Mark Raasveldt & Hannes Mühleisen

DuckDB – The SQLite for Analytics

about:me

- Postdoc in Database Architectures group
- Before that: PhD student
- Before that: Master student

- I like the CWI
- I like databases

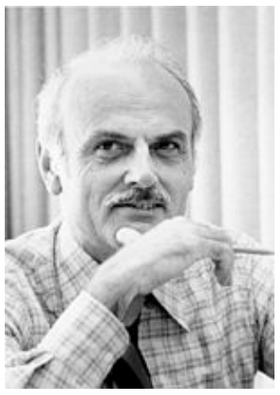
Relational DBMS

- In our group we work on **RDBMS**
  - Relational DataBase Management Systems

- Many of you have probably used a RDBMS
- Taught in schools, used everywhere

CWI Relational DBMS

- RDBMS have been around forever (in CS terms)
- **1970**: Edgar F. Codd presented the relational model



Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of rela-

CWI Relational DBMS

- **1974:**

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

Relational DBMS

- That’s it!
- Since 1974 little has changed:
- SQL/relational model are **still** used everywhere
- What have database researchers been doing!?

Relational DBMS

- SQL is declarative
  - You describe **what** you want to see
  - Not **how** to get the data
- DBMS **decides** the best way of answering queries

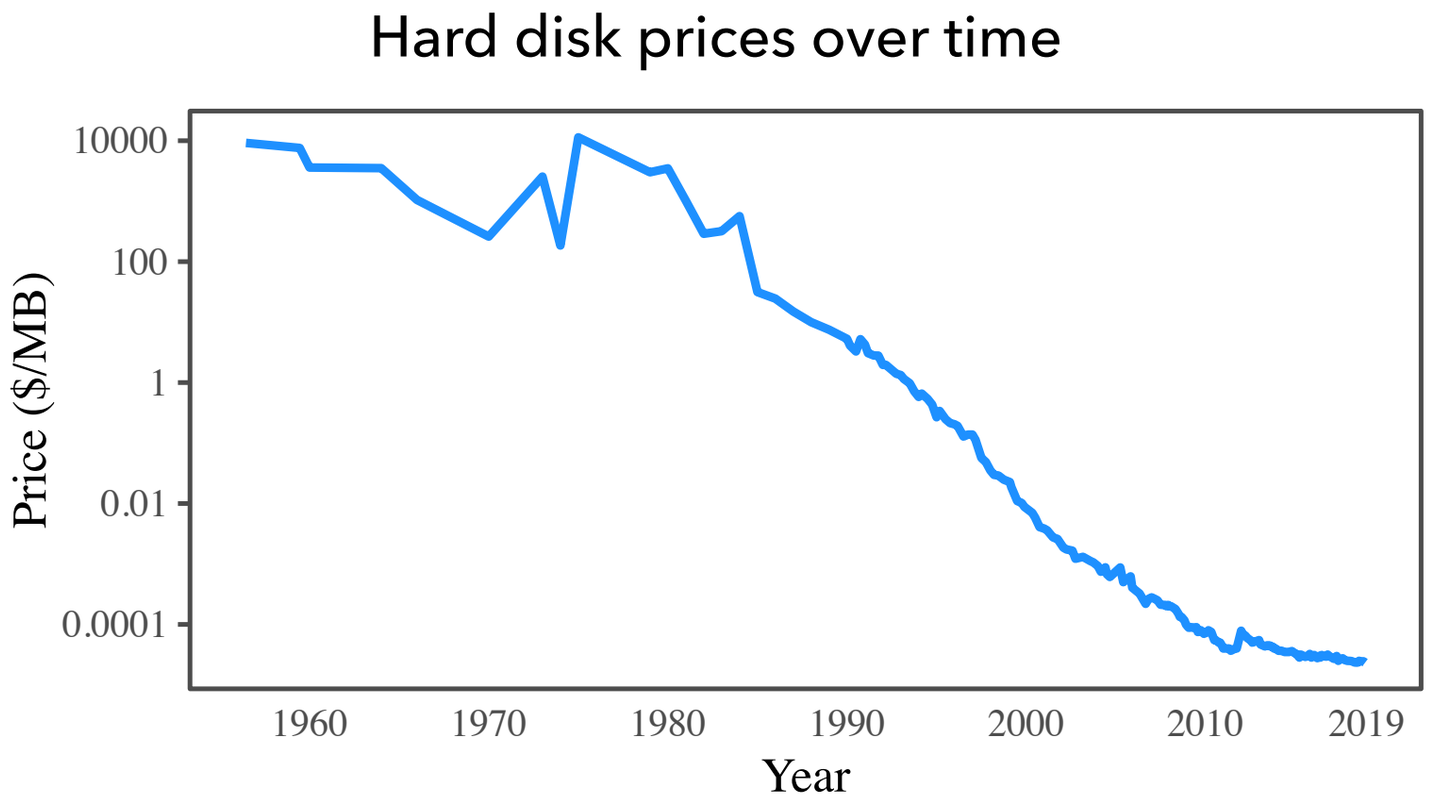
Relational DBMS

- While SQL has not changed, DBMS have!
- Database community has been optimizing these systems for **half a century**

- Original DBMS were made for computers with **kilobytes** of memory
- Modern DBMS are made for computers with **terabytes** of memory (or clusters of computers)

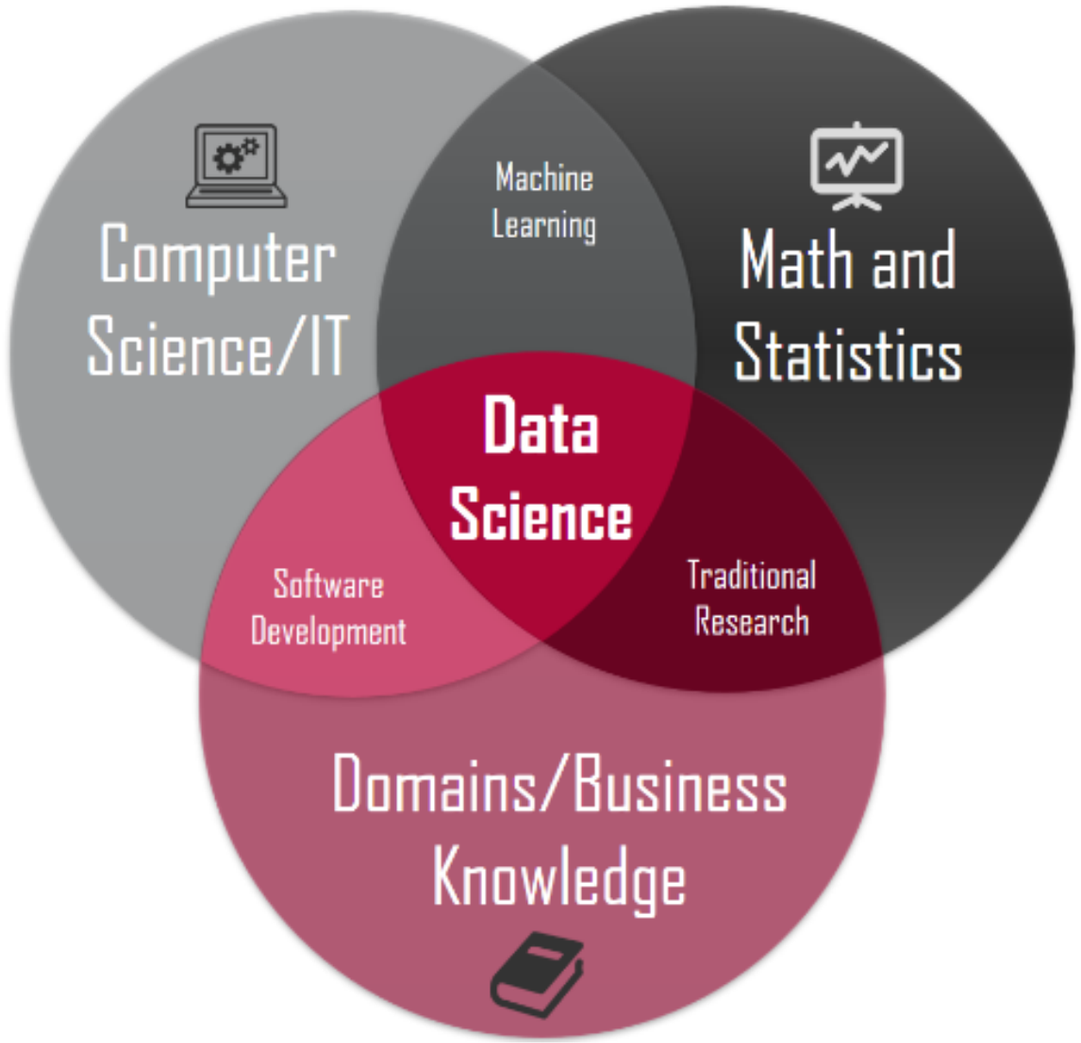
Data Science

- Data has been getting increasingly hyped/popular
- Main reason: storing/analyzing data is cheap now
  - You can store and analyze gigabytes on standard laptops/desktops



