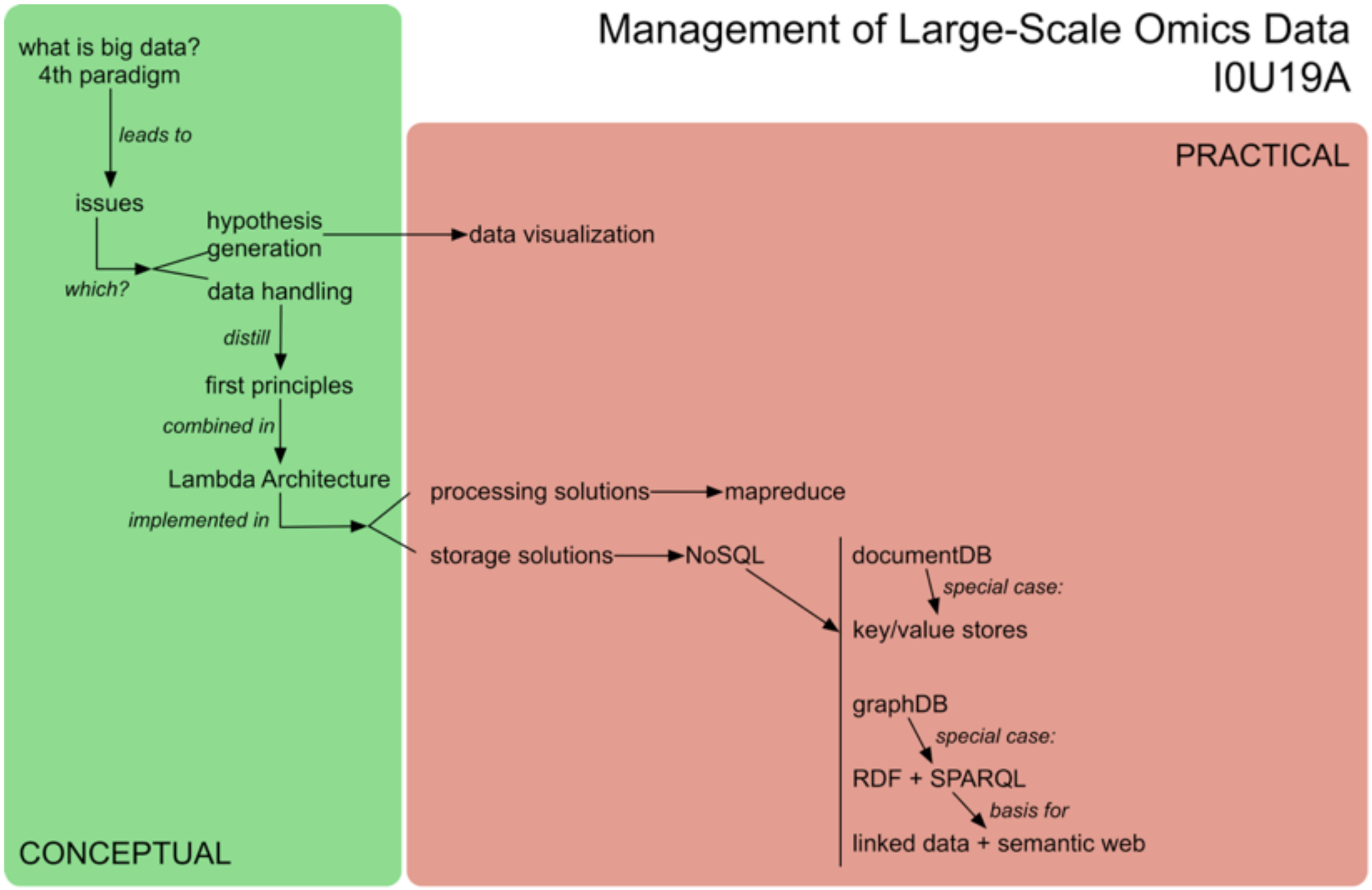
I0U19A - Management of large-scale omics data

Prof Jan Aerts
Faculty of Engineering - ESAT/SCD
Kasteelpark Arenberg 10, 3001 Leuven

jan.aerts@esat.kuleuven.be
http://vda-lab.be/teaching/i0u19a/

Overview of this course



Website

http://vda-lab.be/teaching/i0u19a/index.html

:: Visual Data Analysis Lab ::

People Portfolio Blog posts Jobs Publications Lists Contact

IOU19A - Management of Large-Scale Omics Data

General setup of the course

For each of the topics, we typically have a lecture and an exercise session. Before the exercise session, we also expect you do do an assignment so that you're well-prepared for the exercises.

We will constantly refer back to 3 different datasets within this course, to show which of the technologies discussed fits a certain use of a dataset best. For a description of these datasets, see here.

The schedule below lists 3 types:

- Lectures - regular lectures
- Exercises - exercise sessions on the computer
- Assignments - assignments to be uploaded on Toledo *before* the date indicated. These are most often to prepare for the exercise session.

Schedule

see http://vda-lab.be/teaching/i0u19a/index.html

Exercises & Assignments

- Three datasets
  - genotypes
  - beers in Belgium
  - approved drugs
- Modeled and stored using different database technologies
- => which technology (or combination of technologies) fits a particular dataset (and its intended use) best?
- Preparation of exercise session: **assignment** including *e.g.* modelling of data => answers will be used in exercise session

Evaluation

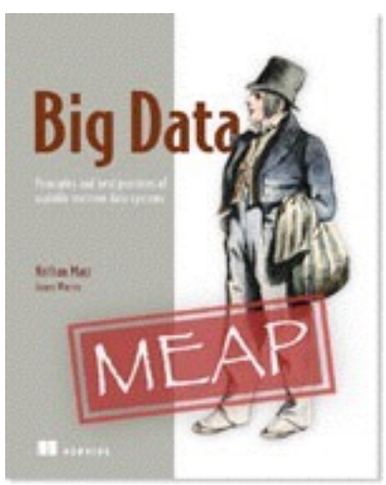
Combination of:

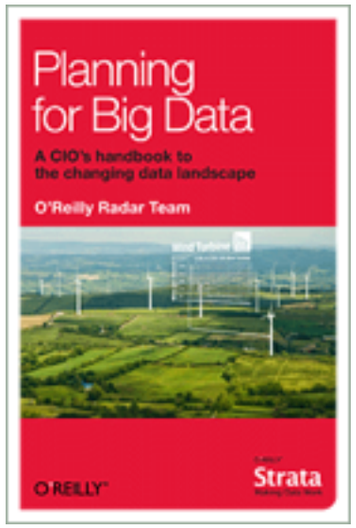
- permanent evaluation (including preparation of exercise sessions): 10%
- take-home data visualization assignment: 20%
- open-book written exam: 70%

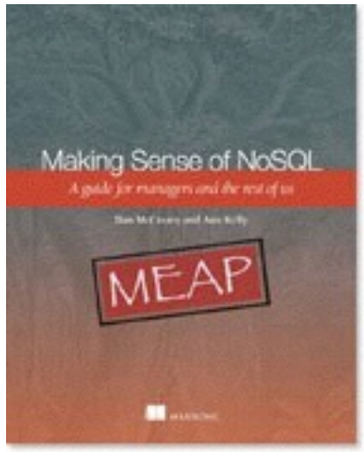
At least 8/20 for each.

