Data Handling

(Lectures on High-performance Computing for Economists IX)

Jesús Fernández-Villaverde[1] and Pablo Guerrón[2]

February 18, 2023

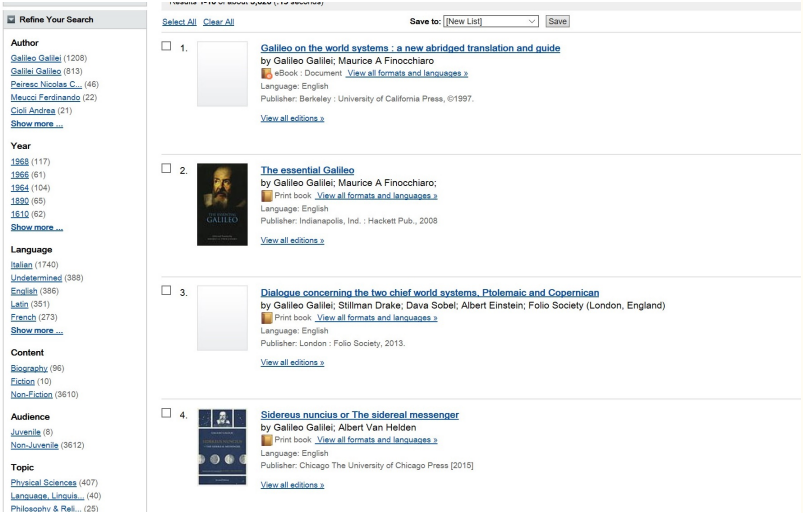
[1]University of Pennsylvania

[2]Boston College

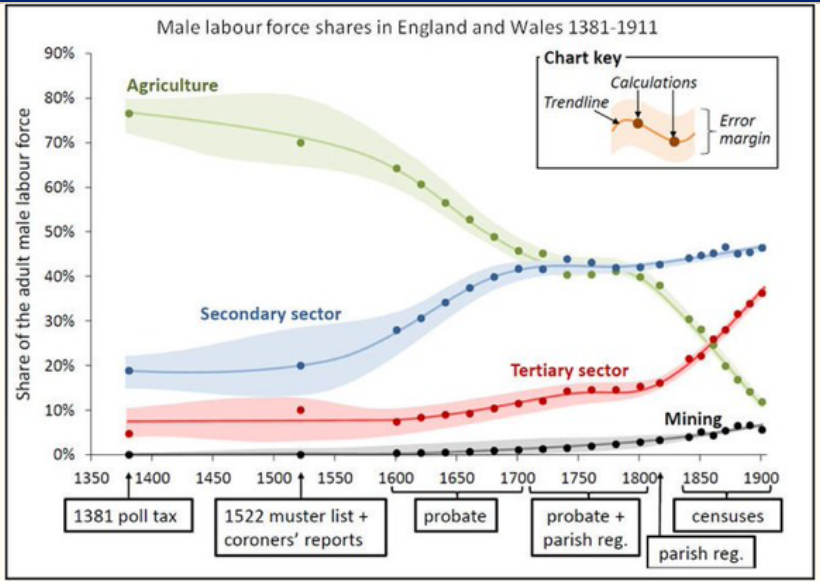
Handling data I

- In modern economics, we often deal with large and complex sets of data (big data).
- Some data are "conventional" (national accounting, micro panels, industry surveys, census data, international trade flows, ...).
- Some data come in "non-conventional" forms (plain text, library records, parish and probate records, GIS data, electricity consumption, satellite imagery, web scraping, network structure, social media, ...).
- Some data are old but now easily available. Check the amazing dataset at https://www.ucl.ac.uk/lbs/.
- This trend will increase over time as more archives get digitalized.
- These large datasets create their own challenges regarding data wrangling, storage, management, visualization, and processing.

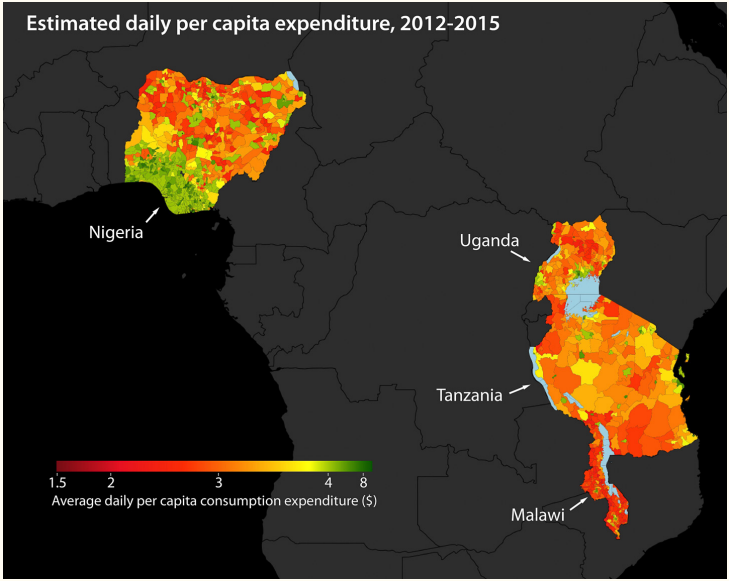
Library data



Parish and probate data



Satellite imagery



Cell phone usage

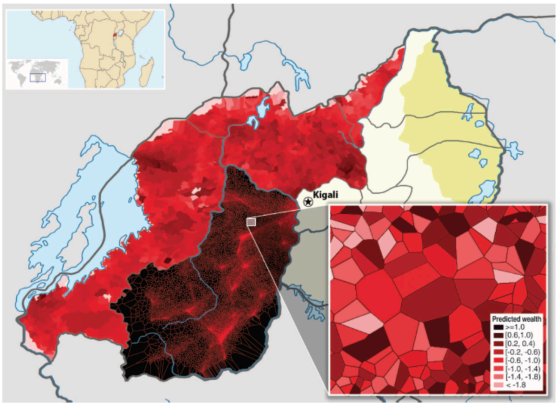


Fig. 2. Construction of high-resolution maps of poverty and wealth from call records. Information derived from the call records of 1.5 million subscribers is overlaid on a map of Rwanda. The northern and western provinces are divided into cells (the smallest administrative unit of the country), and the cell is shaded according to the average (predicted) wealth of all mobile subscribers in that cell. The southern province is overlaid with a Voronoi division that uses geographic identifiers in the call data to segment the region into several hundred thousand small partitions. (**Bottom right inset**) Enlargement of a 1-km^2 region near Kiyonza, with Voronoi cells shaded by the predicted wealth of small groups (5 to 15 subscribers) who live in each region.

Blumenstock et. al. (2015)

Handling data II

- This will become more salient over time: watch the lectures at http://www.equality-of-opportunity.org/bigdatacourse/.
- Why?
  1. Explosion of data sources.
  2. Computational power.
  3. Advances in algorithms: machine learning and modern data structures/databases (influence of Google).
- This topic will require a whole course on its own, so I will only introduce fundamental ideas.
- Also, this lecture should motivate you to understand further the data structures of your favorite programming language (e.g., in `R`, the dataframe; in `Python`, the pandas).

References

- Some basic references:

1. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*, by Hadley Wickham and Garrett Grolemund.
2. *Principles of Data Wrangling: Practical Techniques for Data Preparation*, by Tye Rattenbury *et al.*
3. *Data Wrangling with R*, by Bradley C. Boehmke.
4. *Database Systems: Design, Implementation, Management (12th Edition)*, by Carlos Coronel and Steven Morris.
5. *Designing Data-Intensive Applications*, by Martin Kleppmann.
6. *Big Data: Principles and Best practices of Scalable Realtime Data Systems*, by Nathan Marz and James Warren.

Data vs. metadata I

- A good way to start thinking about how to handle data efficiently is to distinguish between the data and its metadata.
- Data: ultimate information of interest.
- Metadata: data about the data.
- Tye Rattenbury *et al.* subdivide metadata in five aspects:
  1. Structure: format and encoding of its records and fields.
  2. Granularity: kinds of entities that each data record contains information about.
  3. Accuracy: quality of the data.
  4. Temporality: temporal structure of the representation of the data.
  5. Scope: number of distinct attributes represented and the population coverage.

Data vs. metadata II

- For simple projects, the metadata will be trivial, and you do not need to spend much time thinking about it.
- But for complex, large projects, spending some time "getting" the metadata right will be crucial:
  1. Assess how much effort you want to spend in wrangling the data (e.g., manual vs. automatization).
  2. Assess how much effort you want to spend auditing the data.
  3. Assess how much effort you want to spend in storing the data efficiently.
  4. Assess how early decisions regarding the metadata might limit your future analysis.

Alternative data file formats: plain text files

The Quartz guide to bad data

I once acquired the complete dog licensing database for Cook County, Illinois. Instead of requiring the person registering their dog to choose a breed from a list, the creators of the system had simply given them a text field to type into. As a result, this database contained at least 250 spellings of Chihuahua.

- Issues:
  1. Inconsistent spelling and/or historical changes.
  2. N/A, blank, or null values.
  3. 0 values (or -1 or dates 1900, 1904, 1969, or 1970).
  4. Text is garbled.
  5. Lines ends are garbled.
  6. Text comes from optical-character recognition (OCR).

