Relational Databases

Why using databases?

- A place to **store** data

- Provides **structure** to the data

- Provides a mechanism for querying, creating, modifying and deleting data.

Relational Database Management Systems (RDBMS)

If you want to create and use relational databases, you need a piece of software. This software is the RDBMS.

MySQL, PostgreSQL, MariaDB, Oracle, IBM DB2… are all examples of RDBMS.

They all share the main principles, and the language used to query the data is almost (but not completely) identical.

MySQL is the most widely used one, and it’s open source.

SQL: Structured Query Language

Easy peasy language to interact with databases: select & filter information, join tables, create tables, insert/delete data…

To get started, what's difficult is to set up the tools, create 'connections' to and from other tools, understand weird words like "schema", "client", "host" or "DBMS".

Queries get difficult when the structure of the data itself gets complex, answers to business questions are not obvious, the source of the data is dubious...

Most of the time, you need an intermediate level of SQL to get a job in the data world. However, you just need the knowledge to get the right data, not to set up & manage a database.

Relational databases save space & ensure data consistency

NON - RELATIONAL

Employees_stores

emp_ID	First_name	Last_name	Store_id	Store_city	Store_type	Store_emp
1	Vladimir	Popov	A	London	Showcase	15
2	Cinar	Horton	B	Norwich	Regular	3
3	Harry	Beltik	A	London	Showcase	15

RELATIONAL

Employees_stores

emp_id	store_id
1	A
2	B
3	A

Employees

emp_id	First_name	Last_name
1	Vladimir	Popov
2	Cinar	Horton
3	Harry	Beltik

Stores

Store_id	Store_city	Store_type	Store_emp
A	London	Showcase	15
B	Norwich	Regular	3
C	Bristol	Regular	0

If an employee changes store, we just update this.

- Only 1 row per store.
- Easy to maintain and update.
- We can have stores without employees.

In relational databases, **data is scattered across multiple tables**. Good to avoid data redundancy, but not the easiest display for analysing and reporting.

The solution to this challenge is to create a **data mart**.

A data mart is a subset of the database with the information relevant for a specific department, a reporting task, a dashboard, etc.

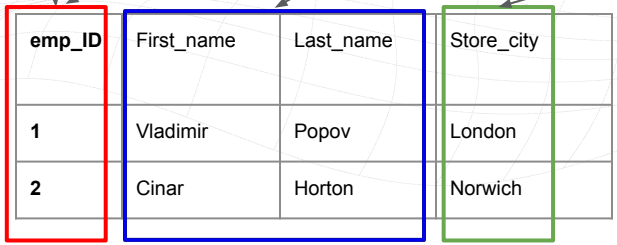
A crucial part of creating a data mart is to **join** tables.

Creating a data mart (from relational to non-relational)

emp_id	store_id
1	A
2	B
3	A

emp_id	First_name	Last_name
1	Vladimir	Popov
2	Cinar	Horton
3	Harry	Beltik

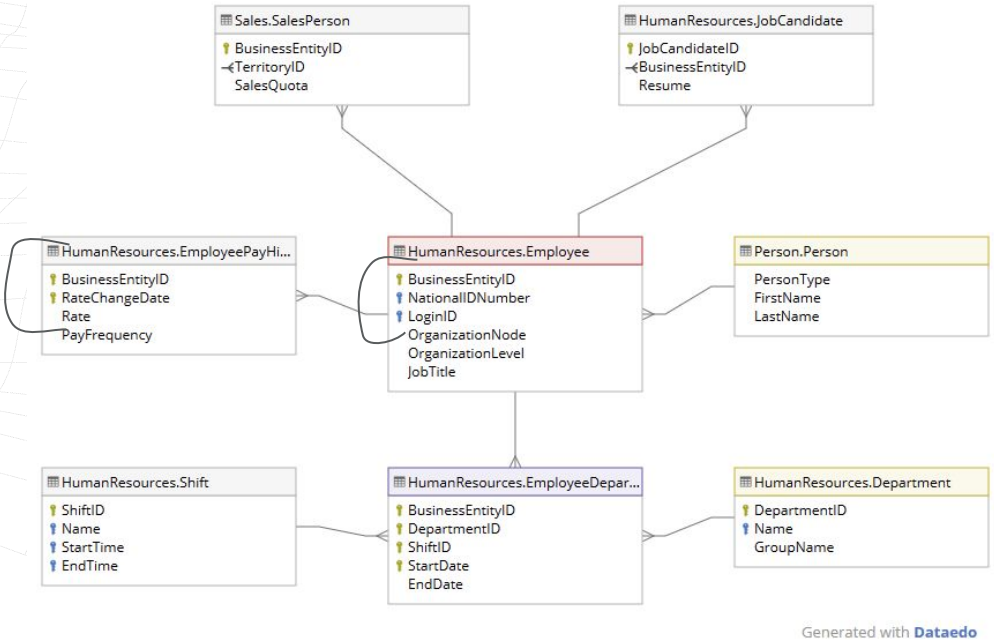
Store_id	Store_city	Store_type	Store_emp
A	London	Showcase	15
B	Norwich	Regular	3
C	Bristol	Regular	0



