DataBase Management Systems (DBMS)

Dr Peadar Grant

September 17, 2024

Contents

1 Database management systems (DBMS)

DataBase Management Systems (DBMS) persistently store data records according to a schema.

Key concepts

Schema defines *how* data is stored. Defines rules.
Records of logically grouped data (e.g. a customer, an invoice)

1.1 Required functionality

Must provide

Data storage/retrieval: fundamental reason for using!
Catalog of schema and data stored
Transaction support for batch updates to be applied atomically
Concurrency control to manage simultaneously connected clients
Recovery to a working state after hardware / software failures
Authorization services to verify connected clients identity and control access.
Data communication support to allow usage from remote computers.
Integrity services to enforce constraints on data as defined by the user.

1.2 Families

Relational databases

- Sometimes called **SQL databases**.
- Examples: Oracle, PostgreSQL, MySQL, Microsoft SQL.

Non-relational databases

Key-Value stores that associate a key with a value. (e.g. Redis)

- Optimised for fast lookup of small pieces of data

Document databases that store a semi-structured document identified by a key (e.g. MongoDB)

- Useful for semi-structured data accessed as a whole

Graph databases that store nodes and their relationships (e.g. Neo4J)

- Mapping, transport planning, relationships, likes, friends.

We'll begin on **relational databases** using **PostgreSQL**.

1.3 Text-based query language

Most database systems incorporate a native programming language allowing us either directly or via a client application manipulate the database:

3 roles:

Data Definition (DDL) to work with schema.
Data Manipulation (DML) to work with data stored according to the schema.
Query language for accessing data

Usually a common language provides all of these functions. SQL best known:

```
SELECT name, finish_time
FROM competitors
ORDER BY finish_time DESC;
```

1.4 Database host

The server process manages the data store and processes requests from clients. The server can be running on any of the following *hosts*:

Hosts

- Standard laptop / desktop computer
- Dedicated server computer (in a data centre environment)
- Cloud-based virtual host, called a compute instance. (e.g. Amazon EC2)
- A managed database service provided by a cloud service provider (e.g. Amazon RDS, Azure, Google Cloud, IBM Cloud)

Database is often a critical piece of our infrastructure and needs to be provisioned appropriately. *Will discuss data centre environment later on!*

DBMS can run on a variety of Operating Systems (often UNIX / Linux):

- We will learn how to use PostgreSQL within a Unix (Linux) environment.

1.5 Client-server

Most database management systems run in a client-server model, even on the same host.

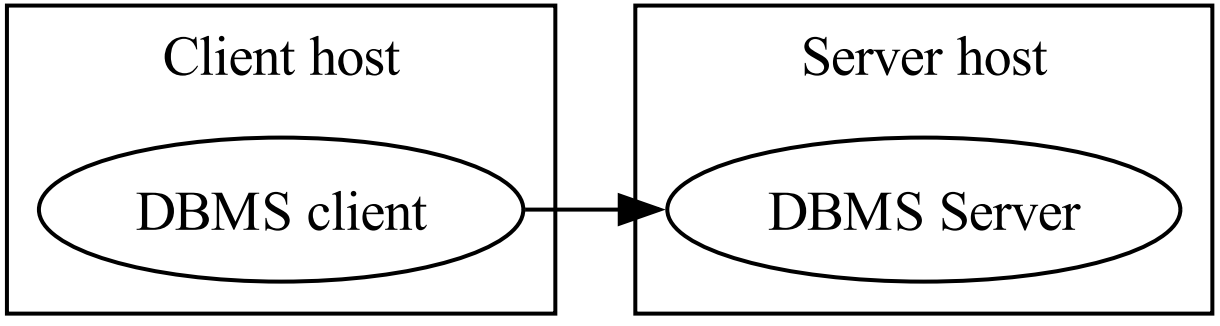

Figure 1: Client-Server model

1.6 Protocol

The client program accesses the server using a server-specific protocol.

Clients normally access the server through IP networks using TCP on a specified port number.

Examples of clients

- Most databases have a simple command-line client that can send requests to the database and display results
- Apps can be written to access database servers using a client library.
  - Generally the text-mode client uses this library internally too!

Important note about clients

- The client may in some cases be running on the same host as the server.
- Software that is the client of a DBMS may itself be a server of another type.

1.7 Concurrency

This also implies that there is a degree of concurrency, where multiple clients access the same database at the same time.

Clients are often heterogeneous, where different types of clients concurrently access the same data.