Alternative data file formats: pdf files

- Many documents follow pdf format.
- Package in R: `Pdftools 2.0`.
- An example:

```
pdf_data(pdf, opw = "", upw = "")
```

Regular expressions I

Regular expressions II

- You need to learn a programming language that manipulates regular expressions efficiently.
- Tye Rattenbury *et al.* claim that between 50% and 80% of real-life data analysis is spent with data wrangling.
- About regular expressions in general:
  1. Tutorial: `https://www.regular-expressions.info/reference.html`.
  2. Online trial: `https://regexr.com/`.
- Modern programming languages have powerful regular expressions capabilities.
- In `Python`: `https://www.tutorialspoint.com/python/python_reg_expressions.htm`.

Regular expressions and R

- In R: https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf.
- Two key packages: dplyr and tidyr part of tidyverse:

```
install.packages("tidyverse")
```

- In particular, learn to use the piping command from dplyr to make code more readable:

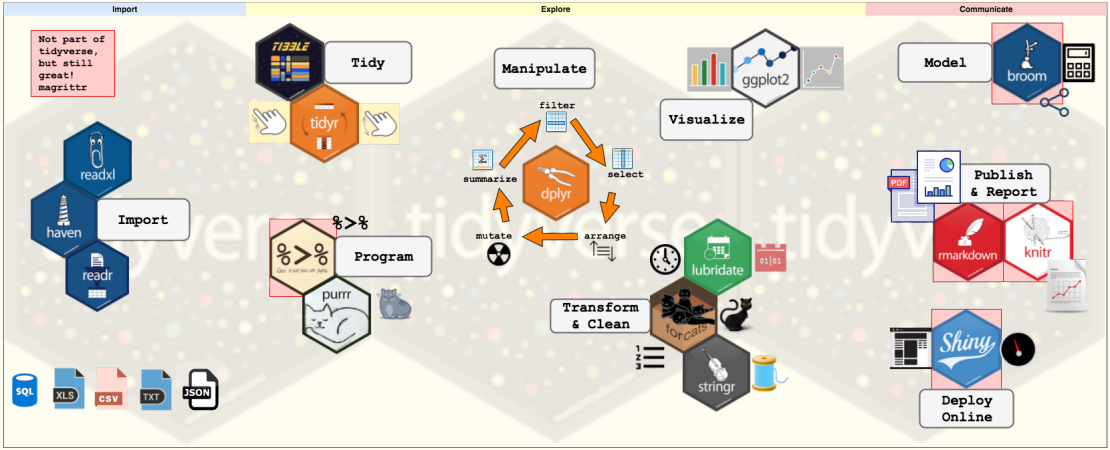
```
x %>% f(y)
f(x, y)
```

- A real example we will discuss below

```
mySelection %>%
  filter(weight < 5) %>%
  select(species_id, sex, weight)
```

- Look also at https://www.tidytextmining.com/ for text mining.

A graph we already saw



Alternative data file formats: JSON I

- JSON: JavaScript Object Notation, https://www.json.org/:
- Hierarchical data format:
  1. A collection of key-value pairs.
  2. An ordered list (array) of values. The values can be themselves either data or another nested structure.
- Very efficient for the storage, transmitting and parsing of data.
- It has gained much popularity with respect to XML.
- Important for modern databases (more on this later).
- At the core of Jupyter.
- UBSON: Universal Binary JSON.

Alternative data file formats: JSON II

Example of JSON data, myObj:

```
{
"name":"Adam Smith",
"age":30,
"universities":[ "Princeton", "Penn", "Minnesota" ]
}
```

Accessing the data:

```
x = myObj.universities[0];
```

JSON and R

In Rm we can install the rjson package.

```
install.packages("rjson")
library (rjson)
```

And use its capabilities to read a JSON object:

```
mydata <- fromJSON (myObj)
mydata_df <- data.frame (NULL)
for(i in seq_along (mydata$universities)) {
df <- data.frame (mydata$universities)
layoff_df <- rbind (layoff_df, df)
}
```

More alternative data file formats

- `HTML` and `XML`.
- Plenty of alternative proprietary data formats:
  1. Microsoft office.
  2. Stata files.
  3. pdf files.
  4. ...
- Usually a bad idea to rely on them...
- ...but sometimes they are the only alternative. Resort to tool such as `Tabula` (`https://tabula.technology`) and `WebPlotDigitizer`.

Spreadsheets

- For datasets of moderate size, spreadsheets are a conventional choice.
- However, you should be careful while using them:
  1. Do not employ their proprietary data formats (i.e., `xlsx`).
  2. Do not perform any computation in the spreadsheet. They are not reproducible, and you are bound to make mistakes (or simply forget what you did).
- Best practices:
  1. Comma-separated values (`CSV`) files are easier to share among co-authors, computers, and across time.
  2. Load the `CSV` file into `Julia` or `R` and run a script file on it. Store the script!
  3. Use `Jupyter`, `Hydrogen`, or similar if you care about showing all the steps in detail.
  4. Use `tidyverse` in `R` to interact with `Excel` and other standard spreadsheets.

Databases

- A database is a self-described, organized collection of records (tuples), each with multiple attributes.
- Components:
  1. Data: the records and attributes of the database.
  2. Metadata: the organization of the database stored in a data dictionary.
- A spreadsheet is, then, just a straightforward database.
- Similarly, a flat file (i.e., a simple `CSV` file) is a trivial database.
- A blockchain is a distributed database updated by consensus through a proof-of-work ticket.

Why databases? I

- Complex data structures require a more sophisticated database (either single or multi-user) with a database management system (DBMS) that stores, manages, and facilitates access to records.
- For instance, your data cannot fit into a simple table without excessive redundancies or without loss of efficiency in its processing.
- Examples in economics: CEX data, individual firm data, ...
- Other times, the data is too large to be stored in RAM, and you just want to select and manipulate some observations efficiently.

A bad design

Why are there blanks in rows 9 and 10?

How to produce an alphabetical listing of employees?

How to count how many employees are certified in Basic Database Manipulation?

Is Basic Database Manipulation the same as Basic DB Manipulation?

What if an employee acquires a fourth certification?

Do we add another column?

ID	ENum	Name	Title	HireDate	Skill1	Skill1Date	Skill2	Skill2Date	Skill3	Skill3Date
1	02345	Brian Oates	DBA	2/14/1995	Basic Database Management	2/14/2002	Advanced Database Management	2/14/2005	Basic Web Design	6/9/2003
2	08273	Marco Bienz	Analyst	7/26/2006	Basic Web Design	3/9/2009	Advance Process Modeling	8/19/2012		
3	06234	Jasmine Patel	Programmer	8/10/2005	Basic Web Design	8/10/2007	Advanced C# programming	8/10/2007	Basic DB manipulation	1/29/2012
4	03373	Franklin Johnson, Jr.	Purchasing Agent	3/15/2002	Advanced Spreadsheets	6/20/2011				
5	13667	Almond, Robert	Analyst	9/30/2012	Basic Process Modeling	9/30/2014	Basic Database Design	5/23/2015		
6	10282	Richardson, Amanda	Clerk	4/11/2011						
7	09382	Susan Mathis	Database Programmer	8/2/2010	Basic DB Design	8/2/2012	Basic Datsbase Manipulation	8/2/2012	Advanced DB Manipulation	5/1/2013
8	14311	Duong, Lee	Programmer	9/1/2014	Basic Web Design	9/1/2016				
9					Master Database Programming					
10					Basic Spreadsheets					
11	09002	Wade Gaither	Clerk	5/20/2010	Advanced Spreadsheets	5/16/2013	Basic Web Design	5/16/2013		
12	13363	Raymond F. Matthews	Programmer	3/12/2012	Basic C# Programming	3/12/2014				
13	09283	Chavez, Juan	Clerk	7/4/2010						
14	04893	Patricia Richards	DBA	6/11/2004	Advanced Database Management	6/11/2006	Advanced Database Manipulation	9/20/2012		
15	13932	Lee, Megan	Programmer	9/29/2013						

A good design

Database name: Ch01_Text

Table name: EMPLOYEE

Employee_ID	Employee_FName	Employee_LName	Employee_HireDate	Employee_Title
02345	Johnny	Jones	2/14/1995	DBA
03373	Franklin	Johnson	3/15/2002	Purchasing Agent
04893	Patricia	Richards	5/11/2004	DBA
06234	Jasmine	Patel	9/10/2005	Programmer
08273	Marco	Blanz	7/29/2006	Analyst
09002	Ben	Joner	5/20/2010	Clerk
09283	Juan	Chavez	7/4/2010	Clerk
09382	Jessica	Johnson	8/2/2010	Database Programmer
10282	Amanda	Richardson	4/11/2011	Clerk
13383	Raymond	Matthews	3/12/2012	Programmer
13567	Robert	Almond	9/30/2012	Analyst
13902	Megan	Lee	9/29/2013	Programmer
14311	Lee	Duong	9/1/2014	Programmer

Table name: CERTIFIED

Employee_ID	Skill_ID	Certified_Date
02345	100	2/14/2002
02345	110	8/9/2003
02345	160	2/14/2005
03373	120	6/29/2011
04893	100	6/11/2006
04893	220	9/20/2012
06234	110	8/10/2007
06234	200	8/10/2007
06234	210	1/29/2012
08273	110	3/8/2009
08273	150	8/19/2012
09002	110	5/15/2013
09002	120	5/15/2013
09382	140	8/2/2012
09382	210	8/2/2012
09382	220	5/1/2013
13383	170	3/12/2014
13567	130	9/30/2014
13567	140	5/23/2015
14311	110	9/1/2014

Table name: SKILL

Skill_ID	Skill_Name	Skill_Description
100	Basic Database Management	Create and manage database user accounts.
110	Basic Web Design	Create and maintain HTML and CSS documents.
120	Advanced Spreadsheets	Use of advanced functions, user-defined functions, and macroing.
130	Basic Process Modeling	Create core business process models using standard libraries.
140	Basic Database Design	Create simple data models.
150	Master Database Programming	Create integrated trigger and procedure packages for a distributed environment.
160	Basic Spreadsheets	Create single tab worksheets with basic formulas
170	Basic C# Programming	Create single-tier data aware modules.
180	Advanced Database Management	Manage Database Server Clusters.
190	Advance Process Modeling	Evaluate and Redesign cross-functional internal and external business processes.
200	Advanced C# Programming	Create multi-tier applications using multi-threading
210	Basic Database Manipulation	Create simple data retrieval and manipulation statements in SQL.
220	Advanced Database Manipulation	Use of advanced data manipulation methods for multi-table inserts, set operations, and correlated subqueries.

