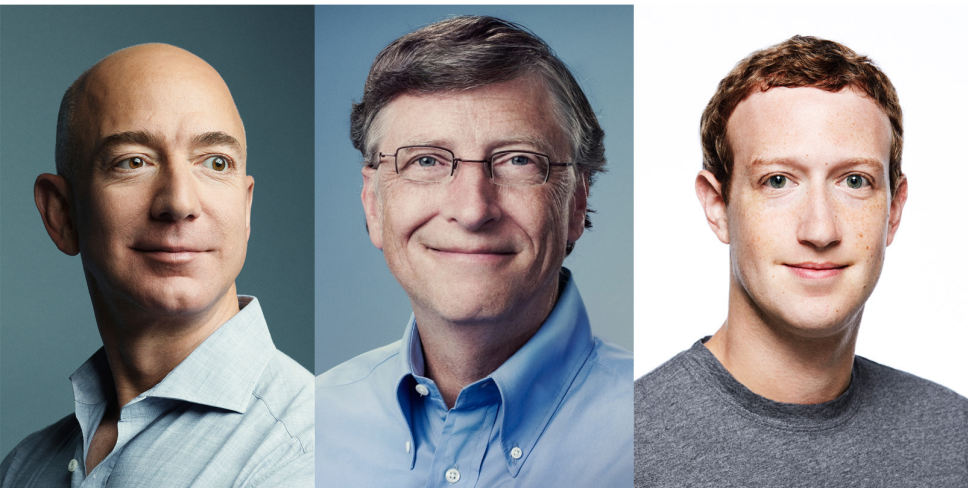


Relational Models and the Structured Query Language

Jamie Saxon

University of Chicago

November 7, 2017



Who are these people?





ORACLE®

Databases as Infrastructure

- ▶ Since they are not consumer-facing products, database systems fly under the radar like our energy infrastructure. But they are very, very important.
- ▶ They power the deep content of the internet: blogs, consumer applications, health services, etc.
 - ▶ Typical web application is server + database + scripting language.
- ▶ Huge problems in health services, government, etc. from failure to construct consistent or interoperable models.
 - ▶ This is totally solvable, but more political than technical!

Databases: or, Another Data Source

- ▶ Databases store data efficiently and retrieve it quickly.
 - ▶ Typical computers have 8-16 GB of memory: a hard limit on what you can simultaneously load. Databases allow efficient row-wise calculations.
 - ▶ Search efficiently over many GBs, or hundreds of GBs of data...
 - ▶ Maintain one copy of the data; allow many people to access it (servers).
 - ▶ Really huge datasets need different models than discussed today.
- ▶ In the 'relational model,' data are stored in interrelated 'tables.'
- ▶ Each record (row) is uniquely identified by a 'primary key' (index).
- ▶ Differs from composite objects in object-oriented programming, since many records in one table may link to a single record of another (e.g., people to a city) and several records may link in different ways (grade to teacher, student, and school).

Our Example: The American Time Use Survey

- ▶ ATUS is a subset of the Current Population Survey.
- ▶ Respondents record all activities performed on a “diary day.”
- ▶ Survey provides micro-data in tables including:
 - ▶ **CPS**: data from the CPS interviews.
 - ▶ **Roster**: Stats on each member of the household + non-hh children, and relationship to respondent.
 - ▶ **Respondent**: Detailed info and summary data on the respondent.
 - ▶ **Activity**: Detailed code and duration of each activity in the day.

Tables of the American Time Use Survey

Respondents				
ID	Line	Children	Worked	...
1	1	0	1	...
2	1	1	0	...
3	1	2	1	...
4	1	0	1	...
5	1	3	1	...
6	1	0	0	...
7	1	1	1	...
...

Activities				
Resp.	Activity #	Code	Duration	...
1	1	010101	150	...
1	2	130124	45	...
1	3	010201	30	...
1	4	020201	10	...
1	5	110101	15	...
1	6	180501	20	...
1
2	1	010101	240	...
...