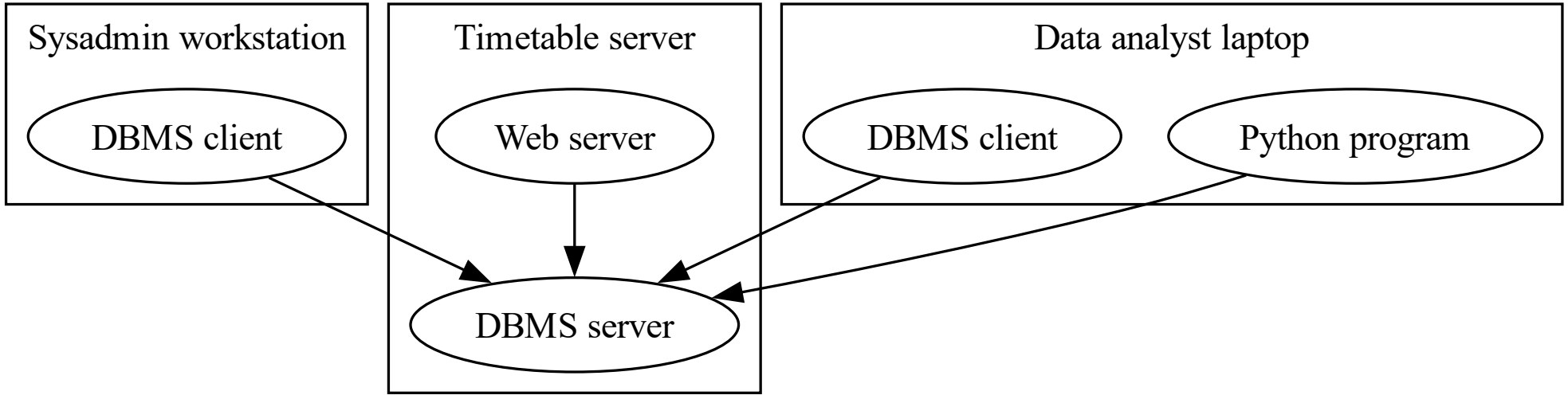

Figure 2: Concurrent access to a college timetable database

1.8 Remote access

A particular pattern you will encounter is where the client program runs on the same host as the DBMS, and remote shell access is used to permit clients to connect to the server.

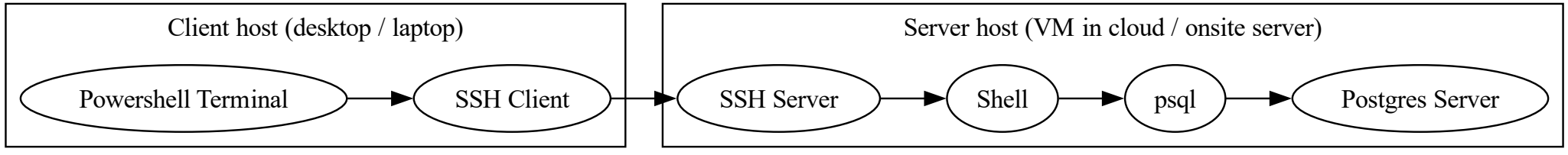

Figure 3: SSH access to a remote database

2 PostgreSQL

We will focus on PostgreSQL as our primary database.

Reasons for choice of PostgreSQL:

- It is free software
- Can be installed on many operating systems.
- No limits / free-trial limitations.
- Support exists for geospatial data, JSON, XML, full-text search etc.
- Has some support for masquerading as other types of DB (e.g. graph data).
- High adoption among many analytics and data science workloads.

As we continue we will refer to PostgreSQL as **Postgres** for brevity.

On a single host, a relational database management system **cluster** provides one or more isolated **databases**. Within a database, data is contained in one or more **tables** according to the **schema**.

3 Defining characteristics

Codd's Seminal paper defines a set of characteristics that a relational database must possess.

We'll use these to explore the basic anatomy of a relational database, specifically PostgreSQL.

Note: the order of these rules has been adjusted to introduce some concepts in a more logical way!

3.1 Foundation rule

For any system that is advertised as, or claimed to be, a relational database management system, that system must be able to manage databases entirely through its relational capabilities.

3.2 Information Rule

All information in a relational database is represented explicitly at the logical level and in exactly one way — by values in tables.