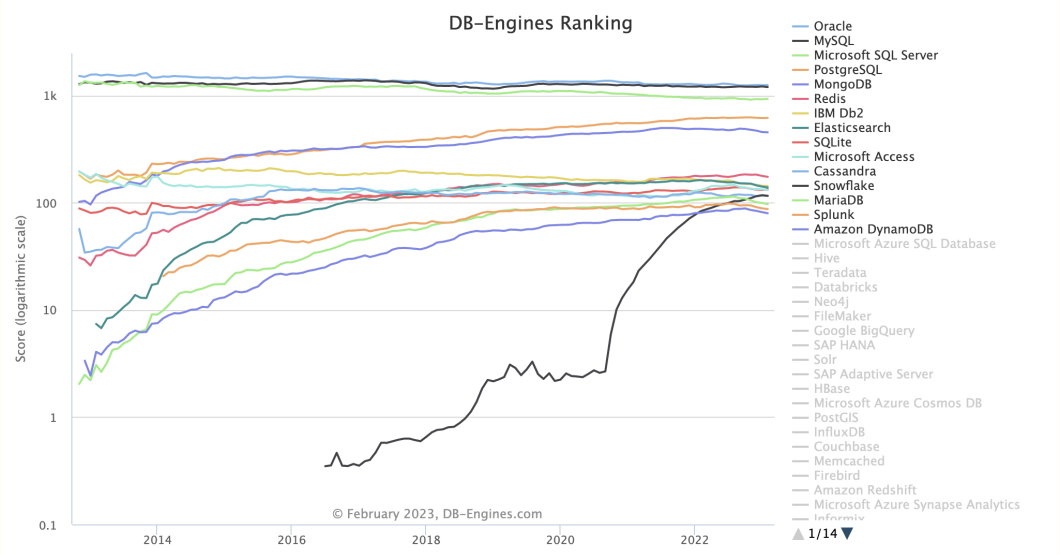
Why databases? II

- Often, you can build your own database in your code using object orientation and user-defined types.
- However, sometimes you need:
  1. Refined capabilities of selection/joins.
  2. Scalability.
  3. Ensure safe concurrent operations on data.
  4. Avoid data anomalies.
  5. Prevent data loss from hardware/software crashes.
  6. Interact with an already built database (e.g., at a statistical agency).
  7. Build your own database.
  8. Parallel computation and optimized data structures.

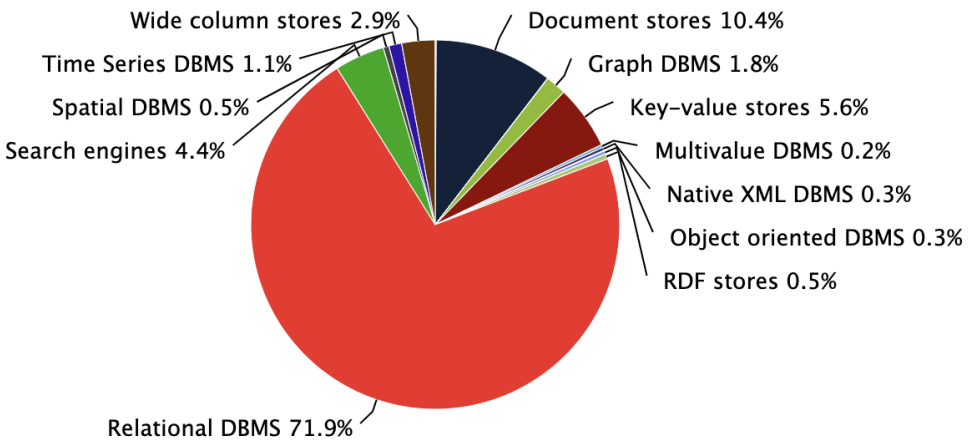
Database engines

- Plenty of industry-strength, scalable DBMS.
- At the core of each DBMS, you have a database engine that creates, reads, updates, and deletes (CRUD) data.
- You can always access the engine directly with an API (for instance, to use within your code in `C++` or `R`). This is likely the most common case for researchers.
- In addition, there is usually a GUI to interact with the DBMS (most famous: `Microsoft Access`).
- Also, general query language for APIs not tied to any database, such as `GraphQL`.
- A good source of information on the popularity of database engines: https://db-engines.com/en/.

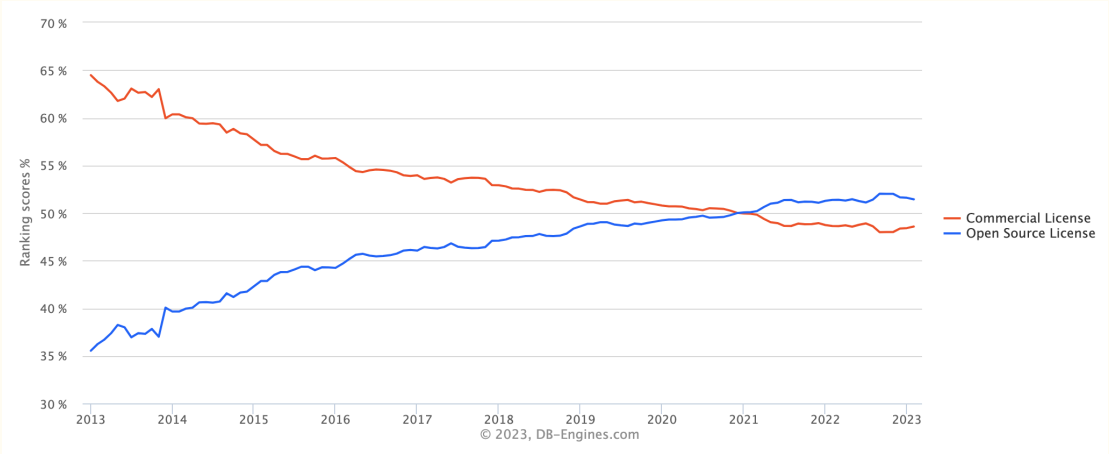
Popularity of databases



Popularity by category



Open source vs. commercial databases



Databases vs. IBM RAMAC, 1956

Database management systems I

- As mentioned before, a DBMS plays three roles:

  1. Data storage: special attention to system and disk failures and to data structures that deliver good performance.

     Interesting application of dynamic programming: Selinger et al. (1979), https://people.eecs.berkeley.edu/~brewer/cs262/3-selinger79.pdf.

  2. Data management: how data is logically organized, who has access to it (read, write), and consistency conditions.

  3. Data access: how access is accessed (queries) and what types of computations are allowed in them.

- In real-life applications, these three tasks can involve high levels of complexity.

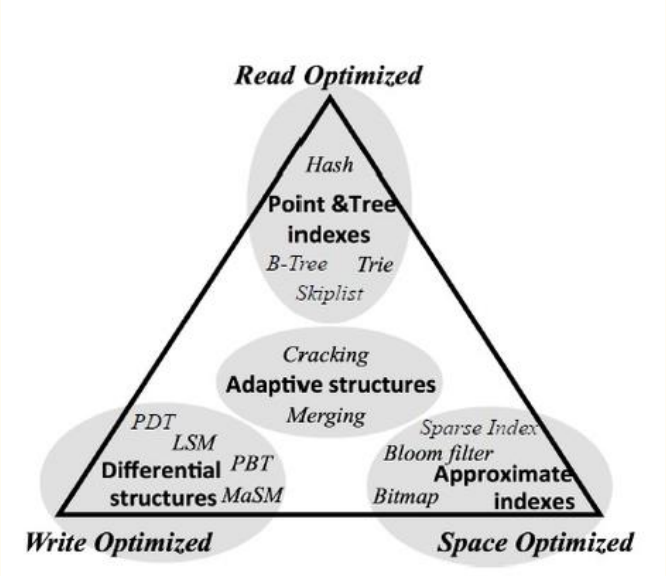
- In particular: multiple people have access to them, and they involve various units of hardware and software (think about an airline reservation system).