CPS				
Resp.	Line	Age	Education	...
1	1	38	38	...
1	2	35	0	...
1	3	4	1	...
2	1	53	1	...
2	2	58	0	...
...

* Primary Keys

The Relational Model: Normal Forms (Good Practice)

Normal forms are defined by E. F. Codd (1971) as:

1. Each record should be 'atomic' that is, non-divisible. A single row/record, should not contain multiple, divisible pieces.
 - ▶ The respondent table should not have a cell with the ages of all household members (stored in a 'roster' file).
2. No non-prime attribute of the table may be dependent on a proper subset of any candidate key.
 - ▶ In the time use survey, a respondent ID and activity number jointly identify an activity. The table should not, therefore, contain information on the respondent.
3. The values in a row should refer uniquely to the key – not to a non-key attribute.
 - ▶ In the time use activity tables, we should not store both the activity code and its interpretation.

To avoid repeating cumbersome calculations, these are sometimes violated.

RDB: Time Use Survey, Revisited

Respondents				
ID	Line	Children	Worked	...
1	1	0	1	...
2	1	1	0	...
3	1	2	1	...
4	1	0	1	...
5	1	3	1	...
6	1	0	0	...
7	1	1	1	...
...

Activities				
Resp.	Activity #	Code	Duration	...
1	1	010101	150	...
1	2	130124	45	...
1	3	010201	30	...
1	4	020201	10	...
1	5	110101	15	...
1	6	180501	20	...
1
2	1	010101	240	...
...

CPS				
Resp.	Line	Age	Education	...
1	1	38	38	...
1	2	35	0	...
1	3	4	1	...
2	1	53	1	...
2	2	58	0	...
...

* Primary Keys

Good Practice, in Practice

- ▶ Each table should contain a single logical element, without repetition.
- ▶ One MUST take some care to understand what is unique in your table, and use that property to link tables: the primary key.
- ▶ Appropriate primary keys are what make databases actually work efficiently.

Using a Database

Relational Database Management Systems (RDBMSs) and the Structured Query Language (SQL)

- ▶ Most of the most-prevalent database systems implement the relational model.* These systems are called RDBMSs.
- ▶ Structured Query Language (SQL) is a standardized (ANSI/ISO 9075) language for interacting with RDBMSs.
- ▶ Originally intended to be user-facing, so 'fairly intuitive.'
- ▶ Despite standardization, the implementations almost all have some (extremely annoying) differences: some tinkering is necessary to migrate between 'dialects.'
 - ▶ Nevertheless, the basic functionality – selecting, inserting, deleting, and altering data, is pretty consistent.

*There are always exceptions...

(Some of) The Most Prevalent Databases

ORACLE®



(Some of) The Most Prevalent Databases

ORACLE®



(Some of) The Most Prevalent Databases

ORACLE®



MariaDB



**This
class.**
SQLite



Microsoft®
SQL Server™



PostgreSQL



mongoDB®

Opening SQLite

- ▶ To just open a file, do:

```
■ sqlite3 atus.sqlite
```

- ▶ This should give you a new prompt.

Navigating SQLite: Time Use Survey

The biggest differences between RDBMS implementations are accessing metadata: table names and schema (columns).

- ▶ To show the tables in the database:

```
sqlite> .tables  
sqlite> .fullschema --indent
```

- ▶ To show the 'schema' of a table (its columns and types):

```
sqlite> .schema cps
```

- ▶ Types in SQLite: integer, real (float), text (string), or null.
 - ▶ Other RDBMSs have more types – e.g., datetime, or even geographies.
 - ▶ SQLite just has date functions.

This is THE basic SQL syntax that you will use.