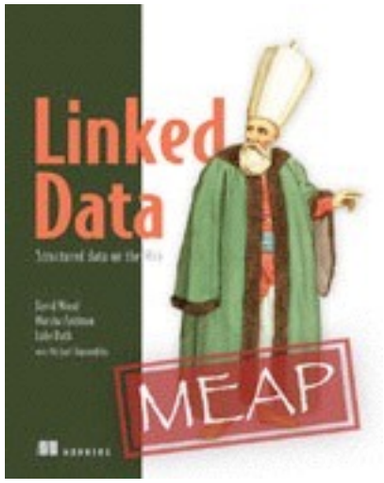
Books

- Marz N & Warren J (2013). *Big Data*. Manning Publications.
- McCreary D & Kelly A (2013). *Making Sense of NoSQL*. Manning Publications.
- Wood D, Zaidman M & Ruth L (2013). *Linked Data*. Manning Publications.





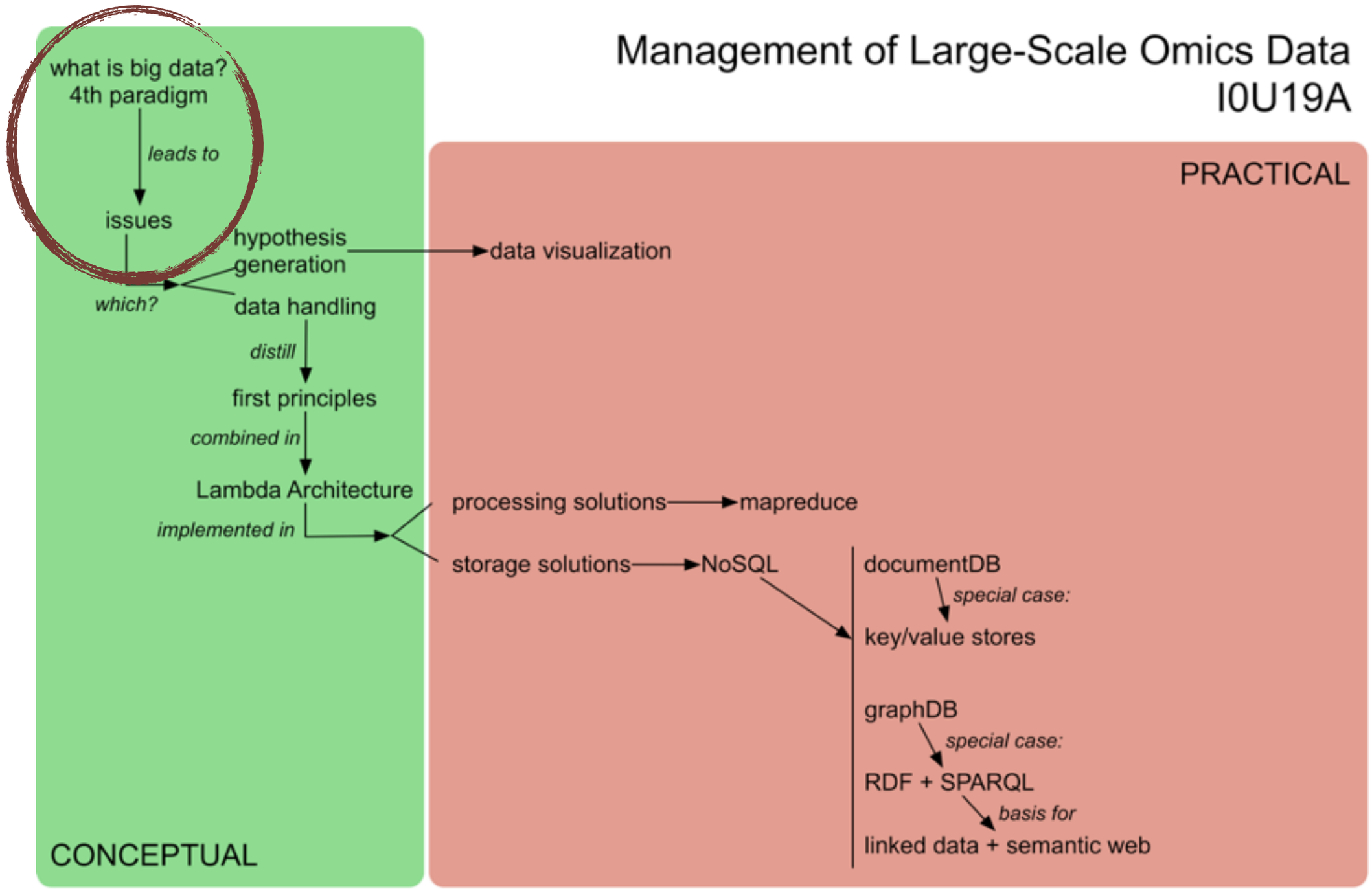




Your background

- scripting?
- SQL?

Today - What is big data?



What is big data?

How would *you* describe “big data”? Can you give examples?

Some examples

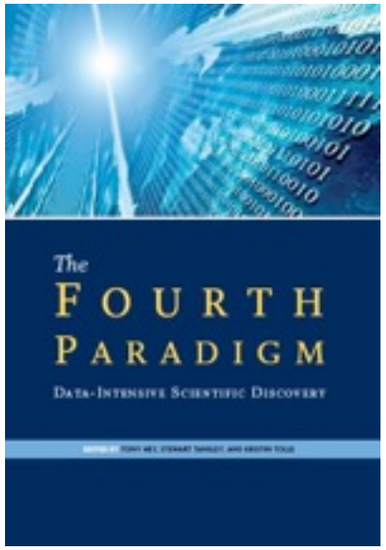
- Netflix
  - analysis of traffic patterns across device types to improve reliability of video streaming
  - recommendation engine based on viewing habits
- Politics: project “Narwhal”
  - Obama campaign operations: don’t knock on door of people who have already volunteered, don’t send email asking for money to people who already contributed
- WeatherSignal
  - repurposes sensors in Android smartphones to map atmospheric readings (barometer, hygrometer, ambient thermometer, light meter)
- Infectious diseases: Spatio-temporal Epidemiological Modeler STEM
  - IBM uses local climate and temperature to find correlations with how malaria spreads => used to predict location of future outbreaks
- Retail (Target)
  - predict future purchasing habits (*e.g.* pregnancy) => targeted ads

Why are these examples different?