Data Science

- This new trend is called **data science**
- Overloaded term



- Data science = Statistics on a macbook

CWI Data Science

- Data Scientists work with **data**
- *Our field:* 50+ years perfecting RDBMS

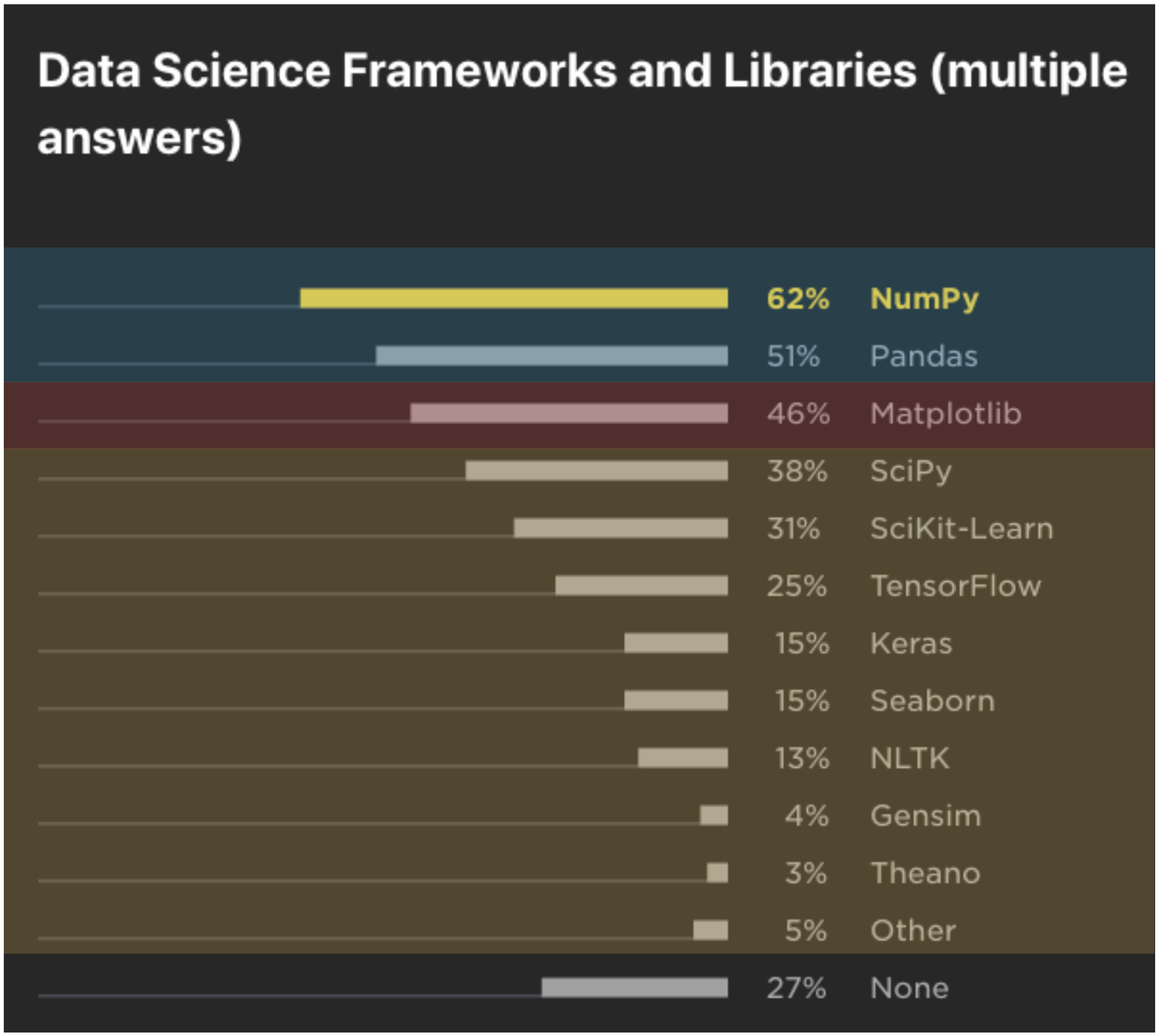
- **Obviously** they use RDBMS!
- … right?