Database management systems II

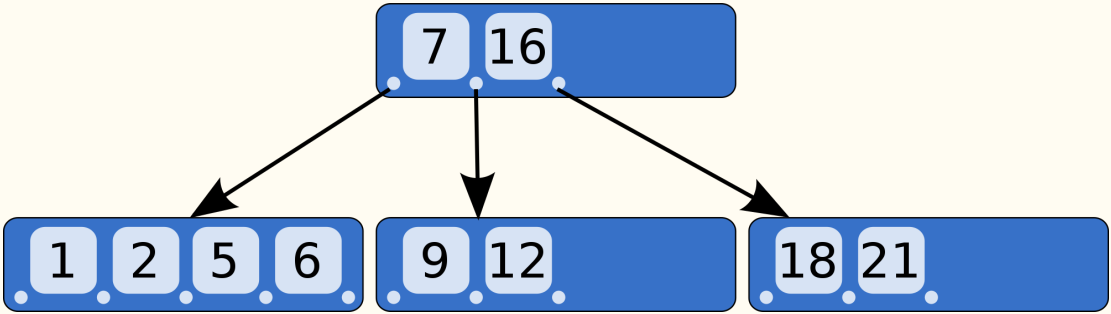
Modern DBMS hide how data is stored from end-user applications:

1. Thus, systems can evolve (i.e., hardware and software implementation of data structures and optimized storage) without affecting you.
2. Similarly, you can change the database (e.g., add a new table) without having to modify the code that queries the database and manipulates the results of the query.
3. The DBMS can handle abstract applications instead of being specifically tied to one design of a concrete application.
4. Most DBMS are declarative, not imperative (tell the software what you want, not how to get it):
   - 4.1 Easier to use for non-programmers (many users will not be)...
   - 4.2 ...but harder to optimize.

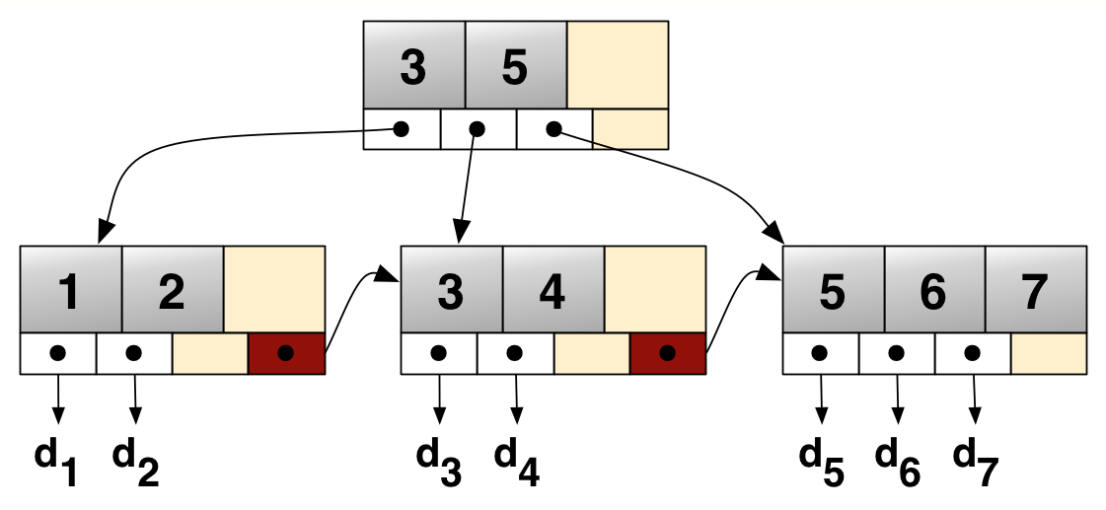
Optimized data structures



B trees



B+ trees



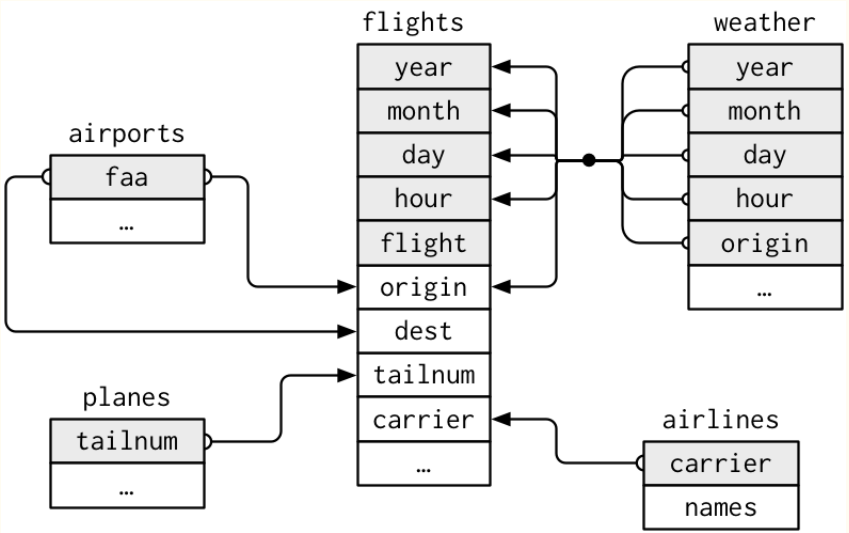
The CAP theorem

- Conjectured by Eric Brewer (2001), but proven by Seth Gilbert and Nancy Lynch (2002).
- In a distributed database, you can only choose two of:
  1. Consistency.
  2. Availability.
  3. Partition tolerance.
- If you think about it, the real trade-off is between consistency and availability since the problem comes from the existence of a partition tolerance.
- Extension: PACELC theorem (Daniel J. Abadi, 2012): even in the absence of partitions, one has to choose between latency (L) and consistency (C).

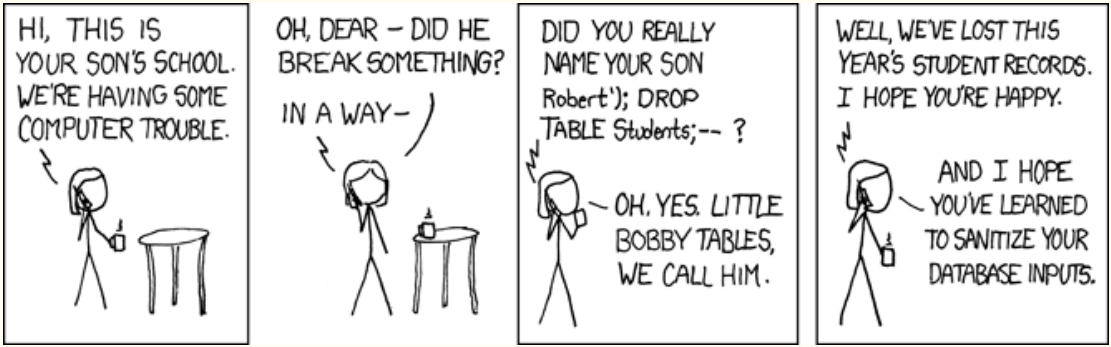
Relational database management systems

- Relational database management system (RDBMS) manage data stored in relations (i.e., a table).
- Each relation has a schema (description of attributes, their types, and constraints). An instance is data satisfying the schema.
- Each record (tuple) is a row of the relation, and each attribute is a column.
- Each attribute has a domain consisting of a finite set of possible values within a few primitive types.
- Each attribute might have constraints (important for safety).
- The schema of the database is the set of relation schemas.
- The relations, not just the individual observations, are of interest.

Relational model



Importance of constraints

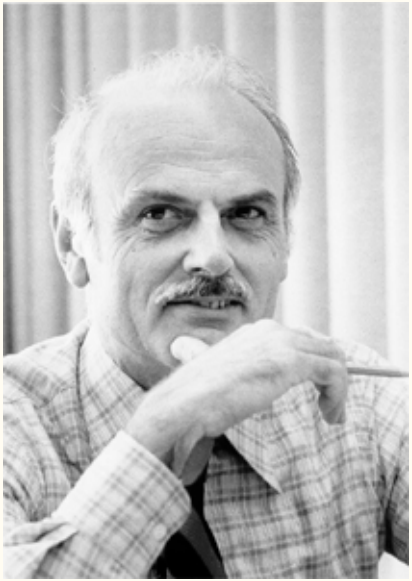


Relational model, algebra, and calculus

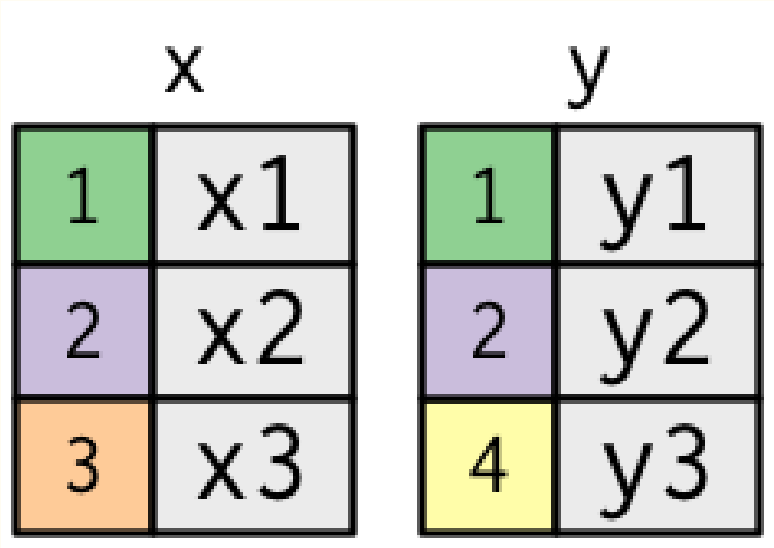
Built around two elements:

1. Relational model:
    1.1 Proposed by Edgar F. Codd (1969, Turing Award 1981).
    1.2 Data is organized as tuples grouped into relations and independent of physical properties of storage.
    1.3 Consistent with first-order predicate logic.
2. Relational algebra and calculus:
    2.1 Proposed, again, by Edgar F. Codd (1972).
    2.2 A collection of operations (mutating joins, filtering joins, and set operations).
    2.3 A way defining logical outcomes for data transformations.