- Data collection is easy
- Data is often unstructured
- Data can be used for many things
- Example datasets available at http://www.datasciencecentral.com/profiles/blogs/big-data-sets-available-for-free

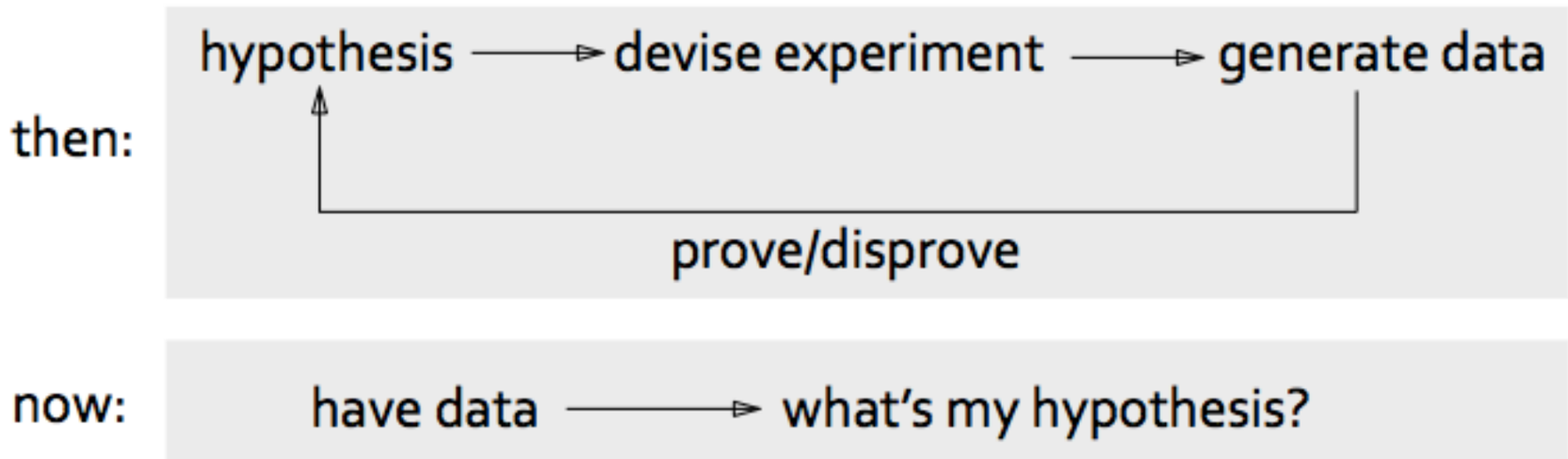
Big data, the bigger picture

"Fourth Paradigm" of scientific research (Jim Grey, Microsoft)



1st	1,000s years ago	observe, then derive
2nd	100s years ago	derive, then observe
3rd	last few decades	simulate
4rd	today	measure

hypothesis-driven research (hypothesis -> experiment -> data)
=> **data-driven research** (I have data -> what questions can I ask?)



- *data analysis*: moving from **hypothesis-first** to **data-first**

- *challenge*: moving from **finding the right answer to a question** to **finding the right question given the data**

What is big data?

- big data = data that exceeds processing capacity of conventional database systems (too big, moves too fast, doesn't fit in database structure)
- Being able to process every item of data in reasonable time removes the troublesome need for sampling
- Necessary counterpart: *agility* - successful exploitation of big data requires experimentation and exploration
- Because it's big: *bring computation to the data instead of the data to the computation*
- *Different way of thinking*

The Three V's

- **Volume** - most immediate challenge to conventional IT structures; principle of big data: *if you can, keep everything*
  - need *scalable storage* + *distributed querying*
  - structured vs unstructured data -> Hadoop: MapReduce + HDFS
    - MapReduce: *map* = distributing a dataset among multiple servers and operating on the data; *reduce* = recombining the partial results
    - HDFS = Hadoop Distributed File System
    - Hadoop: for batch jobs (not interactive)

- **Variety** - data is messy
  - 80% of effort in dealing with data = cleaning up
  - process of moving from source data to processed application data involves loss of information
  - relational databases: not always best destination for the data, even when tidied up (network data -> graph database; XML data -> dedicated XML store; ...)
  - disadvantage of relational database: fixed schema <=> results of computations will evolve with detection and extraction of more signals => semi-structured NoSQL databases provide this flexibility: provide enough structure to organize data but do not require the exact schema of the data before storing it

- **Velocity** - increasing rate at which data flows into an organization, but also of system's output

  - *input* (velocity of incoming data), and
    throughput (speed of taking data from input through to decision)

  - often not possible to simple wait for a report to run or Hadoop job to complete

  - *streaming*: important to consider, because (1) if input data too fast to store in its entirety (*e.g.* Large Hedron Collider @ CERN); (2) application might mandate immediate response to the data

- => need for speed (--> has driven development of key-value stores and columnar databases)

RDBMS refresher

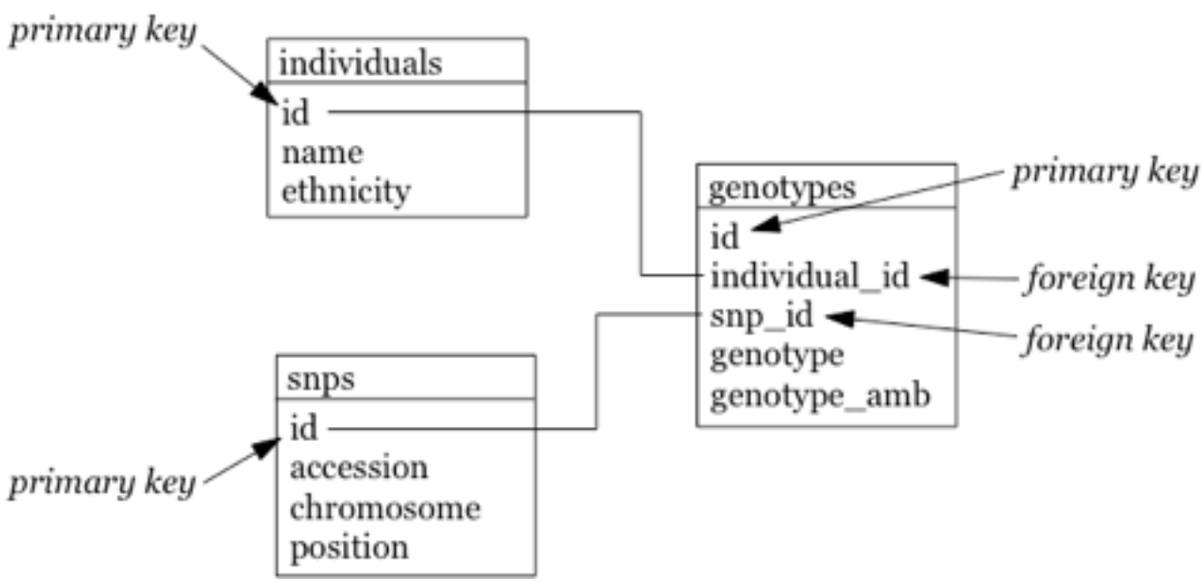
Switch to http://vda-lab.be/2015/02/introduction-to-relational-databases/index.html