- No!
- They use them if forced…
  - Data is in a RDBMS
- Prefer to avoid using them

"If your data fits in memory there is no advantage to putting it in a database: it will only be slower and more frustrating"
- Hadley Wickham

- Maybe they just **don't need** RDBMS technology
- What do data scientists actually use?

Data Science



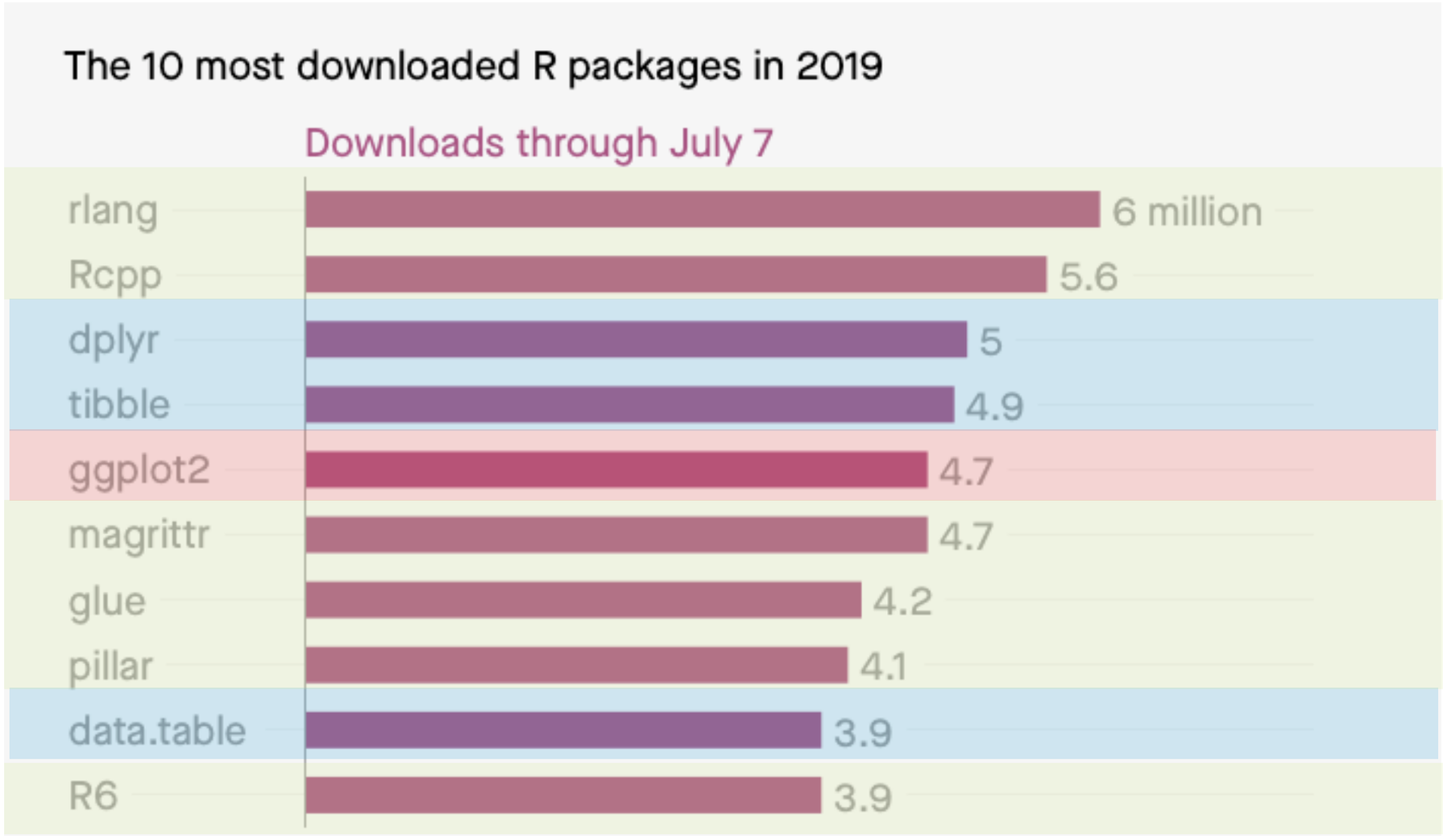
Data Wrangling

Plotting

Analysis
Classification
ML

[1] https://www.jetbrains.com/research/python-developers-survey-2018/

CWI Data Science



Development

Data Wrangling

Plotting

Data Science

- What do pandas, NumPy, dplyr, tibble and data.table have in common?

