Relational database management systems review

Dr Peadar Grant

January 25, 2025

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 recap **relational databases** using **PostgreSQL**.

1.3 PostgreSQL

We will focus on PostgreSQL as our primary relational 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**.

1.4 Text-based query language

DBMS normally 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 (part of DML)

Usually a common language provides all of these functions. SQL best known and most commonly employed:

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

1.5 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).

1.6 Client-server

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

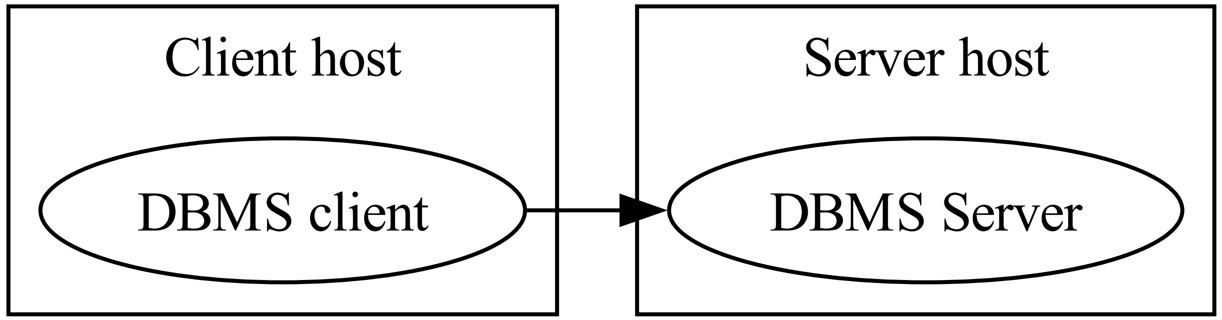


Figure 1: Client-Server model

1.7 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
- Programs / apps / scripts can be written to access database servers using a client library.
  - Generally the bundled text-mode client uses this library internally too!

Important note about clients / servers

- 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.

1.8 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.

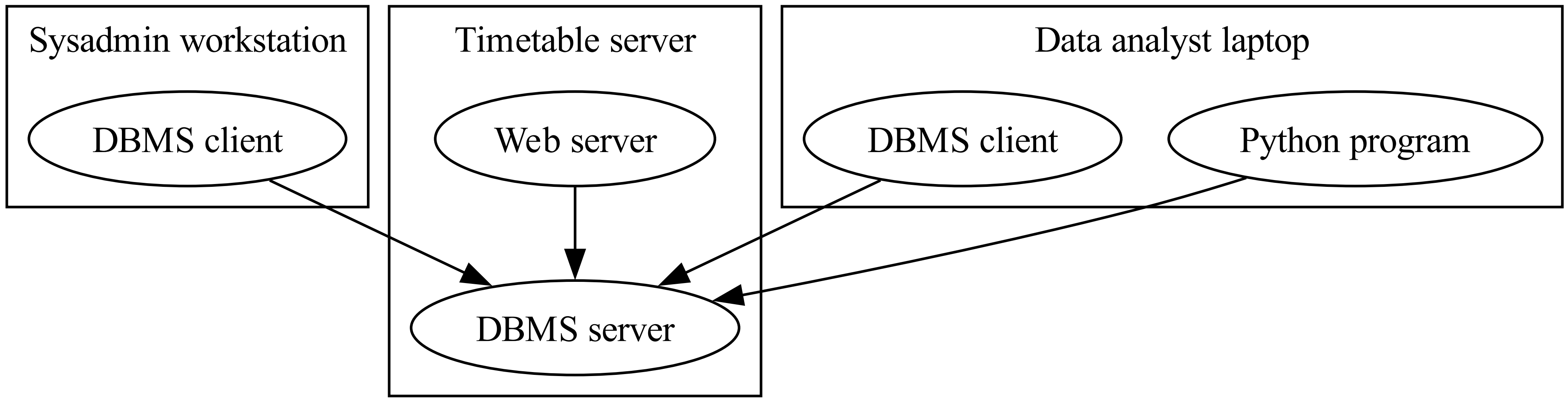


Figure 2: Concurrent access to a college timetable database

1.9 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.

