Week 10: Relational Databases and SQL

LSE MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2024

Ryan Hübert

Slides last updated: 4 December 2024

Outline

- ➜ Relational vs non-relational databases
- ➜ Structured Query Language (SQL)
- ➜ Coding session

Relational vs non-relational databases

Databases

Database system: an organized collection of data that is stored and accessed via a computer

- The way a database is organized is a **schema**
- Since a database is used for data *storage*, a user typically "reads" and "writes" to a database
- Access data via **queries**
- Queries are often constructed/written in **domain-specific languages** like SQL, but not always
- A user can typically read and write via `R` (or `python`)

Relational vs non-relational databases

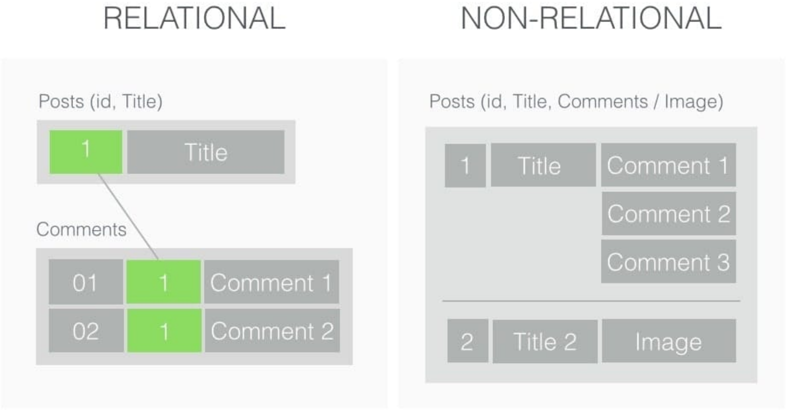
Relational databases

- data is stored in multiple tables to avoid redundancy
- tables are linked based on common **keys**
- SQL is dominant DSL used to access data

Non-relational databases

- data stored in a way that is not based on tabular relations (e.g. MongoDB uses JSON like documents)
- Data is accessed using a wide variety of (sometimes customised) languages

Relational vs non-relational databases



From: Codewave Insights

Relational databases

Relational Database Management Systems (RDBMS):

- ➜ Underlying software system used to maintain relational databases
- ➜ E.g.: MySQL, PostgreSQL, SQLite, MariaDB, etc.

Online Transaction Processing (OLTP) Services:

- ➜ High frequency (many transactions per minute), fast response, many write operations
- ➜ E.g.: Amazon RDS, Google Cloud SQL, Azure SQL Database

Online Analytical Processing (OLAP) Services:

- ➜ Large volume (petabytes of data), lower frequency (few transactions), slower response, mostly read operations
- ➜ E.g.: Amazon RedShift, Google BigQuery, Microsoft Azure SQL Server, Snowflake

Some vocabulary

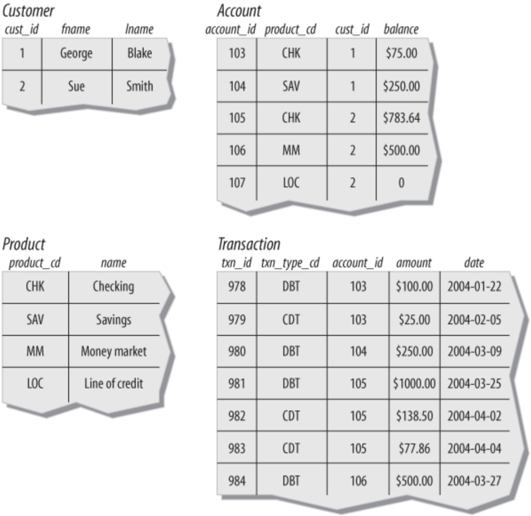
Relational database term	SQL term
Relation	Table
Tuple, record	Row
Attribute, field	Column

(Excerpt from: https://en.wikipedia.org/wiki/Relational_database)

Keys

- Keys are *critical*, allowing the rows of different tables to be connected
- Primary key: A column or set of columns (composite key) which uniquely identifies each row/record in the table
- Foreign key: A primary key of another table