- They perform **database operations**
- Joins, aggregates, basic data transformations, filters, etc.

- Data scientists **need** RDBMS functionality!

- Let’s go back to Hadley’s quote

“If your data fits in memory there is no advantage to putting it in a database: it will only be slower and more frustrating”

- Hadley Wickham

- Why does he think that?
- We have been optimizing them for 50 years!

Data Science Workflow

- Let’s run a data science workflow
- With and without a database system
- And see what happens!

Data Science Workflow

- **Data Science Workflow**
- Load data from CSV file
- Perform preprocessing
  - **ETL**: Extract Transform Load
- Run analysis

Data Science Workflow

- **Our example:**
- Voter data from North Carolina
  - 360MB in CSV format
- Preprocessing:
  - Remove entries with missing phone number or house number
- Compute correlation between phone number and house number

Data Science Workflow

- **Pure R**
- First, install required packages:

```
install.packages(c("data.table", "dplyr"))
```

Data Science Workflow

- **Pure R**
- Then run the analysis

```
# load data from CSV file
ncvoter <- fread('ncvoter.csv', sep='\t')
# filter entries with missing phone/house number
ncvoter %>% filter(!is.na(phone_num), !
is.na(house_num)) -> ncvoter_filtered
# perform the correlation
cor(ncvoter_filtered[['phone_num']],
ncvoter_filtered[['house_num']])
 [1] -0.002638091
```

Data Science Workflow

- **PostgreSQL**
- Install the database server
  - Not trivial without package manager!
  - Download sources and compile

```
$ export PGDATA=ncvoters
$ initdb
$ pg_ctl -D ncvoterdb -l logfile start
```

CWI

Data Science Workflow

- **PostgreSQL**
- Next up: install client

```
install.packages("RPostgres")
```

Data Science Workflow

PostgreSQL

```
------------------------- ANTICONF ERROR ---------------------------
Configuration failed because libpq was not found. Try installing:
 * deb: libpq-dev (Debian, Ubuntu, etc)
 * rpm: postgresql-devel (Fedora, EPEL)
 * rpm: postgreql8-devel, psstgresql92-devel, postgresql93-devel, or postgresql94-devel (Amazon Linux
)
 * csw: postgresql_dev (Solaris)
 * brew: libpq (OSX)
If libpq is already installed, check that either:
(i)  'pkg-config' is in your PATH AND PKG_CONFIG_PATH contains
     a libpq.pc file; or
(ii) 'pg_config' is in your PATH.
If neither can detect , you can set INCLUDE_DIR
and LIB_DIR manually via:
R CMD INSTALL --configure-vars='INCLUDE_DIR=... LIB_DIR=...'
```