3.2.1 Relational data structure

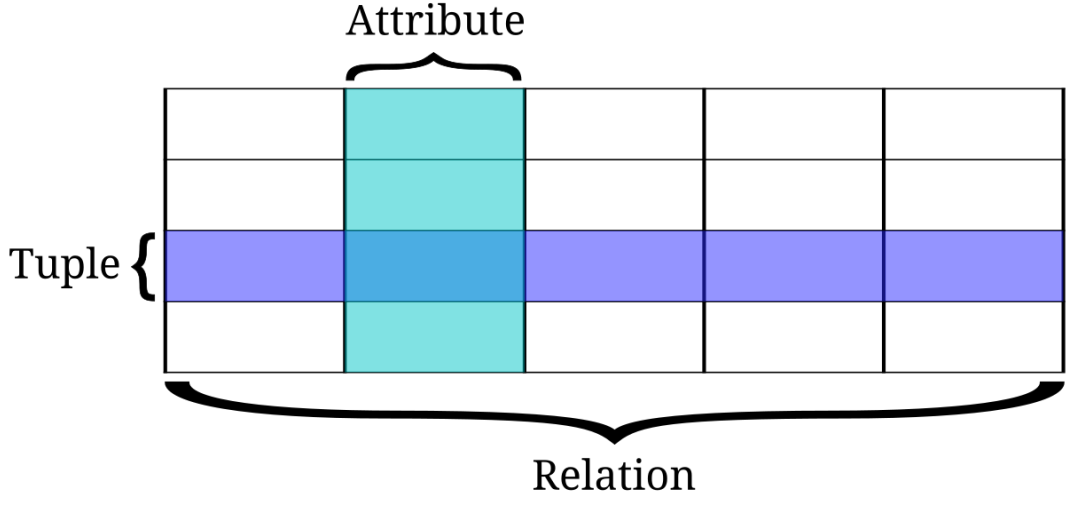

Figure 4: Relational model terminology

Relation (Table): with columns and rows. Table names must be unique.

Base relation is a table defined in the database schema

Derived relation is a virtual table that appears in response to a query

Attribute (Column): holding distinct part of a row:

- Column names must be unique within a table.
- Column has a **data type** (e.g. integer, text) from an allowable list (database-dependent).
- Column may allow / permit **null** values (unknown, empty, undefined, blank)
- Column order has no meaning or significance.

Tuple (Row): a single record contained with a table.

- No theoretical upper limit on number of rows.
- The natural order of rows in a table is meaningless! Do not rely on it!

3.3 Systematic treatment of null values

Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

3.3.1 Null

Null denotes a lack of a value, used to indicate that data is missing, unknown, blank, undefined:

- Null is **not** zero, the empty string, false or any other value.
- Null values will **pollute other expressions** (e.g. arithmetic, comparison)
  - Null does not even equal null!
- **Can test** for null with `IS NULL` operation.

While the concept of null can be confusing, it avoids placeholder data.

3.4 Integrity independence

Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs

3.4.1 Unique constraint

Within a table, a unique constraint on a column or group of columns will prevent duplicate rows existing.

3.4.2 Primary key

Practically we need ways to identify any row individually. Within any table, there may be a number of **candidate keys** that could be used to identify each row.

Simple key: a single column that is:

not null so that every row can be identified.

unique so that every row is distinct.

Complex key: two or more columns:

1. no columns in the key are null
2. together are unique

For each table, one of its candidate keys is selected as the **primary key**.

Should always be encoded explicitly by DML in the database.

3.5 Guaranteed access rule

Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

Often in practice we are operating on groups of rows where we select them based on the values of one or more columns.

3.6 Comprehensive data sublanguage rule

A relational system may support several languages ... However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

1. Data definition.
2. View definition.
3. Data manipulation (interactive and by program).
4. Integrity constraints.
5. Authorization.
6. Transaction boundaries (begin, commit and rollback). *We will meet transactions again later on.*

In PostgreSQL's case, its adoption of SQL fits this rule.

3.7 Dynamic online catalog based on the relational model

The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

In practical terms

The DBMS can provide the schema for any table using the same methods we'd use to query data.

- Self-documenting
- Output can be used for generating reports, diagrams etc.

3.8 View updating rule

All views that are theoretically updatable are also updatable by the system.

Practical meaning

- Relational databases can have views made up of data dynamically drawn on demand from different tables, but should appear as a single table.
- In theory these should be updateable but in practice we have to tell the DB how to do it.
- *We'll come back to this one later when you're familiar with views.*

3.9 Relational Operations Rule / Possible for high-level insert, update, and delete

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

3.10 Physical data independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

Practical meaning

- Changes made to the database software in how it internally stores data shouldn't be visible or require any changes at the client level.

3.11 Logical data independence

Application programs and terminal activities remain logically unimpaired when information-preserv
changes of any kind that theoretically permit unimpairment are made to the base tables.

Practical meaning

- Changes made in the database structure shouldn't affect programs accessing it.
- Difficult to practically achieve in practice!

3.12 Distribution independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

Practical meaning

- We shouldn't be concerned that the DBMS is storing data on multiple file(s) on disk(s).
- More complex DBMS installations may be distributing or *sharding* data over different nodes for space scalability.

3.13 Non-subversion rule

If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).

Practical meaning

There can’t be a “bypass” switch on the integrity constraints!

3.14 Foreign keys

The foreign key is one of the most powerful mechanisms in relational databases. It allows us to explicitly encode the idea of a table being linked to another.

A foreign key defined on a column points to a column (usually the primary key) of another table.

The RDBMS enforces the constraint that a value in the foreign key column must exist in the referenced table. We will meet foreign keys again when we set up multi-table databases.