Relational databases in action



Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

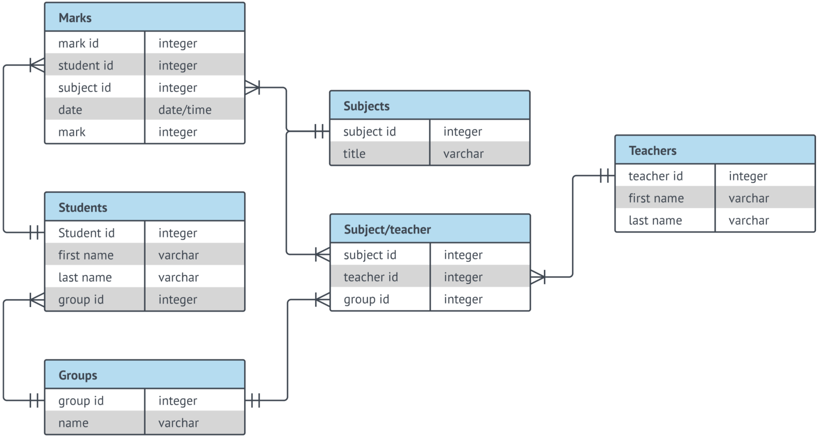
product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

Entity relationship diagrams (ERDs)

A database's **schema** can be represented with an ERD



Marks	
mark id	integer
student id	integer
subject id	integer
date	date/time
mark	integer

Subjects	
subject id	integer
title	varchar

Teachers	
teacher id	integer
first name	varchar
last name	varchar

Students	
Student id	integer
first name	varchar
last name	varchar
group id	integer

Subject/teacher	
subject id	integer
teacher id	integer
group id	integer

Groups	
group id	integer
name	varchar

Source: Lucidchart

Structured Query Language

SQL: Structured Query Language

- A “domain specific language” (DSL) designed to define, control access to, manipulate, and query relational databases
- Initially written SEQUEL (Structured English Query Language), but later changed to SQL because of trademark issues
- Pronounced both S-Q-L and SEQUEL today
- It is a **nonprocedural/declarative language**: User defines what to do, inputs, and outputs, but not the control flow
  - How the statement is executed is left to the *optimizer*, which is opaque to the user
- How long SQL queries depends on optimization
- Performance will vary, but generally faster than standard data frame manipulation in R (and much more scalable)

Some common components of SQL queries

- The result of a SQL query is a table
- **SELECT** columns
- **FROM** a table in a database
- **WHERE** rows meet a condition
- **GROUP BY** values of a column
- **ORDER BY** values of a column when displaying results
- **LIMIT** to only X number of rows in resulting table
- Always required: **SELECT** and **FROM**; rest are optional
- **SELECT** can be combined with operators such as **SUM**, **COUNT**, **AVG**. . .

Some more components of SQL queries

- To merge multiple tables, use **JOIN**
  - Variety of ______ **JOIN** types: **INNER**, **RIGHT**, **LEFT FULL OUTER**
  - For anti-joins, use **RIGHT** or **LEFT** and a **WHERE** clause
  - When handling multiple tables, use aliases (e.g. **FROM table AS t**)
- More complex ways of combining tables include (non-exhaustive):
  - **CROSS JOIN**: Produce all combinations of the two ids
  - **UNION**: De-duped vertical combination of both tables (add **ALL** for dupes)
- SQL also supports common table expressions (CTEs):
  - Lets you build multiple sub-tables within a single query
  - Connect these together with a subsequent **SELECT** statement

SQL and `tidyverse`

SQL is just way to do data manipulations on tabular data

You already know how to work with and manipulate tabular data using `tidyverse`, which is *conceptually* identical

Many SQL queries “resemble” `tidyverse` functions, e.g.:

- In SQL, you `SELECT` columns; in `tidyverse` you `select()` columns
- In SQL, you use `WHERE` to subset rows using a condition; in `tidyverse` you `filter()` rows according to a condition
- In SQL, you `LEFT JOIN` two tables; in `tidyverse` you `left_join()` two tibbles
- Etc.