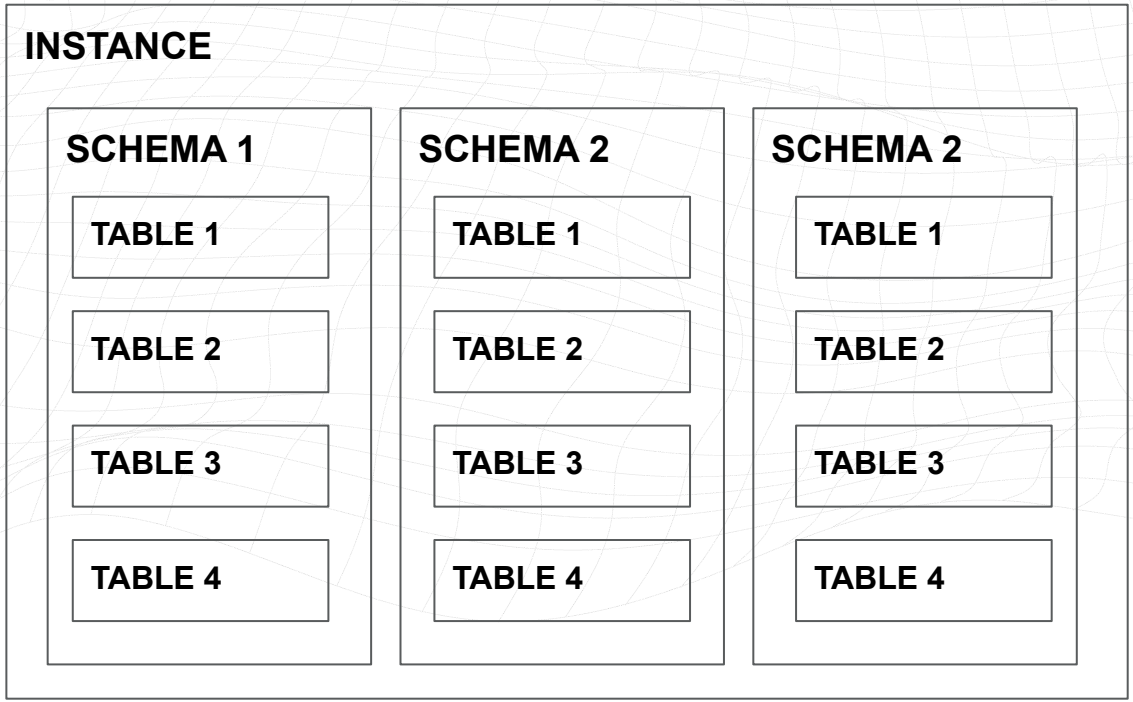
emp_ID	First_name	Last_name	Store_city
1	Vladimir	Popov	London
2	Cinar	Horton	Norwich

Data model

Also referred to as “Entity-Relationship Diagram”, “schema”...



Instances can have multiple schemas*



In MySQL:

Instance = Connection
Schema = Database

*Different DBMS have slightly different terminologies/structures

SQL basic query functions

USE. Defines the schema to use in the session.

```
USE my_schema;
```

SELECT. Used to retrieve data.

```
SELECT colum1, column2 FROM table;
SELECT * FROM table;
```

WHERE. Sets a condition for your select clause

```
SELECT * FROM table WHERE column1 > 10;
```

DISTINCT. Only select unique results

```
SELECT DISTINCT(column_1) FROM table;
```

LIMIT. Restricts the rows returned in the output (useful to test queries in big tables):

```
SELECT * FROM table LIMIT 10;
```

ORDER BY. Sorts results

```
SELECT * FROM table ORDER BY column1;
```

GROUP BY. Groups the data (needs an aggregation function):

```
SELECT colum1, AVG(column2)
FROM table
GROUP BY column 1;
```

HAVING. Defines a condition for your select clause.

```
SELECT colum1, AVG(column2)
FROM table
GROUP BY column 1
HAVING  AVG(column2) >10;
```

COUNT, **SUM**, **AVG**, **MAX**, **MIN**...

https://www.sqltutorial.org/sql-cheat-sheet/

SQL Basic Components

```sql
SELECT
  name,
  gender,
  SUM(number) AS total
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
WHERE
    year BETWEEN 2000 AND 2003 OR
    (year>=2000 AND year<=2003) OR
    year IN (2000,2001,2002,2003)
GROUP BY
  1,2
HAVING
    SUM(number)>4000000
ORDER BY
  total DESC
LIMIT 10
```

SELECT existing fields or create derived fields (aggregated or not)

FROM which table to query

WHERE to filter for specified conditions

GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

HAVING keyword specifies a search condition for an aggregate or a group

ORDER BY specified variables, if not specified by default it is ASC

LIMIT rows to display

MySQL data types

When creating a table, you'll need to define the data types of each column. The most common ones are:

- **INT**(size) - Integer with different sizes
- **FLOAT**(size,d) - decimal numbers, with increasing 'accuracy' & 'size'
- **VARCHAR**(size) - A string with variable length
- **DATETIME** - Date & time in CCYY-MM-DD hh:mm:ss format. There are other options in case you just need the time, the year, etc.

There are many more data types, designed to optimize the space & fit every need, but you will mostly use these ones.
Full docs here: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

From the October 2012 Issue

```
SELECT * FROM t1;
```