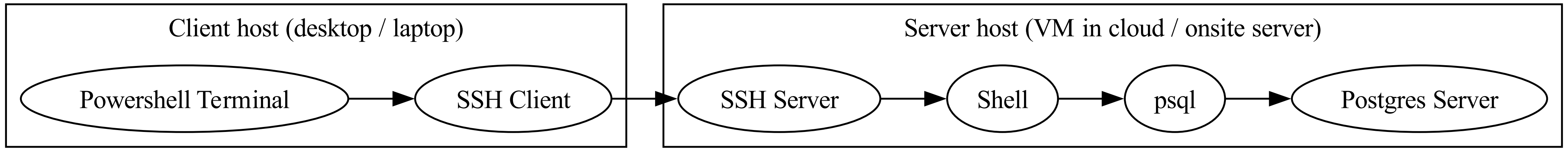

Client host (desktop / laptop)
Powershell Terminal
SSH Client
Server host (VM in cloud / onsite server)
SSH Server
Shell
psql
Postgres Server

Figure 3: SSH access to a remote database

2 Relational databases (RDBMS)

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!

2.1 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.

2.2 Information Rule

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

2.2.1 Relational data structure

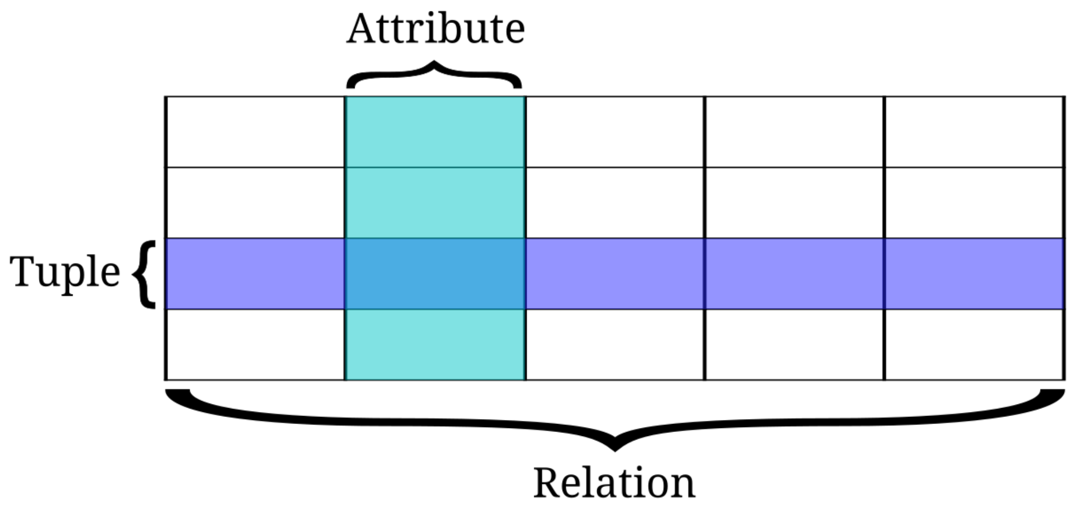


Figure 4: Relational model terminology

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!

2.3 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.

2.3.1 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.

2.4 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

2.4.1 Unique constraint

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

2.4.2 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.

2.5 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.

2.6 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.

2.7 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.

2.8 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.*

2.9 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.

2.10 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.

2.11 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!

2.12 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.

2.13 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 Single-table databases

Recall that:

- A PostgreSQL *cluster* provides:
- One or more **databases** that each have
- (One or more **schemas** that each hold)
- One or more **tables**

where each table:

- has separate **columns** holding individual attribute values
- stores data each record in **rows**

3.1 Data definition language

Key concepts will be easily learned through repeated practice and reference to

CREATE table to define table name, columns (type, constraints)

ALTER table for incremental changes.