Scripts are in http://vda-lab.be/teaching/i0u19a/students-only/

Why do relational databases break down with big data?

individual	ethnicity	rs12345	rs12345 _amb	chr_12345	pos_12345	rs98765	rs98765 _amb	chr_98765	pos_98765	rs13579	rs13579 _amb	chr_13579	pos_13579
individual_A	caucasian	A/A	A	1	12345	A/G	R	1	98765	G/T	K	5	13579
individual_B	caucasian	A/C	M	1	12345	G/G	G	1	98765	G/G	G	5	13579

“Normal forms”



individuals

id	name	ethnicity
1	individual_A	caucasian
2	individual_B	caucasian

genotypes

id	individual_id	snp_id	genotype	genotype_amb
1	1	1	A/A	A
2	1	2	A/G	R
3	1	3	G/T	K
4	2	1	A/C	M
5	2	2	G/G	G
6	2	3	G/G	G

snps

id	accession	chromosome	position
1	rs12345	1	12345
2	rs98765	1	98765
3	rs13579	5	13579

1. Querying scalability

Database schema normalization: every piece of information is stored only once

=> makes updating data easier and safer

=> requires less space (!)

my_data

individual	ethnicity	rs12345	rs12345_amb	chr_12345	pos_12345	rs9876	rs9876_amb	chr_9876	pos_9876	rs2468	rs2468_amb	chr_2468	pos_2468
ind_A	cauc	A/A	A	1	12345	A/G	R	1	9876	G/T	K	5	2468
ind_B	cauc	A/C	M	1	12345	G/G	G	1	9876	G/G	G	5	2468

SNP = single nucleotide polymorphism

normalize

individuals

id	name	ethnicity
1	ind_A	cauc
2	ind_B	cauc

genotypes

id	individual_id	snp_id	genotype_amb
1	1	1	A
2	1	2	R
3	1	3	K
4	2	1	M
5	2	2	G
6	2	3	G

snps

id	accession	chromosome	position
1	rs12345	1	12345
2	rs9876	1	9876
3	rs2468	5	2468

- advantage of normalized database: you can ask any question

- disadvantage of normalized database: to get an answer you will have to *join* tables => is expensive (*i.e.* becomes very (!) slow) if you have to combine many large tables (millions of rows)

“Return all names of individuals that have heterozygous SNPs on chromosome 1”

```
SELECT DISTINCT i.name
FROM individuals i, snps s, genotypes g
WHERE i.id = g.individual_id                <== join
AND s.id = g.snp_id                         <== join
AND s.chromosome = 1
AND g.genotype_amb NOT IN (“A”,”C”,”G”,”T”);
```

- Solution: **de-normalize**

Getting exon positions for a gene in **Ensembl** (= normalized database; 74 tables):

/usr/local/mysql/bin/mysql -u anonymous -h ensembldb.ensembl.org homo_sapiens_core_78_38

gene

gene_id	seq_region_id	seq_region_start	seq_region_end	description	...
1	226034	4274	19669	FAM39B protein	...
2	226034	24417	25944	F379 retina specific protein	...
...	...	...	...	...	...

exon_transcript

exon_id	transcript_id	rank
1	1	1
2	1	2
...	...	...

transcript

transcript_id	gene_id	seq_region_id	seq_region_start	seq_region_end	...
1	1	226034	4274	19669	...
2	2	226034	24417	25944	...
...	...	...	...	...	...

exon

exon_id	seq_region_id	seq_region_start	seq_region_end	...
1	226034	19397	19669	...
2	226034	14600	14754	...
...	...	...	...	...

```
SELECT g.description, e.seq_region_start
FROM gene g, transcript t, exon_transcript et, exon e
WHERE g.gene_id = t.gene_id
AND t.transcript_id = et.transcript_id
AND et.exon_id = e.exon_id
AND g.description LIKE “FAM170B %”;
```

description	seq_region_start
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49121839
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49135656
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49141253
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49142930
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49145782
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49150805
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49121844
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49141253
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49142930
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49145782
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49148607
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49150805
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49122086
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49135877
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49141253
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49137495
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49141253
FAM170B antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:45006]	49150805

18 rows in set (0.08 sec)

- Solution: **de-normalize**

Getting exon positions for a gene in **UCSC** (= denormalized database; 10,014 tables!!):

/usr/local/mysql/bin/mysql -u genome -h genome-mysql.cse.ucsc.edu -A hg19

geneid

bin	name	chrom	txStart	txEnd	cdsStart	cdsEnd	exonCount	exonStarts	exonEnds	...
585	chr1_1.1	chr1	14695	35736	14695	35736	10	14695,14969,15795,16853,17232,17605,17914,18267,24737,35720,	14829,15038,15891,17055,17257,17742,18061,18379,24891,35736,	...
586	chr1_2.1	chr1	228291	228654	228291	228654	1	228291,	228654,	...
...	...	...	...	...	...	...	...	...	...	...

```
SELECT exonStarts
FROM geneid
WHERE name = “chr1_1.1”;
+-----------------------------------------------------------------------------+
| exonStarts                                                                  |
+-----------------------------------------------------------------------------+
| 14695,14969,15795,16853,17232,17605,17914,18267,24737,35720, |
+-----------------------------------------------------------------------------+
```

