach?

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>“T cell differentiation”, “positive regulation of interleukin-2 biosynthesis”</td>
</tr>
</tbody>
</table>

violates *atomicity*

---

Next try: duplicating rows

<table>
<thead>
<tr>
<th>id</th>
<th>symbol</th>
<th>species</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>“T cell differentiation”, “positive regulation of interleukin-2 biosynthesis”</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>“positive regulation of interleukin-2 biosynthesis”</td>
</tr>
</tbody>
</table>

problem: redundancy
Normalization

- Tables with redundant data are said to be not in *normal form*

- We *normalize* the schema by representing different kinds of data in different tables

### Normalized schema

**table: gene**

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
</tr>
</tbody>
</table>

**table: gene_process**

<table>
<thead>
<tr>
<th>gene_id</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P05542</td>
<td>positive regulation of interleukin-2 biosynthesis</td>
</tr>
</tbody>
</table>
Primary keys and foreign keys

• A primary key for a table is one or more columns which are guaranteed to be unique for every row in that table.

• A foreign key for a table is one or more columns that refer to a primary key in some other table.
Choosing primary keys

- Must be unique
  - gene symbol may be a bad choice
- Primary key should be immutable
  - should not change during lifetime of db
- ‘Natural’ vs ‘surrogate’
  - natural keys come from existing columns
    - Potentially useful for relating to external databases
  - surrogate keys are artificial and have no meaning

- Are database accessions ‘natural’?

Now that we have a normalized database, how do we query across multiple tables?

- Data can be retrieved from >1 table using the JOIN operator

- The JOIN operator is actually the composition of two operators
  - product
  - restrict
Joining two tables

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gene_id</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P05542</td>
<td>positive regulation of interleukin-2 biosynthesis</td>
</tr>
</tbody>
</table>

Join syntax

```
SELECT *
FROM
gene, gene_process
WHERE
gene.gene_id = gene_process.gene_id
```
### gene

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
</tr>
</tbody>
</table>

### gene_process

<table>
<thead>
<tr>
<th>gene_id</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P05542</td>
<td>positive regulation of interleukin-2 biosynthesis</td>
</tr>
</tbody>
</table>

SELECT * FROM gene, gene_process

### Cartesian product of gene, gene_process

<table>
<thead>
<tr>
<th>g.gene_id</th>
<th>g.symbol</th>
<th>g.species</th>
<th>gp.gene_id</th>
<th>gp.process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
</tbody>
</table>

SELECT * FROM gene AS g, gene_process AS gp WHERE g.gene_id = g.gene_process.gene_id
Further normalizations

table: gene

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species</th>
<th>species_common_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>Homo sapiens</td>
<td>Human</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>Canis familiaris</td>
<td>Dog</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>Ovis aries</td>
<td>Sheep</td>
</tr>
</tbody>
</table>

- not fully normalized
- non-primary key columns are dependent on each other

Further normalizations

table: gene

<table>
<thead>
<tr>
<th>gene_id</th>
<th>symbol</th>
<th>species_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>IL18R_Human</td>
<td>9606</td>
</tr>
<tr>
<td>P33704</td>
<td>CD4_CANFA</td>
<td>9615</td>
</tr>
<tr>
<td>P05542</td>
<td>CD4_SHEEP</td>
<td>9940</td>
</tr>
</tbody>
</table>

table: species

<table>
<thead>
<tr>
<th>species_id</th>
<th>common_name</th>
<th>scientific_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9606</td>
<td>human</td>
<td>Homo sapiens</td>
</tr>
<tr>
<td>9615</td>
<td>dog</td>
<td>Canis familiaris</td>
</tr>
<tr>
<td>9940</td>
<td>sheep</td>
<td>Ovis aries</td>
</tr>
</tbody>
</table>
More normalizations

<table>
<thead>
<tr>
<th>gene_id</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>Immune response</td>
</tr>
<tr>
<td>P33704</td>
<td>Immune response</td>
</tr>
<tr>
<td>P05542</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>P05542</td>
<td>positive regulation of interleukin-2 biosynthesis</td>
</tr>
</tbody>
</table>

gene_process

needs ‘keyword’ table
- controlled vocabularies
- ontologies

gene_process

<table>
<thead>
<tr>
<th>gene_id</th>
<th>term_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q13478</td>
<td>6955</td>
</tr>
<tr>
<td>P33704</td>
<td>6955</td>
</tr>
<tr>
<td>P05542</td>
<td>30217</td>
</tr>
<tr>
<td>P05542</td>
<td>45086</td>
</tr>
</tbody>
</table>

term

<table>
<thead>
<tr>
<th>term_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6955</td>
<td>Immune response</td>
</tr>
<tr>
<td>30217</td>
<td>T cell differentiation</td>
</tr>
<tr>
<td>45086</td>
<td>positive regulation of interleukin-2 biosynthesis</td>
</tr>
</tbody>
</table>
Denormalization

• The process of attempting to optimize the performance of a database by adding redundant data

• Usually proceeds from a normalized database

• Sometimes required to improve performance for large batch/data-mining queries (eg Ensembl -> BioMart)

Should a database schema always be normalized?

• It depends…
  – updates vs queries
  – type of queries
  – philosophical disposition

• Know when to stop normalizing
  – normalized = more tables, more joins

• Data ‘warehouses’
  – e.g. BioMart
Examples

- Bulk data downloads
  - performance is a factor
  - Query optimized, denormalized database

- Large genomic data repository
  - Data integrity, storage efficiency
  - Normalized database

- OLAP (On Line Analytical Processing)
  - Data summaries and calculated values that are not in the parent database
  - Denormalized database with pre-computed fields

Summary

Relational databases are backed by theory
- powerful
- fast (usually)
- but some things are hard or difficult to express

Further reading on Normalization Theory
Further Reading on normalization theory


Dealing with very dense data

• Microarrays
• Next-gen Sequencing
• **Wiggle**
  - Large amounts of scored data with genomic coordinates
  - Too many table rows for a relational database
  - Solution is a hybrid databaseserialized data approach

WIG is a format specification introduced by the UCSC Genome Browser and also adopted by GBrowse

1) The WIG file is converted to a query-optimized binary file
2) A pointer to the binary file is stored in the database
3) An external adapter queries the binary file

http://genome.ucsc.edu/goldenPath/help/wiggle.html
http://gmod.org/wiki/GBrowse/Uploading_Wiggle_Tracks

• **SAM/BAM (Sequence Alignment/Map)**
  - NGS data generates huge numbers of aligned reads
  - The SAM specification allows efficient storage of read alignments against reference sequences
  - BAM is a highly efficient, compressed binary version of SAM
  - The SAMTools package provides utilities for handling the alignment data.
  - Third party implementers are starting to support SAM/BAM, for example Bio::DB::SAM/GBrowse

http://samtools.sourceforge.net/
• Choose a design that fits your data and working environment

• There are some tasks for which relational databases are not appropriate

• There are plenty of database schemas and tools out there
  – know which one to use and when
  – extend vs write your own
Problem Set

A quick primer on SQLite

- SQLite is a simple, stand alone, file based RDBMS

- The sqlite3 client is installed by default on many unix-like operating systems, including Mac OS X

- sqlite3 accepts two kinds of commands: meta-commands (preceded by a dot) and SQL statements.

Some useful meta-commands:

.schema tablename
.quit
.help (shows all of the available meta-commands)

Creating and loading a database:

$ sqlite3 my_database_name < my_SQL_file.sql