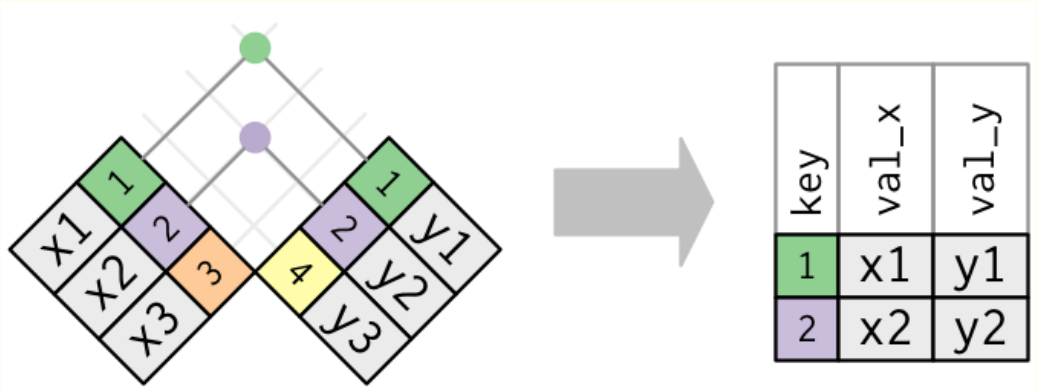
Edgar F. Codd (1923-2003)



Example of data

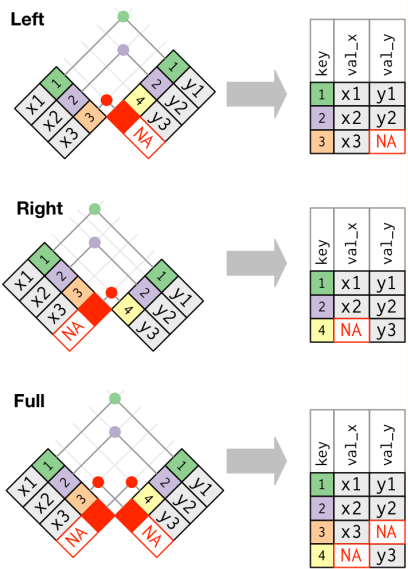


Inner join

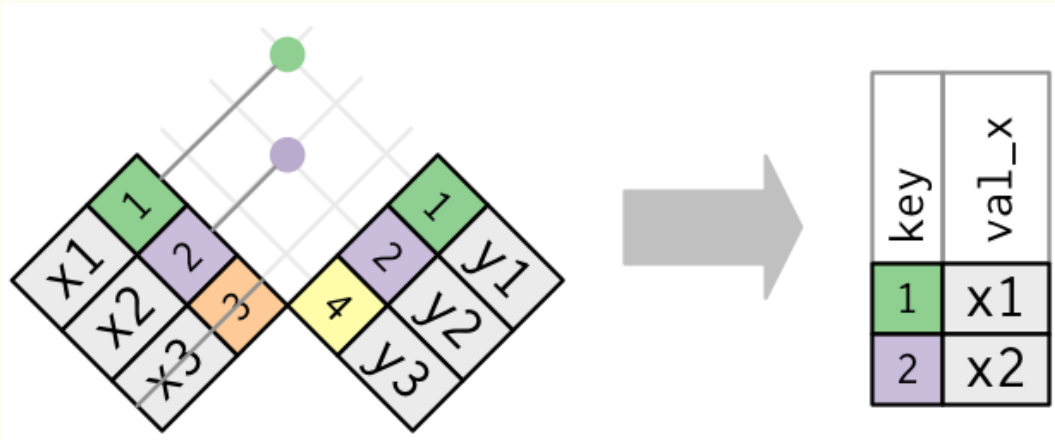


key	val_x	val_y
1	x1	y1
2	x2	y2

Outer joins


Left
x1
x2
x3
y1
y2
y3
NA
key val_x val_y
1 x1 y1
2 x2 y2
3 x3 NA
Right
x1
x2
x3
y1
y2
y3
NA
key val_x val_y
1 x1 y1
2 x2 y2
4 NA y3
Full
x1
x2
x3
y1
y2
y3
NA
NA
key val_x val_y
1 x1 y1
2 x2 y2
3 x3 NA
4 NA y3

Semijoin



Most popular relational database management systems

SQL I

- SQL (Structured English Query Language) is a domain-specific language for defining, managing, and manipulating data in relational databases.
- Developed at IBM in the early 1970s. Popularized by Oracle in the late 1970s.
- Based on Codd's twelve rules (actually, 13, from 0 to 12) of an RDBMS:

  https://computing.derby.ac.uk/c/codds-twelve-rules/.
- Current standard: SQL:2016. Check https://modern-sql.com/.
- Many different implementations (both open source and commercial) with some differences in syntax and adherence to current standards.

SQL II

- Good implementations follow the `ACID` (Atomicity, Consistency, Isolation, Durability) standard:
  1. Atomicity: either all operations in the database succeed, or none do.
  2. Consistency: a transaction in the database cannot leave the database in an inconsistent state.
  3. Isolation: one transaction in the database cannot interfere with another.
  4. Durability: a completed transaction persists, even after applications restart.
- Thus, you can understand `SQL` as choosing consistency over availability in the CAP theorem (although "consistency" in `ACID` and the CAP theorem are slightly different concepts). Most likely, the right choice in research.

SQL III

- Morover SQL has more procedural instructions than originally.
- In fact, SQL, after the introduction of Persistent Stored Modules (PSMs), is Turing complete.
- Also, over time, SQL has incorporated many objected-oriented features $\Rightarrow$ object-relational database management system (ORDBMS).
- Distributed computation: Apache Drill at https://drill.apache.org/ (also for many NoSQL databases).
- You can try basic SQL instructions at http://sqlfiddle.com/.

Open-source implementations I: PostgreSQL

- Current release: 10.5, available at `https://www.postgresql.org/`.
- Evolved from the Interactive graphics and retrieval system (Ingres) project at Berkeley, led by Michael Stonebraker (Turing Award 2014).
- Powerful ORDBMS implementation that can handle the most complex tasks.
- Available for all OS (for instance, it is the default in `macOS` Server).
- Highly extensible: user-defined data types, custom functions, and allows for programming in different languages (including the definition of DSLs).
- Many add-ons, such as the `PostGIS` geospatial database extender.
- Multiversion concurrency control (less critical for economists unless you have many coauthors and RAs).

Open-source implementations II: SQLite

- Available at https://sqlite.org/about.html, but pre-installed in macOS and most Linux distributions.
- Current release: 3.25.2.
- Uses PostgreSQL as a reference platform, but SQLite is serverless.
- Extremely popular, as it does not require a client-server engine (it is contained in a C programming library), and its installation is rather compact and with "zero configuration." Attractive features for economics.
- Bindings for all popular programming languages.
- Faster than regular file I/O in your operating system with a carefully designed application file format: a complete SQLite database is stored in a single cross-platform disk file.

Some SQLite instructions: basic interaction I

Getting Started with SQL: A Hands-On Approach for Beginners, by Thomas Nield.

To lunch the command-line shell

```
sqlite3
```

You can also add commands after `sqlite3` as in any other `Unix/Linux` program.

To exit:

```
sqlite> .exit
```

Alternative GUI $\Rightarrow$ `SQLite Studio`: `https://sqlitestudio.pl/`

Some SQLite instructions: basic interaction II

Help:

```
sqlite> .help
```

To read commands from script files:

```
sqlite> .read myfile
```

To print a string:

```
sqlite> .print STRING
```

To load a file:

```
sqlite> .output FILENAME
```

Finally, to comment:

```
sqlite> -- This is a comment
```

Some SQLite instructions: basic interaction III

To check existing databases and associated files:

```
sqlite> .databases
```

To create a database

```
sqlite3 Economists.db
```

To check existing tables:

```
sqlite> .tables
```

To check the schema of tables:

```
sqlite> .schema
```

In practice, you automatize the task described below with script files and mix in your favorite programming language.

Some SQLite instructions: DDL - Data Definition Language I

To create a table:

```
sqlite> CREATE TABLE Faculty (
Name TEXT,      NOT NULL,
Age INTEGER CHECK (Age=>0 and Age<100),
Field CHAR (20),
PhD CHAR (25),
PRIMARY KEY(Name),
FOREIGN KEY(id));
```

Note:

1. Capital case, optional (`SQLite` is mainly case insensitive) but common.
2. Keys are also optional.
3. `SQLite` uses dynamic typing. Most `SQL` database engines use static, rigid typing. I am following the standard typing convention in `SQL` and relying on affinity rules here.