- ▶ Selecting all (*) columns from the cps table:

```
SELECT * FROM cps;
```

- ▶ You can also name specific columns:

```
SELECT marital_status, years_education FROM cps;
```

- ▶ Each query ends by a semi-colon.
- ▶ Upper case keywords an old convention: SQL strings are often used in other languages, and therefore aren't color-highlighted. Not necessary.
- ▶ There is absolutely no requirement about the formatting of the query.

Running Queries in SQLite

To make the output clearer, you can do:[†]

- `.mode columns`
- `.headers on`

You can also run queries in a file, like so:

- `sqlite3 atus.sqlite < ex/limit_cps.sql`

We'll talk about interfacing to pandas, on Wednesday

[†]You can also put these two lines in your `~/.sqliterc` file, that will run each time you open `sqlite3`.

- ▶ To reduce output, for exploration, use 'LIMIT':

```
SELECT * FROM cps LIMIT 10;
```

- Make requirements with 'WHERE':

```
SELECT years_education
FROM cps
WHERE years_education > 0; /* i.e., exists */
```

- Can make multiple requirements with AND or OR:

```
SELECT years_education, state_code
FROM cps
WHERE years_education > 0 AND
      state_code = 17; — note the single '=' sign!
```

- Comment to end of line by --, or between two markers /* to */.

- ▶ This functions exactly as groupby() in pandas.
- ▶ 'Group' by one or more variables, to aggregate over others.
 - ▶ Unlike most RDBMSs, SQLite won't complain if you mix and match aggregating functions and non-aggregated fields – so be careful.
- ▶ Many functions: AVG, SUM, MAX, MIN, COUNT, etc.

SELECT

```
number_of_hh_children,  
AVG(daily_time_alone_non_work/60)
```

FROM

```
respondents
```

GROUP BY

```
number_of_hh_children
```

```
;
```

- Sort by one or more fields, ascending or descending (ASC or DESC).

SELECT

state_code,

AVG(educational_attainment > 42) **AS** Bachelors

FROM cps

WHERE

educational_attainment > 0 /* i.e., defined */

GROUP BY state_code

ORDER BY Bachelors **DESC**

LIMIT 10;

- Note also the use of AS (like in python!), to alias a long column name.

- ▶ In households with children, what is the likelihood that a spouse or partner is present, by levels of education. Must JOIN tables.
- ▶ Alternatively, this can be done with multiple tables in 'FROM' and join conditions under 'WHERE.'

SELECT

```
educational_attainment,  
AVG(spouse_or_partner_present == 1) Married,  
COUNT(spouse_or_partner_present == 1) "(N)"
```

FROM cps

JOIN respondents ON

```
cps.case_id = respondents.case_id AND  
cps.line_no = 1
```

WHERE

```
number_of_hh_children > 0
```

GROUP BY educational_attainment;

- ▶ In households with children, what is the likelihood that a spouse or partner is present, by levels of education. Must JOIN tables.
- ▶ Alternatively, this can be done with multiple tables in 'FROM' and join conditions under 'WHERE.'

SELECT

```
educational_attainment,  
AVG(spouse_or_partner_present == 1) Married,  
COUNT(spouse_or_partner_present == 1) "(N)"
```

```
FROM cps,  
      respondents
```

WHERE

```
cps.case_id = respondents.case_id AND  
cps.line_no = 1 AND  
number_of_hh_children > 0
```

```
GROUP BY educational_attainment;
```

- ▶ HAVING is like WHERE for grouped values.
- ▶ For instance, you could present averages only for groups with COUNT() larger than 30, or averages or sums above a threshold.