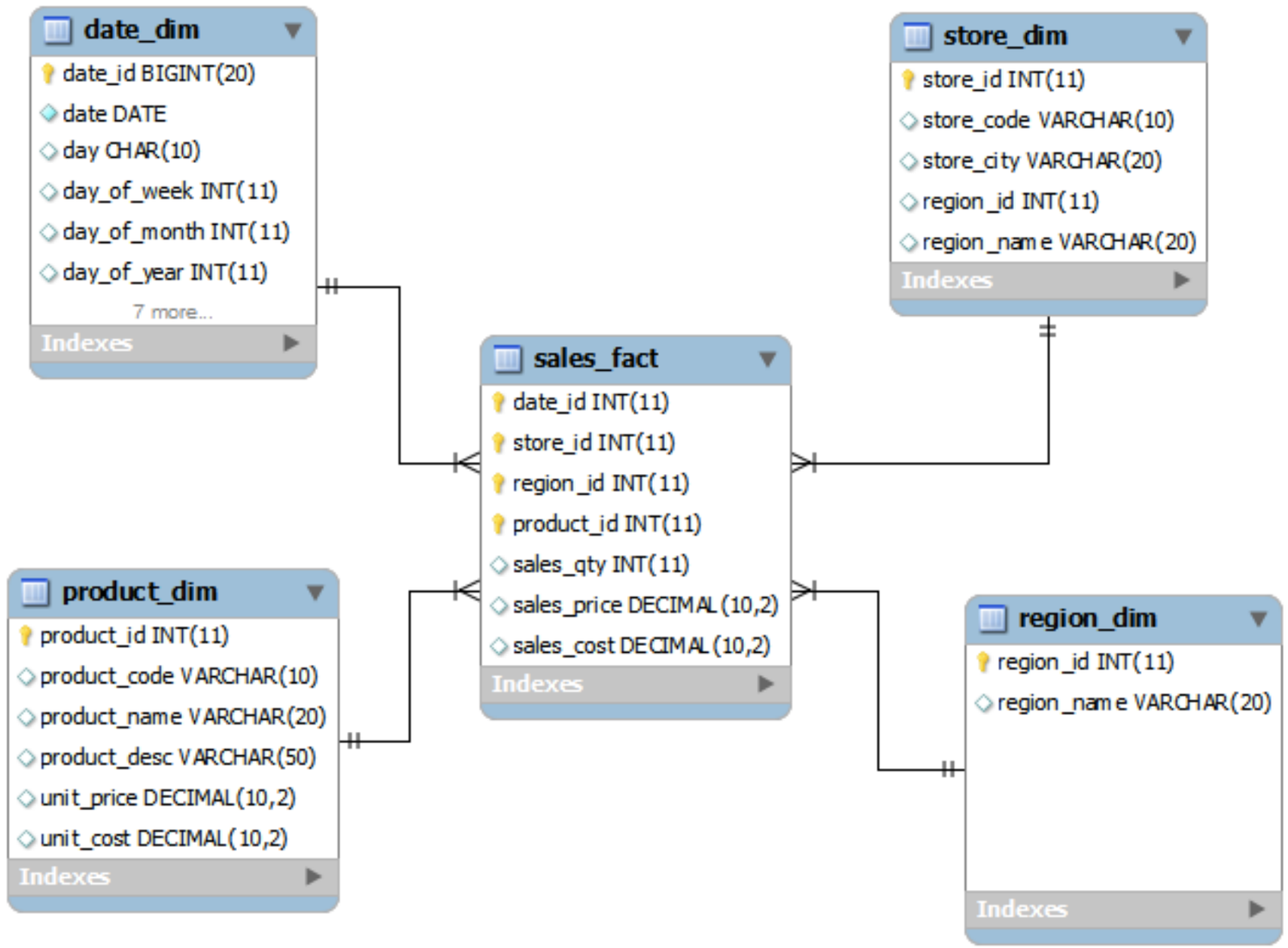
Star schema

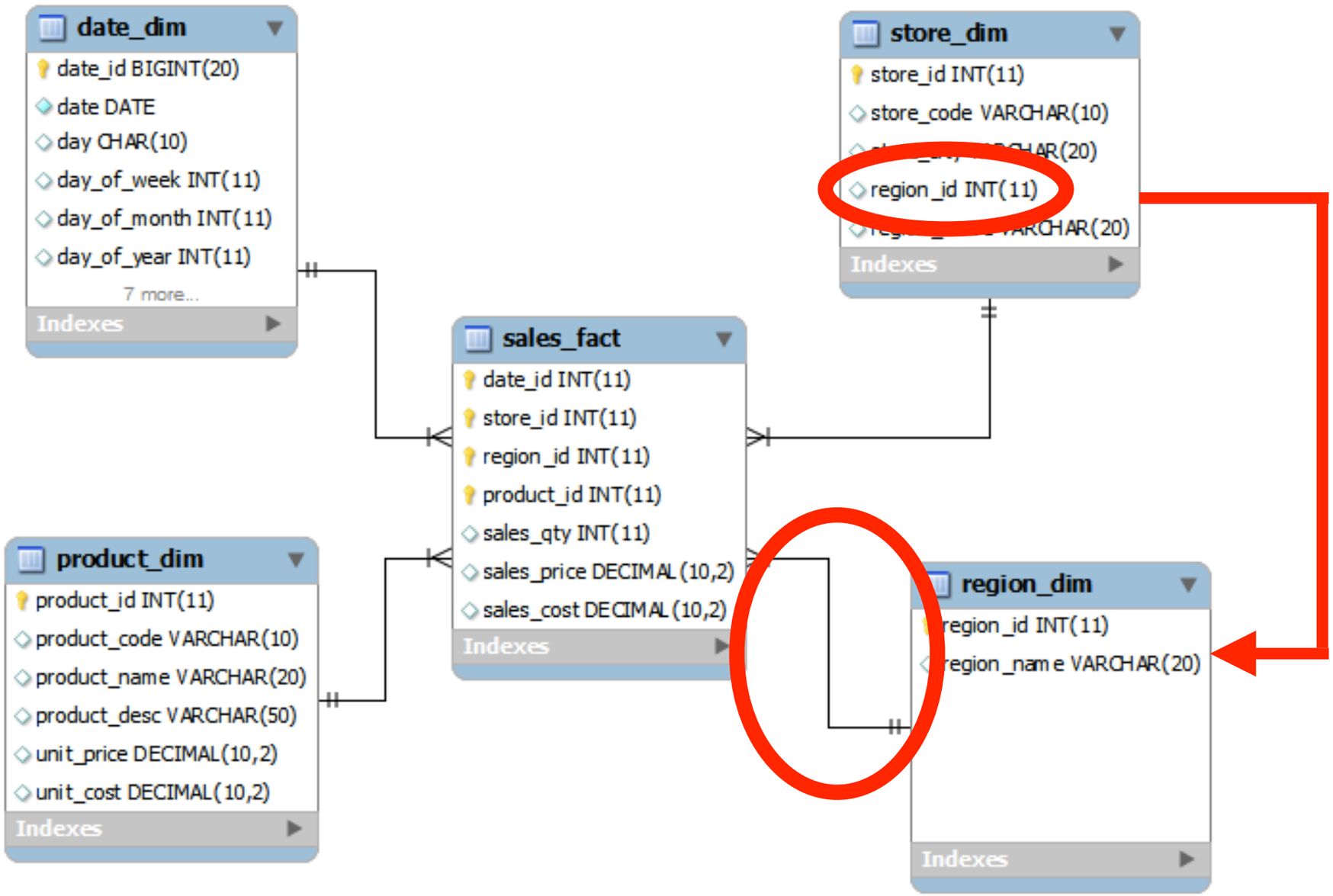
- Enables fast querying of data by minimizing joins (necessary in normalized schema)