- That didn't work
- Need server installed and in PATH to run client

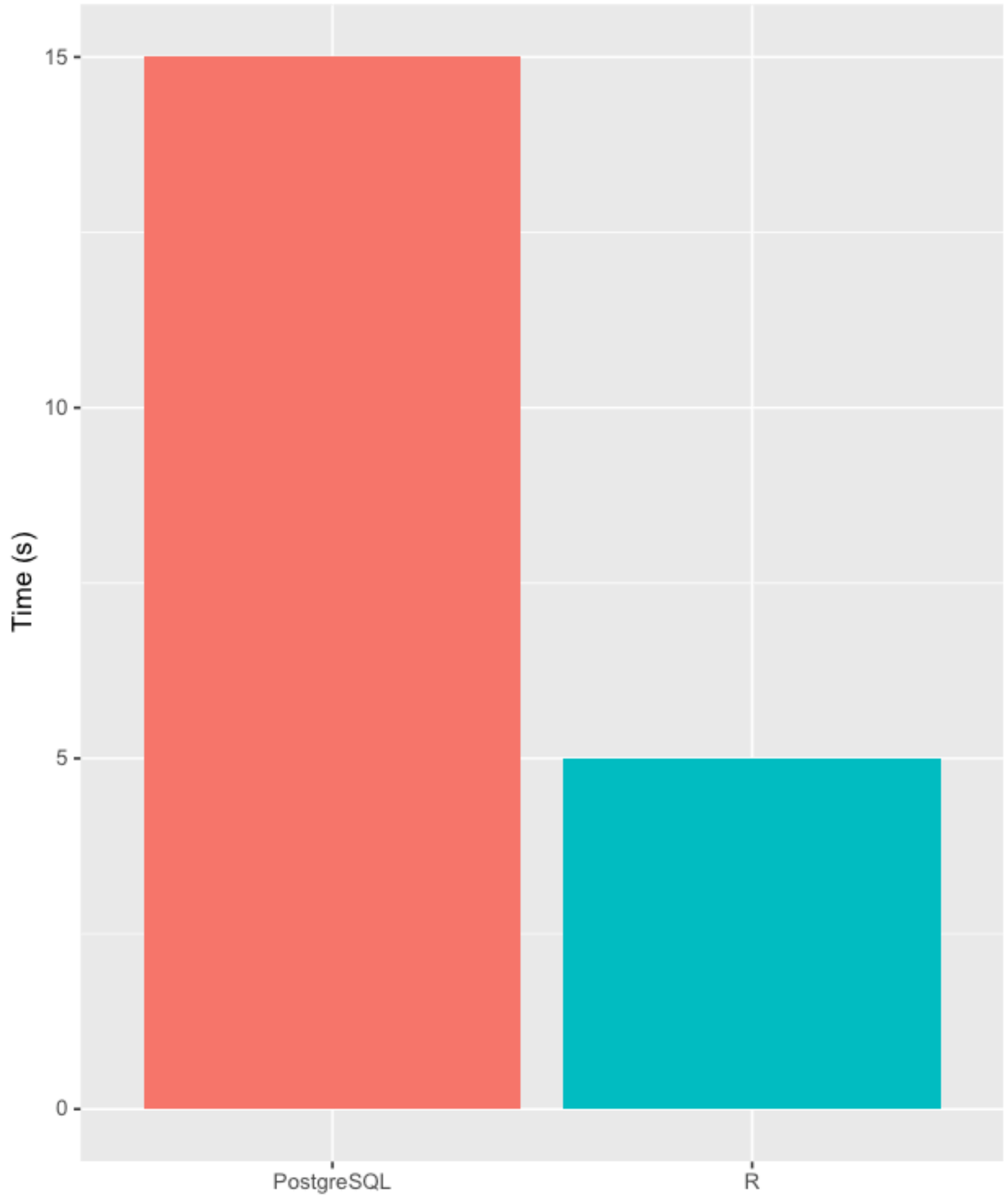
Data Science Workflow

- **PostgreSQL**

```
library(RPostgres)

con <- DBI::dbConnect(RPostgres::Postgres(), host='localhost',
dbname='postgres')
dbSendQuery(con, "CREATE TABLE ncvoters(...);")
dbSendQuery(con, paste0("COPY ncvoters FROM '", getwd(), "/",
"ncvoter.csv' (FORMAT 'csv', DELIMITER '|', QUOTE '\"', NULL '')"))
dbSendQuery(con, "DELETE FROM ncvoters WHERE phone_num IS NULL OR
house_num IS NULL")
ncvoter_filtered <- dbReadTable(con, "ncvoters")
cor(ncvoter_filtered[['phone_num']], ncvoter_filtered[['house_num']])
```

Data Science Workflow



Data Science Workflow

- Hadley is right
- For data science workflows, database system **are** frustrating and slow

- What are the problems?
- How can we fix them?

RDBMS Problems

- Problems
  - Difficult to install
  - Difficult to setup/maintain
  - Slow data transfer to/from client
  - Poor interface with client application
  - External state: cannot copy/paste!