Some SQLite instructions: DDL - Data Definition Language II

Beyond the standard types (text, integer, character, XML,...), we can define our own types

```
sqlite> CREATE ROW TYPE personalAddress (
Street CHARACTER VARYING (25),
City CHARACTER VARYING(20),
State CHARACTER (2),
PostalCode CHARACTER VARYING (9));
```

To alter a table (note: some of the options of ALTER TABLE are not supported by SQLite):

```
sqlite> ALTER TABLE Faculty ADD COLUMN Phone INTEGER;
sqlite> ALTER TABLE Faculty ADD COLUMN personalAddress
    addr_type;
```

To drop a table:

```
sqlite> DROP TABLE Faculty;
```

Some SQLite instructions: DML - Data Manipulation Language

To insert a record:

```
sqlite> INSERT INTO Faculty (Name, Age, Field, PhD)
VALUES('Adam Smith', 35, 'Economics', 'Glasgow');
VALUES('David Ricardo', 42, 'Economics', 'London');
```

To modify the record:

```
sqlite> UPDATE Faculty SET Name = 'David Ricardo' WHERE
    Name = 'Adam Smith';
sqlite> UPDATE Faculty SET Age = Age+1;
```

To delete the record:

```
sqlite> DELETE FROM Faculty WHERE Field = 'Economics';);
```

Some SQLite instructions: DQL - Data Query Language I

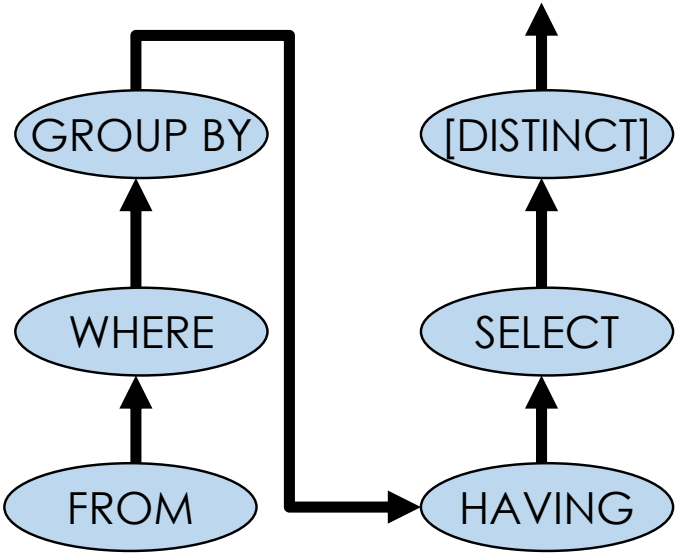
To list records:

```
sqlite> SELECT Name, Field FROM Faculty;
```

To select records:

```
sqlite> SELECT * FROM Faculty ORDER BY Age ASC;
sqlite> SELECT * FROM Faculty WHERE Age>50;
sqlite> SELECT * FROM Faculty WHERE Age>50 ORDER BY Age DESC;
sqlite> SELECT Name FROM Faculty WHERE  Name ~ 'A.*'
sqlite> SELECT MIN(Age) FROM Faculty;
sqlite> SELECT MAX(Age) FROM Faculty WHERE Field = 'Economics';
sqlite> SELECT Field AVG(Age) FROM Faculty GROUP by Field;
sqlite> SELECT Field AVG(Age) FROM Faculty GROUP by Field HAVING COUNT(*)>2;
sqlite> SELECT Field AVG(Age) AS avg_age, COUNT(*) as size FROM Faculty
    GROUP WHERE Age>50 by Field HAVING COUNT(*)>2 ORDER BY Age DESC;
```

Some SQLite instructions: DQL - Data Query Language II



Some SQLite instructions: DQL - Data Query Language III

To (inner) join records:

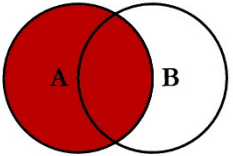
```
sqlite>  SELECT Name Dues FROM Faculty INNER JOIN
    AmericanEconomicAssociation on Faculty.Name =
    AmericanEconomicAssociation.Name;
```

Similar instructions for cross and outer joins.

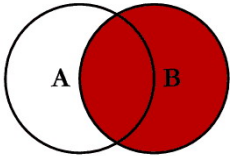
You can insert NULL

```
sqlite>  INSERT INTO Faculty (Name, Age, Field, PhD)
VALUES('J.M. Keynes', NULL, 'Economics', 'Cambridge');
sqlite>  SELECT * FROM Faculty WHERE Age IS NOT NULL;
```

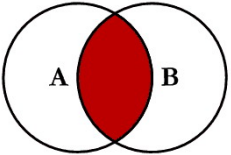
SQL JOINS



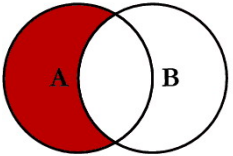

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



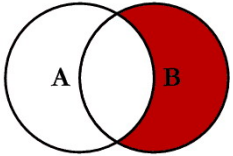

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



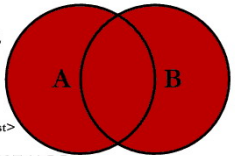

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



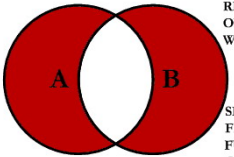

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Some SQLite instructions: DQL - Data Query Language IV

You can create your own views:

```
sqlite>  CREATE VIEW Econ_Faculty_View AS
SELECT Name, Age
FROM Faculty
WHERE Field = 'Economics';
```

The select can be as sophisticated as you want or subselect from the view.

```
sqlite> SELECT * FROM Econ_Faculty_View;
```

You cannot, however, to DELETE, INSERT or UPDATE statements on a view.

To drop a view:

```
sqlite>  DROP VIEW Econ_Faculty_View;
```

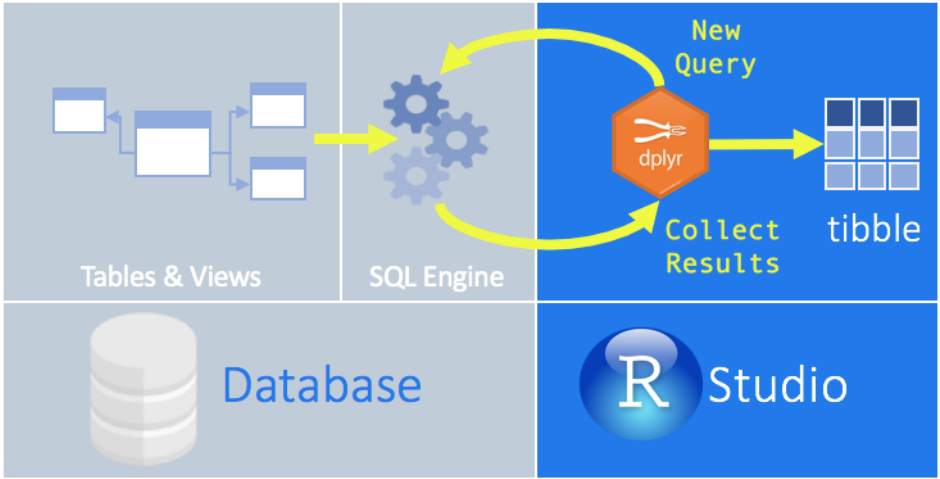
SQL and R I

- You can run SQL in R or R in the SQL server.
- The former approach is more common in research.
- Check:
  1. https://db.rstudio.com/.
  2. https://datacarpentry.org/R-ecology-lesson/05-r-and-databases.html.
- In addition, new versions of RStudio integrate interaction with SQL.

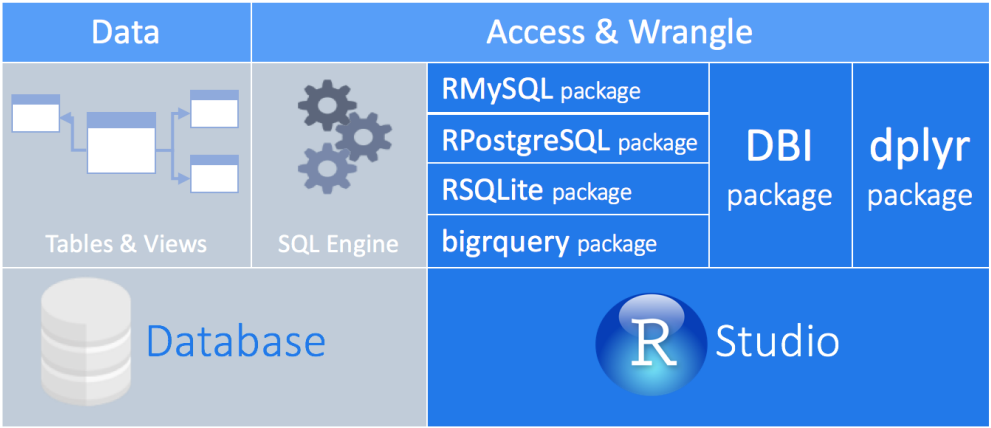
SQL and R II

- Package `dplyr`: provides a flexible grammar of data manipulation centered around data frames. In particular, `dplyr` allows you to translate the `dplyr` verbs into `SQL` queries and use the `SQL Engine` to run the data transformations. You need to install `dbplyr` (a backend for databases) as well: it translates `R` code into database-specific variants.
- Package `RSQLite`: embeds the `SQLite` database engine in R and provides an interface compliant with the `DBI` package (a database interface definition for communication between R and relational database management systems).
- Package `odbc`: provides a DBI-compliant interface to `Open Database Connectivity (ODBC)` drivers, including `SQL Server`, `Oracle`, and `MySQL` (and also `PostgreSQL`, `SQLite`).
- Package `dbplot`: allows processing a plot's calculations inside a database.