- 2 attributes: (1) always 2 levels deep; (2) contains only one large table that is the focus of the model (**fact table**) plus >1 **dimension tables**

- database using star schema = reporting database (OLAP; != the authoritative source of the data)
  => temporarily forget the rules of normalization

- signals that you deviate from true star schema: (1) desire to retain the relationships between dimensions (= “snowflaking”); (2) existence of more than one fact table


date_dim
date_id BIGINT(20)
date DATE
day CHAR(10)
day_of_week INT(11)
day_of_month INT(11)
day_of_year INT(11)
7 more...
Indexes
store_dim
store_id INT(11)
store_code VARCHAR(10)
store_city VARCHAR(20)
region_id INT(11)
region_name VARCHAR(20)
Indexes
sales_fact
date_id INT(11)
store_id INT(11)
region_id INT(11)
product_id INT(11)
sales_qty INT(11)
sales_price DECIMAL(10,2)
sales_cost DECIMAL(10,2)
Indexes
product_dim
product_id INT(11)
product_code VARCHAR(10)
product_name VARCHAR(20)
product_desc VARCHAR(50)
unit_price DECIMAL(10,2)
unit_cost DECIMAL(10,2)
Indexes
region_dim
region_id INT(11)
region_name VARCHAR(20)
Indexes


date_dim
date_id BIGINT(20)
date DATE
day CHAR(10)
day_of_week INT(11)
day_of_month INT(11)
day_of_year INT(11)
7 more...
Indexes
store_dim
store_id INT(11)
store_code VARCHAR(10)
region_id INT(11)
Indexes
sales_fact
date_id INT(11)
store_id INT(11)
region_id INT(11)
product_id INT(11)
sales_qty INT(11)
sales_price DECIMAL(10,2)
sales_cost DECIMAL(10,2)
Indexes
product_dim
product_id INT(11)
product_code VARCHAR(10)
product_name VARCHAR(20)
product_desc VARCHAR(50)
unit_price DECIMAL(10,2)
unit_cost DECIMAL(10,2)
Indexes
region_dim
region_id INT(11)
region_name VARCHAR(20)
Indexes

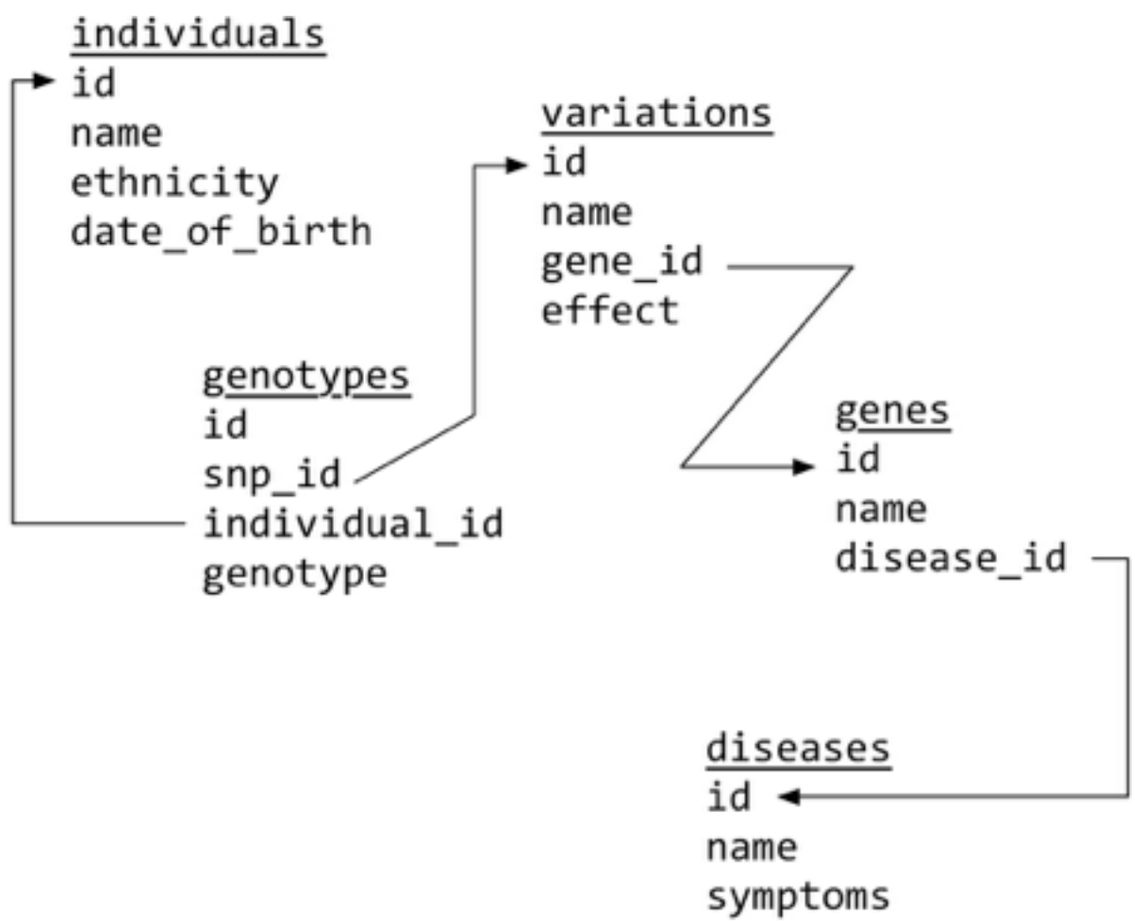
How did we go from relational to star schema?

- Everything revolves around sales => base the fact table on the sale table (one row in fact table = one row in sale table)

- Flattened the relationships all the way up the relational foreign key chain => keys in all reference tables become foreign keys in the fact table
  Create dimensions for the data pointed to by each of the foreign keys.