RDBMS Problems

- Client/Server Architecture
- Good for multiple users/separate machine
- Pointless for single-user
- Makes install/setup much more difficult

RDBMS Problems

- Installation needs to be easy
  - `pip install` for python
  - `install.packages` for R

RDBMS Problems

- Transfer efficiency is crucial
- SQL is not sufficient for data science
- Data scientists use R/Python packages
- Connectors not built for bulk transfer

Don't Hold My Data Hostage –
A Case For Client Protocol Redesign

Mark Raasveldt
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
m.raasveldt@cwi.nl

Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

ABSTRACT

Transferring a large amount of data from a database to a client program is a surprisingly expensive operation. The time this requires can easily dominate the query execution time for large result sets. This represents a significant hurdle for external data analysis, for example when using statistical software. In this paper, we explore and analyse the result set serialization design space. We present experimental results from a large chunk of the database market and show the inefficiencies of current approaches. We then propose a columnar serialization method that improves transmission performance by an order of magnitude.

Keywords

Databases, Client Protocols, Data Export

1. INTRODUCTION

Transferring a large amount of data out of a database system to a client program is a common task. Examples include complex statistical analysis or machine learning applications that need access to large samples for model construction or verification. However, that operation is expensive. It is even more costly if the data is transported over a network connection, which is necessary if the database server runs on a separate machine or in the cloud.

Result set serialization (RSS) has a significant impact on overall system performance. Figure 1 shows the time taken to run the SQL query `"SELECT * FROM lineitem"` using an ODBC connector and then fetching the results for various data management systems. We see large differences between systems and disappointing performance overall. Modern data management systems need a significant amount of time to transfer a modest amount of data from the server to the client, even when they are located on the same machine.

From a survey of popular machine learning tools, we have found that none of them integrate into database systems beyond the bulk transfer of data sets. RapidMiner [18] and Weka [16] allow analyzing data from a database connection,

Proceedings of the VLDB Endowment, Vol. 10, No. 10

Netcat (10.25s)
MySQL 101.3
ODBC 170.9
Embed. V. 189.6
Postgr.SQL 202
MonetDB 221.3
MySQL+C 391.3
Hive 429.9
Mongo DB 686.5
Operation: Connection, Query Execution, ODBC Transfer
0, 200, 400, 600
Wall clock time (s)

Figure 1: Wall clock time for retrieving the `lineitem` table (SF10) over a loopback connection. The dashed line is the wall clock time for `netcat` to transfer a CSV of the data.

but this is strictly limited to loading. The users must issue their own queries to load the data from the database. This will likely be a query such as the one above. None of R's machine learning packages [19] use a database interface or import a package that does. Instead, they again rely on the user to provide them with data in a table that is already loaded into memory. The same is true for Python-based toolkits like SciKit-Learn [26] or TensorFlow [2].

Because of the large cost of data export, analysts settle for exporting small samples from the database. This way, data export is not a bottleneck in their analysis pipelines. However, this generally reduces the accuracy of machine learning and classification algorithms.

The issue of slow result export has been identified before. A large amount of previous work focuses on avoiding data export by performing the computations in the database instead of exporting the data [30]. However, these solutions require large, system-specific, overhauls of existing pipelines and are difficult to implement and optimize. There is little to no scientific work done on improving result set serialization.

In this paper, we perform a rigorous examination on the design of existing serialization formats. We analyse how they perform when transferring various data sets in different network scenarios, and examine why they show this performance. We explore the design space of serialization formats with experiments, and look at the different trade-offs that are made when designing a client protocol.

Don't Hold My Data Hostage
A Case For Client Protocol Redesign
VLDB2017

RDBMS Problems

- SQL is a nice language…
- But not as interface for clients!
- Embedding SQL as strings is unnatural
  - No syntax highlighting
- Whole class of security vulnerabilities:
  - SQL injection

CWI

RDBMS Problems

- SQL as a string is not easily composable
- dplyr allows us to chain operations:

```
ncvoter %>%
filter(!is.na(phone_num), !is.na(house_num)) %>%
select(phone_num+1)
```

- Easily inspect intermediates:

```
ncvoter %>%
filter(!is.na(phone_num), !is.na(house_num)) %>%
head
```

RDBMS Problems

- SQL: temporary tables

```
dbSendQuery(con, "CREATE TEMPORARY TABLE
ncvoters_filtered AS SELECT * FROM ncvoters WHERE
phone_num IS NULL OR house_num IS NULL")
dbGetQuery(con, "SELECT * FROM ncvoters_filtered
LIMIT 10")
```