SELECT

```
number_of_hh_children Children,  
AVG(daily_time_alone_non_work/60)  
  AS "Alone Time",  
COUNT(number_of_hh_children) N  
FROM respondents  
GROUP BY number_of_hh_children  
HAVING N > 30;
```

- ▶ You can use subqueries as tables, for multiple levels of grouping.
- ▶ How much time does each sex claim to spend in 'Personal/Private activities' such as 'necking' and 'private activity, unspecified'?

```
SELECT sex, COUNT(sex), AVG(time) FROM (  
    SELECT roster.case_id, AVG(edited_sex) sex,  
           SUM((activity_code = 10401) * (duration))  
             AS time  
    FROM roster  
    INNER JOIN activities ON  
        roster.case_id = activities.case_id AND  
        roster.line_no = 1  
    WHERE 18 < edited_age AND edited_age < 30  
    GROUP BY roster.case_id  
) GROUP BY sex;
```

On The Structure

- ▶ Good news is: SQL queries basically always follow the same structure.
- ▶ You may or may not need all the pieces, but there's no question about the order– there's only one way.
- ▶ There are a few more keywords, but the format of the query on the last page is as complicated as `SELECT` statements get.

Creating Tables

CREATE TABLE

- ▶ Most likely, you will be a database user rather than developer, but it's useful to understand how to create a simple database.

```
■ sqlite3 my_new_db.sqlite
```

```
CREATE TABLE test (id INTEGER PRIMARY KEY NOT NULL,  
                    Name TEXT, Birthday TEXT,  
                    Fruit TEXT);
```

- ▶ This defines the schema of a table.
 - ▶ SQLite recognizes: INTEGER, REAL, TEXT, BLOB, and NULL.
- ▶ The id column will auto-increment, to provide a unique primary key.
- ▶ NOT NULL and PRIMARY KEY are constraints, that help the RDBMS optimally parse a statement.

- Insert values like so:

```
INSERT INTO test (Name, Birthday, Fruit)
VALUES ("Annie Oakley", "1860-08-13", "Peaches");
```

```
INSERT INTO test (Name, Birthday, Fruit)
VALUES ("Freddie Mercury", "1946-09-05", "Plums");
```

```
INSERT INTO test (Name, Birthday, Fruit)
VALUES ("Lyndon Johnson", "1908-08-27", "Cumquat"),
       ("Jason Bourne", "1970-09-13", "Starfruit"),
       ("Jesus of Nazareth", "0000-12-25", "Grapes");
```

- Multiple values are both convenient and efficient.

When the RDBMS evaluates a statement, it begins by 'parsing' and compiling it: reading it and turning it into a computer-ready, optimized plan. Most RDBMSs treat this as separate from the execution step, so you can 'recycle' the execution plan.

Creating a Table From the Command Line

- ▶ There's a shortcut in `sqlite3` for loading data from a csv file.
 - ▶ All RDBMSs provide some analogous functionality for loading data.

```
■ CREATE TABLE chicago (Last TEXT, First TEXT,  
Position TEXT, Department TEXT, FP TEXT, SH TEXT,  
Hours INTEGER, Salary REAL, Wage REAL);  
■ .mode csv  
■ .import salaries.csv chicago  
■ .mode columns
```

- ▶ Let's ask some simple questions:
 - ▶ Best-paid salaried workers in the city?
 - ▶ Departments with the highest average salary?
 - ▶ Departments with the highest fraction non-salaried workers?
- ▶ You'll try this, for homework.

Access from Python

Using the sqlite3 Library

- ▶ Straightforward python interface: `sqlite3` ([docs](#)).
- ▶ And pandas totally builds in support, like `read_csv/read_json`.

```
import sqlite3

con = sqlite3.connect("atus.sqlite")
result = con.execute("""
    SELECT case_id, line_no, family_income
    FROM cps WHERE line_no = 1
    LIMIT 10""")

for row in result: print(row)
```

Using the sqlite3 Library

- ▶ Straightforward python interface: `sqlite3` ([docs](#)).
- ▶ And pandas totally builds in support, like `read_csv/read_json`.

```
import sqlite3, pandas as pd

con = sqlite3.connect("atus.sqlite")
cps = pd.read_sql_query("""
    SELECT case_id, line_no, family_income
    FROM cps WHERE line_no = 1
    LIMIT 10""",
    con, index_col = "case_id")

print(cps)
```

Broader Example: Child Engagement

- ▶ Want direct child engagement by parent education.
 - ▶ Will sum over activities, per person.
 - ▶ Use `years_education`, `activity_code`, and `duration`; join on `case_id` and `line_no`.
 - ▶ Codes are codes 301XX, 302XX, 303XX.
 - ▶ Require children in household and respondent to work (`edited_labor_force_status < 3`).
-
1. Write a SQL query to give us the formatted data.
 2. Use this query load the data in pandas, and plot it.