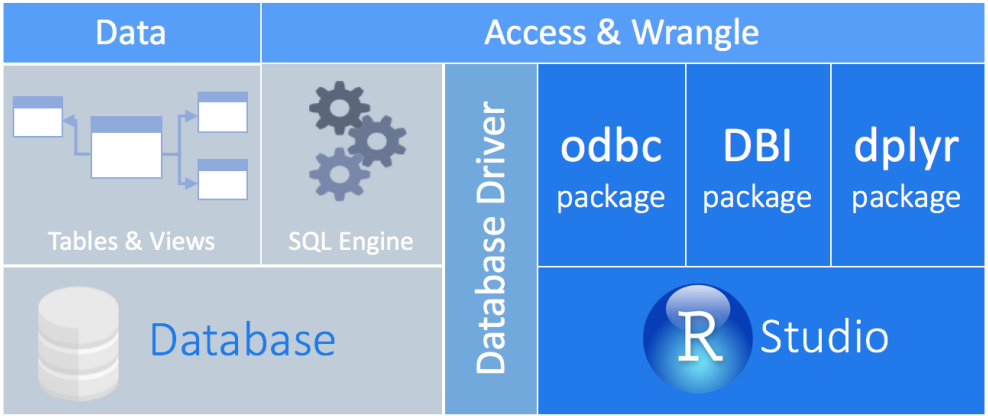
Use dplyr to interact with the database



Open Source Databases



Commercial Databases



SQLite in R I

Let us first clear everything:

```
rm(list=ls())
```

We install required R packages:

```
install.packages(c("dplyr", "dbplyr", "RSQLite"))
```

We load relevant packages

```
library(dplyr)
library(dbplyr)
library(RSQLite)
```

SQLite in R II

We download a standard SQLite database used to teach and install it in a new directory:

```
dir.create("data_class_computation", showWarnings = FALSE)
download.file(url = "https://ndownloader.figshare.com/
    files/2292171", destfile = "data_class_computation/
    portal_mammals.sqlite", mode = "wb")
```

We connect R to SQLite:

```
mammals <- DBI::dbConnect(RSQLite::SQLite(), "data_class_
    computation/portal_mammals.sqlite")
```

We inspect the database:

```
src_dbi(mammals)
```

SQLite in R III

We select some observations with SQL syntax:

```
mySelection <- tbl(mammals, sql("SELECT year, species_id,
    plot_id FROM surveys"))
```

We look at the top 5 observations:

```
head(mySelection, n = 5)
```

But it is easier to select with with dplyr syntax:

```
mySelection <-  tbl(mammals, "surveys")
```

We can look again at the top 5 observations:

```
head(mySelection, n = 5)
```

You can also load the query in a R Notebook.

SQLite in R IV

We pipe the selection:

```
mySelection %>%
  filter(weight < 5) %>%
  select(species_id, sex, weight)
```

We link across tables:

```
species <- tbl(mammals, "species")

left_join(mySelection, species) %>%
  filter(taxa == "Rodent") %>%
  group_by(taxa, year) %>%
  tally %>%
  collect()
```

`dyplr` allows easy implementation of all four joins for dataframes.

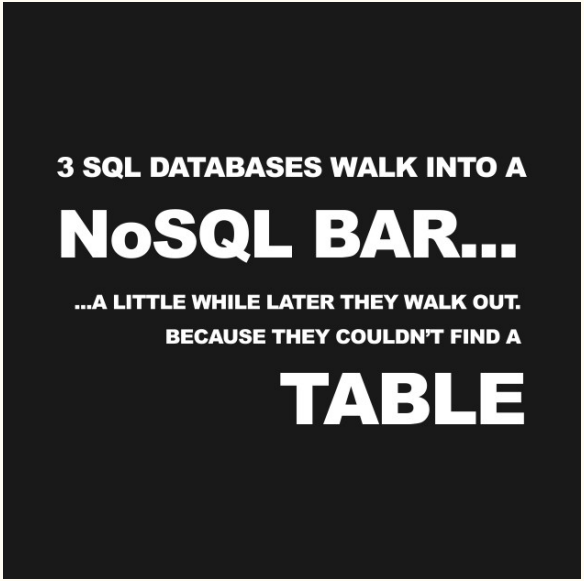
Alternatives

- `DuckDB`.
- It supports analytical query workloads, also known as Online analytical processing (OLAP).
- Complex, relatively long-running queries that process significant portions of the stored dataset.
- It interacts well with `R`: `duckdb`.

NoSQL I

- Databases not based on tabular relations.
- Originally, it meant `No+SQL`.
- Today most `NoSQL` databases include some `SQL` features, so most people call it `Not only SQL`.
- Concept existed since the 1960s (such as hierarchical databases), but it became popular in the early 2000s.
- Interesting example of move towards *less* abstraction.
- Why?
  1. Usually better at dealing with big and distributed data because of their capability to scale and parallelize.
  2. Schemaless data representations require less planning and allow for easier ex-post adjustments.
  3. Faster to code.

So you remember NoSQL



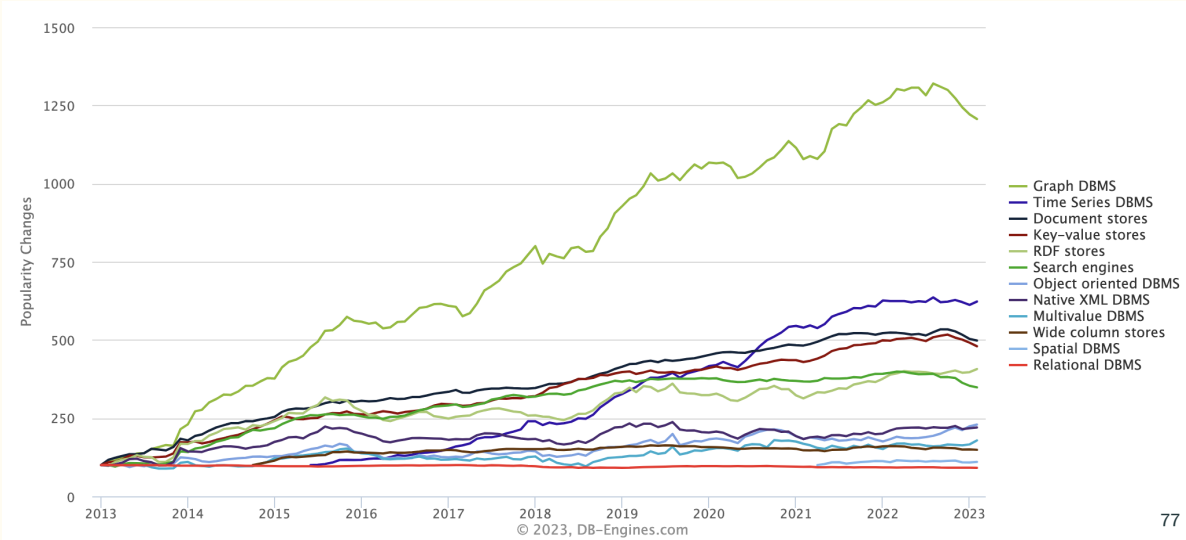
NoSQL II

- Instead of ACID, `NoSQL` follows BASE:
  1. Basic availability: each request gets a response (successful or not).
  2. Soft state: the state of the database changes over time, even without any input (for eventual consistency).
  3. Eventual consistency: the database may be momentarily inconsistent but will eventually reach consistency.
- Some `NoSQL` such as `Neo4j`, though, still deliver ACID.
- `NoSQL` chooses availability over consistency in the CAP theorem. Note the importance of web applications.

NoSQL III

- NoSQL databases systems include a wide set of alternative approaches:
  1. Document stores: schema-free organization of data ⇒ MongoDB, Couchbase.
  2. Key-value stores: pairs of keys and values ⇒ Redis, Memcached.
  3. Wide column stores: store data in records with huge numbers of dynamic columns ⇒ Cassandra, HBase.
  4. Time series DBMS: optimized for handling time series data: each entry is associated with a timestamp ⇒ InfluxDB, Graphite.
  5. Graph DBMS: represent data in graph structures as nodes and edges. ⇒ Neo4j, AllegroGraph.
  6. XML ⇒ MarkLogic, BaseX.
  7. Search engines ⇒ Elasticsearch, Splunk.
  8. Multimodel ⇒ Amazon DynamoDB, Microsoft Azure Cosmos DB.
- Also, object databases (although they have not taken off).

NoSQL: trends



NoSQL in economics

- Uses in economics:
  1. Graph databases, for their potential to allow us to discover important relational patterns.
  2. Time Series DBMS, to deal with financial and other high-frequency data.
  3. Data collections that might change over time in structure.
- Additional references:
  1. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement (2nd Edition)*, by Luc Perkins with Eric Redmond and Jim Wilson.
  2. *Next Generation Databases: NoSQL, NewSQL, and Big Data*, by Guy Harrison.

MongoDB

- `MongoDB` (from "humongous"), most popular NoSQL database.
- Current release: 4.0, `https://www.mongodb.com/`.
- Built around `BSON, Binary JSON`, a version of `JSON`.
- Dual structure:
  1. Documents are stored in collections using the `BSON` format. A collection is a group of related documents with shared indices.
  2. MongoDB collections belong to a database.
- Used, for example, by CERN to collect data from the Large Hadron Collider.
- Versatile and easy to use (expressive language for queries).