RDBMS Problems

- Cannot copy/paste!

```
library(data.table)
library(dplyr)

# load data from CSV file
ncvoter <- fread('ncvoter.csv', sep='\t')
# filter entries with missing phone/house number
ncvoter %>% filter(!is.na(phone_num), !
is.na(house_num)) -> ncvoter_filtered
# perform the correlation
cor(ncvoter_filtered[['phone_num']],
ncvoter_filtered[['house_num']])
```

RDBMS Problems

- Cannot copy/paste!

```
library(RPostgres)

con <- DBI::dbConnect(RPostgres::Postgres(), host='localhost',
dbname='postgres')
dbSendQuery(con, "CREATE TABLE ncvoters(...);")
dbSendQuery(con, paste0("COPY ncvoters FROM '", getwd(), "/",
"ncvoter.csv' (FORMAT 'csv', DELIMITER '|', QUOTE '\"', NULL '')"))
dbSendQuery(con, "DELETE FROM ncvoters WHERE phone_num IS NULL OR
house_num IS NULL")
ncvoter_filtered <- dbReadTable(con, "ncvoters")
cor(ncvoter_filtered[['phone_num']], ncvoter_filtered[['house_num']])
```

RDBMS Problems

- Current RDBMS are not a good fit for data science
- Why do we (database researchers) care?

RDBMS Problems

- Data science is popular
- Popular = money $$

RDBMS Problems

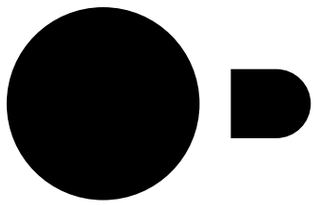
- Data scientists are re-inventing RDBMS
  - Poorly!

- No optimizers
- Huge intermediates
- No out-of-memory computation
- No parallelism
- No scalability

RDBMS Problems

- RDBMS can solve all of these problems
  - And more!
- Current RDBMS are “slow and frustrating”
- But they don’t need to be!

CWI



DuckDB
an Embeddable Analytical RDBMS

DuckDB

- Why “Duck” DB?
- Hannes used to own a pet duck

CWI DuckDB

- DuckDB: The SQLite for Analytics

- **Core Features**
- Simple installation
- Embedded: no server management
- Single file storage format
- Fast analytical processing
- Fast transfer between R/Python and RDBMS

- **Simple Installation & Usage**
- Many lessons learned from SQLite
- Embedded system
  - Zero external dependencies
    - All dependencies are inlined
  - Amalgamation (`duckdb.cpp` & `duckdb.hpp`)
  - Integrated into package managers:
    - `pip install duckdb`
      `install.packages("duckdb")`

- **Simple Installation & Usage**
- DuckDB does not rely on any external state
  - e.g. separate config files, environment variables

- **Single-File Storage Format**
- Files are easier to handle for users than directories
  - **See:** Microsoft Office

- **Composable Interface**

```
import duckdb

# open database file
db = duckdb.open('cities.db')

# filter on cities with a high population
fcities = db.table('cities').filter('population > 1000')
# print a snapshot
fcities.show()
# now perform a grouped aggregate
fcities.aggregate(‘country, sum(population)')
```

- **Composable Interface**

```
import duckdb

# tight integration with native structures
df = pandas.read_csv('cities.csv')
# directly scan a pandas dataframe
duckdb.from_df(df).filter('population > 1000')
```

- **Efficient ETL Workloads**
- Traditional systems are not optimized for ETL:
  - Bulk updates
  - Bulk deletions
  - Adding/removing columns
- e.g. a common query is replacing NULL values:

```
UPDATE tbl
SET val=NULL
WHERE val=-999;
```

- This can update the entire column!

- **Efficient Snapshotting for Undo**
- ETL workflows are generally trial-and-error
- When a mistake is made, user has to start over
- Typically involves reloading from CSV file again
  - This can take a very long time!

- **Efficient Snapshotting for Undo**
- RDBMS supports `ROLLBACK`/versioning of data
  - Current systems not optimized for many different versions of entire columns
- DuckDB has MVCC that is optimized for bulk updates/deletes to support this use case

CWI DuckDB

DuckDB

- DuckDB is free and open-source
- Currently in pre-release (v0.1.9)
- We have a website: www.duckdb.org
- Source Code: https://github.com/cwida/duckdb

- Feel free to try it
- And send us a bug report if anything breaks!