Exercise

- Draw a star design to optimize analysis of the data stored in a database that looks like this, when you’re interested in the genotypes:



2. Writing scalability

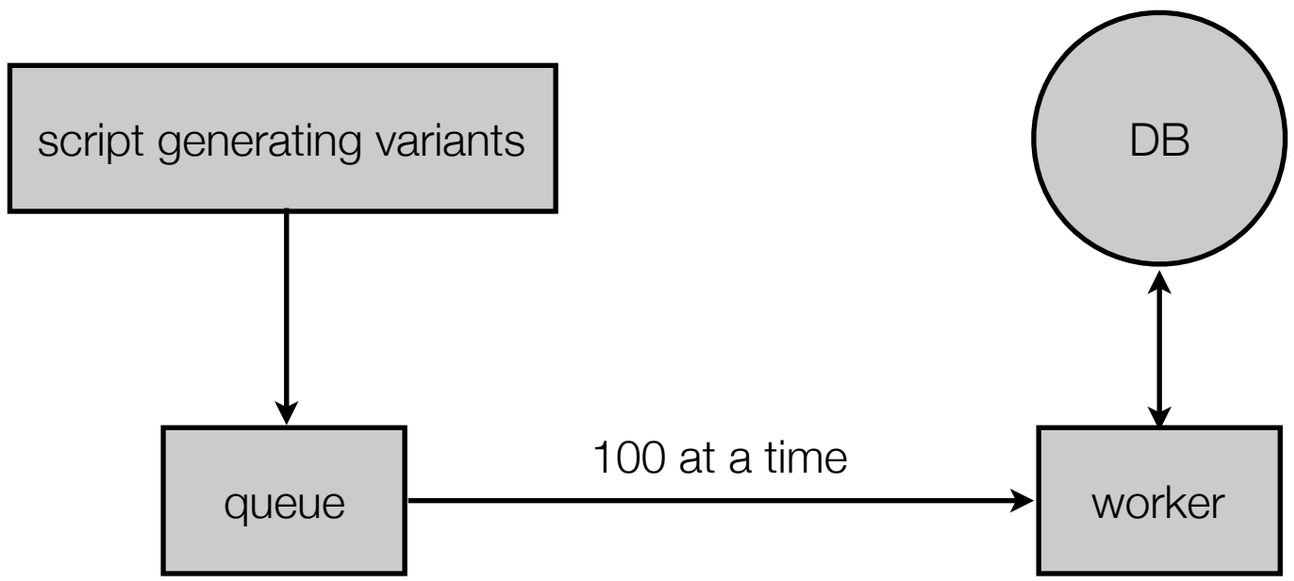
Suppose you're writing a tool to store genomic variants as they are identified in a large number of .bam files into a relational database

=> "Timeout error on inserting to database"

column name	**type**
sample ID	varchar(255)
chromosome	char(5)
position	integer
reference allele	char(1)
alternative allele	char(1)
genotype	char(2)

- Solution 1: **queuing**

wasteful to only do a single insert at a time => batch many inserts in a single request => no timeouts anymore (but queue will get longer)



=> with ever bigger loads: again bottleneck DB

- Solution 2: **sharding**

  use multiple database servers, each with a subset of the data (= "**horizontal partitioning**" or "**sharding**")

  e.g. 1 server per chromosome

  but:

  - all your application code needs to know how to find the shard for each key
  - when databases too big again: split shards (*e.g.* p- vs q-arm; per Mb; ...)
  - if so: need to update all application code that interacts with DB

- general challenges concerning storage and writing:
  - **fault-tolerance is hard**: as number of machines increases -> higher chance that one of them goes down
  - **complexity is pushed to application layer**: distributed nature of your data is not abstracted away from you (sharding)
  - **lack of human fault-tolerance**: system must be carefully thought out to limit the damage a human mistake can cause
  - **maintenance** is an enormous amount of work (re-sharding!)

First Principles for Big Data

Need to ask ourselves: “At the most fundamental level, what does a data system do?”

Data systems don’t just memorize and regurgitate information. They combine bits and pieces together to produce answers.

Not all bits of information are equal: some is derived from other => what is the most raw form of information? (NGS sequencing data: bam? fastq? images?)

You answer questions on your data by running functions that take data as input. => **query = function(all data)** => goal of data system = compute arbitrary functions on arbitrary data

Main Big Data properties

- computational systems should be **self-aware** of their distributed nature => sharding, replication, ... are handled for you

- data is **immutable** => when you make a mistake you might write bad data but at least you didn’t destroy good data => human fault-tolerant

Data immutability

- No more updates!
- How to store the number of friends in a social graph?

update sum

name	friends
Tom	2
Tim	1
Tony	4

→

update list

name	friends
Tom	[Frank, Freddy]
Tim	[Frank]
Tony	[Frank, Fletcher, Flint, Francis]

→

only add records

id	who	timestamp	action	who
1	Tom	20100402	add	Frank
2	Tony	20100404	add	Frank
3	Tom	20100407	remove	Frank
4	Tim	20100409	add	Frank
5	Tom	20100602	add	Freddy
6	Tony	20100818	add	Francis
7	Tom	20100819	add	Frank
8	Tony	20101021	add	Flint
9	Tony	20110101	add	Fletcher

“facts”

Other desired properties of a Big Data system

- **robust & fault-tolerant**: [1] system is complex (duplicated data, distributed data, concurrency, ...); [2] human fault-tolerant (deploying incorrect code that corrupts values in database)
- **low latency** reads and updates
- **scalable**: maintain performance with increasing data and/or load by just adding resources to the system
- **general**
- **extensible**: make it easy to do large-scale migrations
- allows **ad hoc queries**

- **minimal maintenance**
  - maintenance = the work required to keep a system running smoothly

    => choose components that have a small *implementation complexity*: rely on simple algorithms & components

    common trick: push complexity out of the core components and into pieces of the system whose outputs are discardable after a few hours (see later)

- **debuggable**: be able to trace for each value in the system exactly what caused it to have that value

Getting ready for exercises

- For exercise server: see information on Toledo
- Usernames & passwords: see paper slips

Exercise 1 - data modelling

- Suppose you want to model a social graph. People have names, and know other people. Every “know” is reciprocal (so if I know you then you know me). The data might look like this:

  Tim knows Terry
  Tom knows Terry
  Terry knows Gerry
  Gerry knows Rik
  Gerry knows James
  James knows John
  Fred knows James
  Frits knows Fred