SQL query examples

Table 1 named client

```
##      id  name      gender billed account_id
## [1,] "1" "Alice"   "F"    "500"  "101"
## [2,] "2" "Bob"     "M"    "750"  "102"
## [3,] "3" "Charlie" "F"    "200"  "103"
```

Table 2 named account

```
##      id    balance
## [1,] "101" "5000"
## [2,] "102" "3000"
## [3,] "103" "7000"
```

SQL query examples

This returns a table with the name and account_id columns of client:

```
   SELECT name, account_id FROM client;
```

The tidyverse equivalent:

```
client %>%
  select(name, account_id)
```

Returns:

```
##      name      account_id
## [1,] "Alice"   "101"
## [2,] "Bob"     "102"
## [3,] "Charlie" "103"
```

SQL query examples

This returns a table with all columns of `client` but only rows where the `gender` variable is "F":

```
    SELECT * FROM client WHERE gender = 'F';
```

The `tidyverse` equivalent:

```
client %>%
  filter(gender == "F")
```

Returns:

```
##      id  name      gender billed account_id
## [1,] "1" "Alice"   "F"    "500"  "101"
## [2,] "3" "Charlie" "F"    "200"  "103"
```

SQL query examples

This returns a table with two columns, total_billed and avg_billed and one row giving the total billed and average billed amounts for female clients in client table:

```
   SELECT SUM(billed) AS total_billed,
          AVG(billed) AS avg_billed
   FROM client
   WHERE gender = 'F';
```

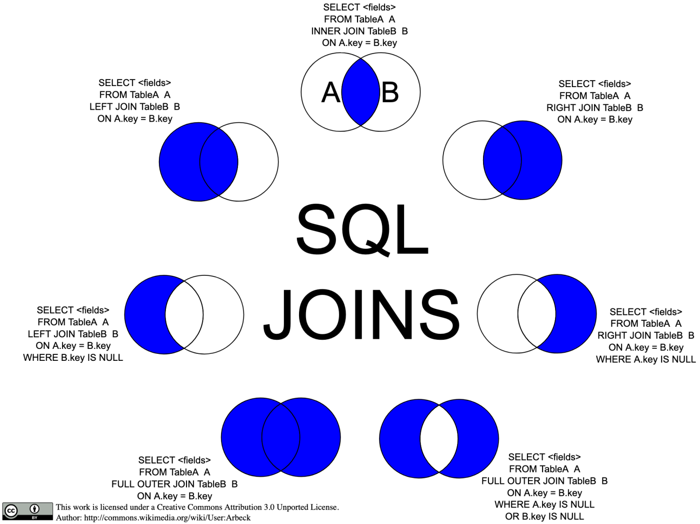
The tidyverse equivalent:

```
client %>%
  filter(gender == "F") %>%
  summarise(total_billed = sum(billed),
            avg_billed = mean(billed))
```

Returns:

```
##      total_billed avg_billed
## [1,]          700        350
```

SQL JOINs



```
SELECT <fields>
FROM TableA  A
INNER JOIN TableB  B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA  A
LEFT JOIN TableB  B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA  A
RIGHT JOIN TableB  B
ON A.key = B.key
```

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

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

```
SELECT <fields>
FROM TableA  A
FULL OUTER JOIN TableB  B
ON A.key = B.key
```

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

From: https://upload.wikimedia.org/wikipedia/commons/9/9d/SQL_Joins.svg

SQL JOIN examples

This returns a table with two columns name and balance created by inner joining tables client and account by their shared keys, account_id and id:

```
   SELECT client.name, account.balance
   FROM client JOIN account
   ON client.account_id = account.id;
```

The tidyverse equivalent:

```
client %>%
  inner_join(account,
             by = c("account_id" = "id")) %>%
  select(name, balance)
```

Returns:

```
##      name      balance
## [1,] "Alice"   "5000"
## [2,] "Bob"     "3000"
## [3,] "Charlie" "7000"
```

Coding session

Coding session

Download from moodle:

- ➜ `public Facebook data (individual csv files)`

Code:

- ➜ `01-sql-intro.Rmd`
- ➜ `02-sql-join-and-aggregation.Rmd`

General information on how to connect to SQL databases with R: https://solutions.rstudio.com/db/