SELECT

```
respondents.case_id,  
cps.years_education AS 'Education',  
SUM((activity_code/100 IN (301, 302, 303))  
    * duration/60.) AS 'Engagement'
```

```
FROM respondents
```

```
INNER JOIN cps ON
```

```
    respondents.case_id = cps.case_id AND  
    respondents.line_no = cps.line_no
```

```
INNER JOIN activities ON
```

```
    respondents.case_id = activities.case_id
```

```
WHERE
```

```
    number_of_hh_children > 0 AND  
    edited_labor_force_status < 3
```

```
GROUP BY respondents.case_id;
```

```
import sqlite3, pandas as pd
from matplotlib import pyplot as plt

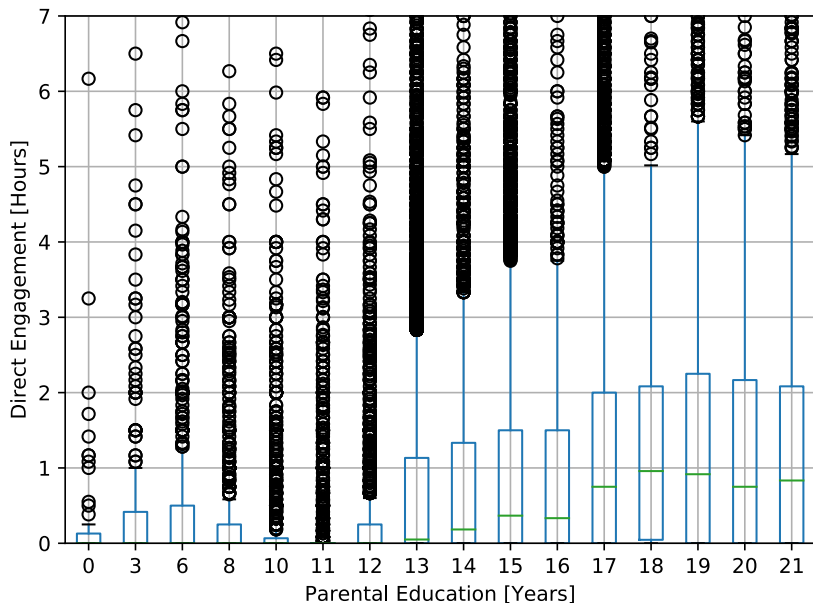
con = sqlite3.connect("atus.sqlite")

with open("ex/direct_engagement.sql") as f:
    query = f.read() # entire file into string.

df = pd.read_sql_query(query, con)
ax = df.boxplot("Engagement", "Education")

plt.suptitle("")
ax.set(title = "", ylim = (0, 7),
        xlabel = "Parental Education [Years]",
        ylabel = "Direct Engagement [Hours]")
ax.figure.savefig("engagement.pdf")
```


Engagement: Results



Housework: Python

```
import sqlite3, pandas as pd

con = sqlite3.connect("atus.sqlite")

with open("ex/housework.sql") as f:
    df = pd.read_sql(f.read(), con)

ax = df[df.sex == 2].plot(x = "ed", y = "housework",
                        label = "Women")
df[df.sex == 1].plot(x = "ed", y = "housework",
                    label = "Men", ax = ax)

ax.set_ylabel("Average Daily Housework [Hours]")
ax.set_xlabel("Education [Years]")

ax.figure.savefig("housework.pdf")
```

Housework: Results