Mongo data model

NoSQL and R

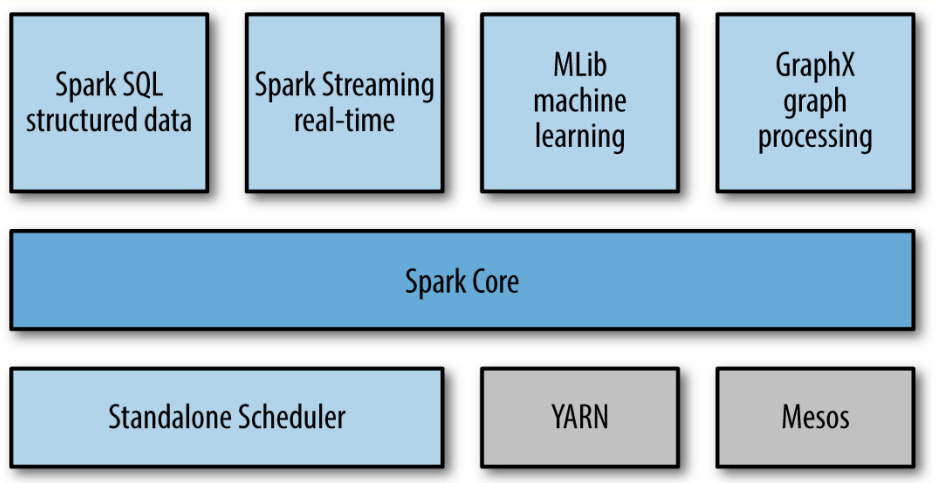
- Less polished support than for SQL.
- Package nodbi for general backend.
- For MongoDB, we have package mongolite:

```
install.package("mongolite")
library(mongolite)
m <- mongo("mtcars", url = "mongodb://readwrite:test@mongo
    .opencpu.org:43942/jeroen_test")
alldata <- dmd$find('{}')
print(alldata)
test <- dmd$find(
  query = '{"cut" : "Premium"}',
  fields = '{"cut" : true, "clarity" : true}',
  limit = 5)
print(test)
```

Spark

- Available at `https://spark.apache.org/`.
- Current version: `2.3.2`.
- A fast and general-purpose cluster computing system.
- Modern alternative to `Hadoop` (although without a file management system).
- High-level APIs in `Java`, `Scala`, `Python`, and `R`.
- Interacts well with `SQL` and has a beautiful machine learning library, `MLlib`.
- Allows for real-time processing and querying.
- *Learning Spark: Lightning-Fast Big Data Analysis* by Holden Karau and Andy Konwinski.

Spark stack


Spark SQL
structured data
Spark Streaming
real-time
MLib
machine
learning
GraphX
graph
processing
Spark Core
Standalone Scheduler
YARN
Mesos

RDDs

- Organized around resilient distributed datasets (RDDs).
- An RDD is a collection of items distributed across computer nodes that can be manipulated in parallel.
- Operations: transformations ("map", "filter") and actions ("count", "collect").
- Why resilient? Automatically rebuilt on failure.
- It can be stored on disk or memory.
- Completely lazy evaluation.

Spark example code

```
def  inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \
             .filter(inside).count()
print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Spark and R


sparklyr
dplyr
GraphX
(graphframes)
Streaming
ML
Extensions
APACHE
Spark

Spark in R I

Let us first clear everything:

```
rm(list=ls())
```

We install required Spark package:

```
install.packages("sparklyr")
```

We load the relevant package and install Spark:

```
library(sparklyr)
spark_install(version = "2.3.0")
```

Spark in R II

We contect to Spark:

```
sc <- spark_connect(master = "local")
```

We install the package with some cute data:

```
install.packages(c("nycflights13"))
```

We load the relevant package and install Spark:

```
library(dplyr)
flights_tbl <- copy_to(sc, nycflights13::flights, "flights
    ")
src_tbls(sc)
```

Some piping:

```
flights_tbl %>% filter(dep_delay == 2)
```

Machine learning

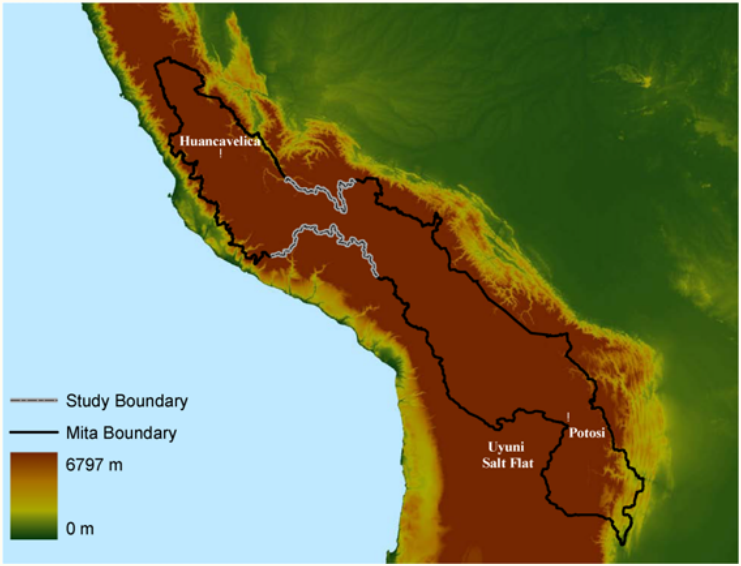
- Many of the most popular algorithms in machine learning are coded in reliable, state-of-the-art libraries.
- Most famous:
  1. `Tensorflow`, `https://www.tensorflow.org/`.
  2. `scikit-learn`, `http://scikit-learn.org/stable/`.
- Note, however, that if you are going to write frontier papers in machine learning, chances are you will need to write much (most?) of your code.

GIS

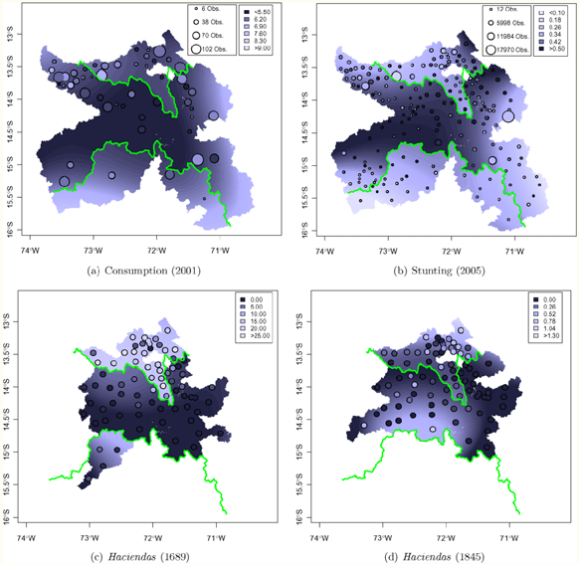
- Geographic information systems (GIS) capture, store, manipulate, and display geographic information data.
- Goes back to John Snow's 1855 map of the Soho cholera outbreak.
- Why the current boom? Spatial econometrics and quantitative spatial economics:
  1. *A Primer for Spatial Econometrics: With Applications in R*, by Giuseppe Arbia.
  2. Redding and Rossi-Hansberg (2017).
- Resources:
  1. `https://www.gislounge.com/`.
  2. `https://gisgeography.com/`

John Snow, cholera epidemics 1858

The effects of the Mita I



The effects of the Mita II



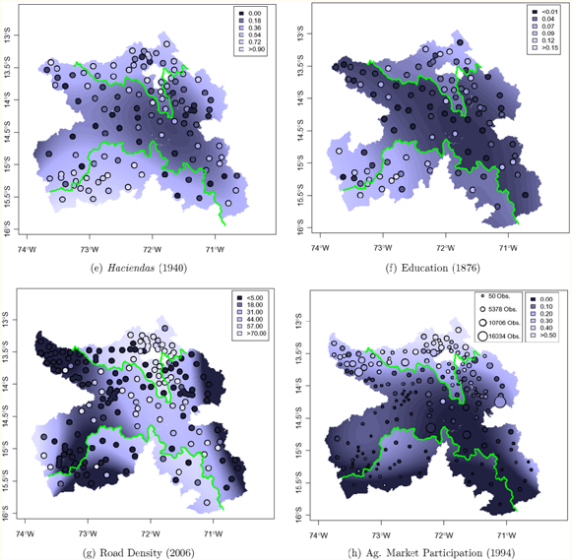
(a) Consumption (2001)

(b) Stunting (2005)

(c) *Haciendas* (1689)

(d) *Haciendas* (1845)

The effects of the Mita III



The effects of the Mita IV

	Dependent Variable						
	Log Equiv. Household Consumption (2001)			Stunted Growth, Children 6–9 (2005)			
Sample Within:	<100 km of Bound. (1)	<75 km of Bound. (2)	<50 km of Bound. (3)	<100 km of Bound. (4)	<75 km of Bound. (5)	<50 km of Bound. (6)	Border District (7)
	Panel A. Cubic Polynomial in Latitude and Longitude						
Mita	−0.284	−0.216	−0.331	0.070	0.084*	0.087*	0.114**
	(0.198)	(0.207)	(0.219)	(0.043)	(0.046)	(0.048)	(0.049)
R^2	0.060	0.060	0.069	0.051	0.020	0.017	0.050
	Panel B. Cubic Polynomial in Distance to Potosí						
Mita	−0.337***	−0.307***	−0.329***	0.080***	0.078***	0.078***	0.063*
	(0.087)	(0.101)	(0.096)	(0.021)	(0.022)	(0.024)	(0.032)
R^2	0.046	0.036	0.047	0.049	0.017	0.013	0.047
	Panel C. Cubic Polynomial in Distance to *Mita* Boundary						
Mita	−0.277***	−0.230**	−0.224**	0.073***	0.061***	0.064***	0.055*
	(0.078)	(0.089)	(0.092)	(0.023)	(0.022)	(0.023)	(0.030)
R^2	0.044	0.042	0.040	0.040	0.015	0.013	0.043
Geo. controls	yes	yes	yes	yes	yes	yes	yes
Boundary F.E.s	yes	yes	yes	yes	yes	yes	yes
Clusters	71	60	52	289	239	185	63
Observations	1478	1161	1013	158,848	115,761	100,446	37,421

QGIS

- QGIS, current version: 3.28.
- Check https://qgis.org/en/site/. Also, note large number of plugins.
- Works with PostGIS, which adds support for geographic objects to the PostgreSQL.
- Alternative: to work directly in Python or R.
- Check https://www.jessesadler.com/post/gis-with-r-intro/.