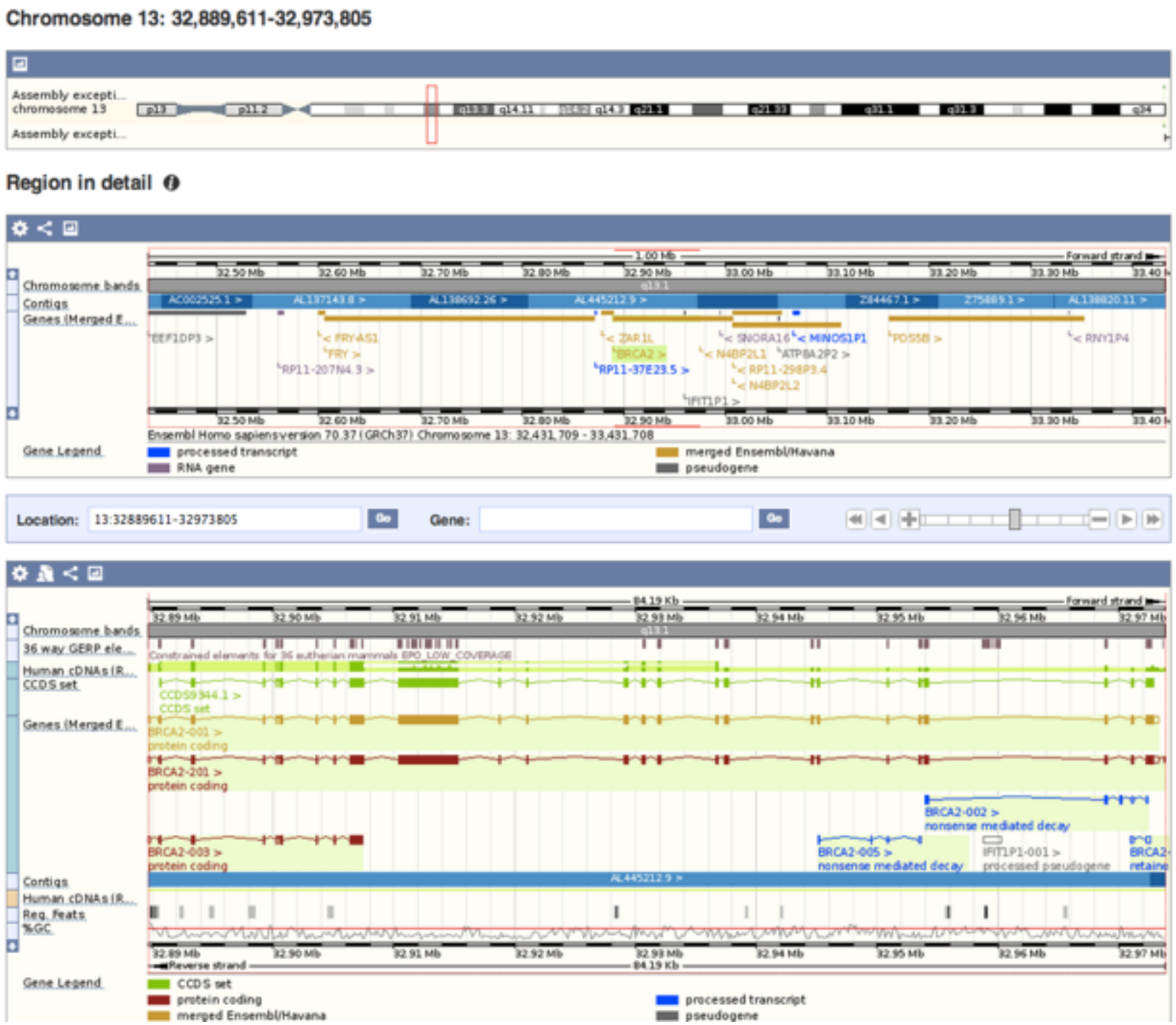
knower	knowee
Tim	Terry
Tom	Terry
Terry	Gerry
Gerry	Rik
Gerry	James
James	John
Fred	James
Frits	Fred

Given that you really want to have this in a relational database, how would you find out who are the **friends of the friends of James**? **On the teaching server: create a database, load the data, and write the SQL query**.

Exercise 2 - genome browsers

- Most used online genome browsers:
  - http://www.ensembl.org ”Ensembl”
  - http://genome.ucsc.edu “UCSC”

Ensembl


Chromosome 13: 32,889,611-32,973,805
Region in detail
Location: 13:32889611-32973805
Go
Gene:
Go
Chromosome bands
Contigs
Gene Legend
processed transcript
RNA gene
merged Ensembl/Havana
pseudogene
36 way GERP ele...
Human cDNAs (R...
CCDS set
Genes (Merged E...
Contigs
Human cDNAs (R...
Reg. feats
%GC
Gene Legend
CCDS set
protein coding
merged Ensembl/Havana
processed transcript
pseudogene

UCSC


UCSC Genome Browser on Human Feb. 2009 (GRCh37/hg19) Assembly
move <<< << < > >> >>> zoom in 1.5x 3x 10x base zoom out 1.5x 3x 10x
chr13:32,510,733-33,352,682 841,950 bp.
enter position, gene symbol or search terms
go
Scale
chr13:
200 kb
hg19
UCSC Genes (RefSeq, UniProt, CCDS, Rfam, tRNAs & Comparative Genomics)
move start
move end
Click on a feature for details. Click or drag in the base position track to zoom in. Click side bars for track options. Drag side bars or labels up or down to reorder tracks. Drag tracks left or right to new position.
track search
default tracks
default order
hide all
add custom tracks
track hubs
configure
reverse
resize
refresh
collapse all
Use drop-down controls below and press refresh to alter tracks displayed.
Tracks with lots of items will automatically be displayed in more compact modes.
expand all
Mapping and Sequencing Tracks
refresh
Base Position
Chromosome Band
STS Markers
FISH Clones
Recomb Rate
deCODE Recomb

Getting to the underlying genome browser data

- Both databases use relational database underneath the browser:
  - `mysql -h ensembldb.ensembl.org -P 5306 -u anonymous \ homo_sapiens_core_70_37`
  - `mysql --user=genome --host=genome-mysql.cse.ucsc.edu \ -A hg19`

What’s the main difference?

- log into the teaching server (IP number: see Toledo)
- log into the Ensembl database with mysql
  - `show tables;`
  - `SELECT * FROM gene LIMIT 2;`
    `SELECT * FROM exon LIMIT 2;`
    `SELECT * FROM transcript LIMIT 2;`
    `SELECT * FROM exon_transcript LIMIT 2;`
- log into the UCSC database using mysql
  - `show tables;`
  - `SELECT * FROM refGene LIMIT 2;`
- What is different in the way they store the information?