DROP table without warning!

Sometimes it's easiest to specify information up front in the `CREATE` statement.

Often it's better to start with a workable definition and amend using`ALTER`.

3.2 Sample problem

We will mainly do this by example.

Example

Mary is a local crafter who makes her own pottery. She runs a small shop where she sells her own wares.
At the moment she writes down all sales in a book:

- Date and sometimes time
- Description of what sold
- Quantity
- Price
- Total

Having done a database class, she realises that she could easily put together a table to replace the book.

3.3 Table names

Following a consistent and workable pattern with table names will help a lot:

1. names should be **descriptive**
   - decide on a sensible structure on pluralisation
2. try to keep names reasonablly **short**
   - You will be typing them a lot!
3. do not use spaces, use the underscore instead (_)
   - spaces in most text-based systems are a receipe for disaster!
4. table names are **case insensitive**

Table names *can* be changed afterwards.

Our table will be called **sales**.

3.4 Column names

Similar set of rules for columns as for tables:

Again following a consistent and workable pattern with column names will help a lot:

1. names should be **descriptive**
2. try to keep names reasonablly **short**
3. do not use spaces, use the underscore instead (_)

Column names *can* be changed afterwards.

Example

In our example, we'll choose the following column names:

1. `timestamp`
2. `description`
3. `quantity`
4. `total_price`

4 Datatypes and constraints

Choosing the correct datatypes is vitally important for making a usable database!

TEXT for text (only!)

- Other database systems use CHAR, VARCHAR etc. Avoid!

Numeric data largely falls into the categories of: integer, fixed precision and arbitary precision data.

INT for integers

- Also have `SMALLINT`, `BIGINT` etc for range

NUMERIC for numeric data incl. decimals and currency

- Easy to make mistakes choosing float instead!

4.1 Good practices

1. Do not store numbers as text.
2. Do not store boolean true / false (or any synonyms) as text.
3. Do not sure enumerated types as text. Either use an ENUM and/or a foreign-keyed table.
4. Anything where you need absolute precision after the decimal point must be NUMERIC, not FLOAT.
5. Do not be tempted to store a single logical date/time as separate date and time columns. Always use a single timestamp for this.

4.2 NOT NULL

The NOT NULL constraint prevents a particular column value being null. Writes will fail if set to NULL or if unspecified unless default column value specified.

```
-- During table creation:
CREATE TABLE tasks (
   description text not null,
   /* other columns */
);
```

4.2.1 Changing NOT NULL status

Whether NULL should be allowed depends on problem of interest. By default NULL is allowed. Must specify NOT NULL if not.

Can modify the column's NULLable status afterwards:

```
-- make a column NOT NULL
ALTER TABLE tasks ALTER COLUMN description SET NOT NULL

-- remove the NOT NULL constraint from a column
ALTER TABLE tasks ALTER COLUMN description DROP NOT NULL
```

Example

In our example we will set all columns to not null, as none are optional.

4.3 UNIQUE

The UNIQUE constraint prohibits two rows from having an equal set of attribute values for the columns specified:

- Use UNIQUE after column definition if column should be unique
- Use UNIQUE(col1, col2) if two or more columns should be unique after the column definition

```
-- During table creation:
CREATE TABLE tasks (
   description text unique,
   /* other columns as necessary */
   project_code text,
   subproject_code text,
   unique(project,subproject)
);
```

Example

In our example we’ve no UNIQUE columns (yet!)

4.3.1 Modifying UNIQUE afterwards

Just as with NOT NULL we can modify UNIQUE later on:

```
-- Afterwards
ALTER TABLE tasks ADD UNIQUE (project, subproject);
```

4.4 Default values

Default values are automatically inserted when unspecified in INSERT statement. Often used in conjunction with NOT NULL. Can be static value or based on functions (see later).

```
-- when creating a table
CREATE TABLE sales (
    /* other columns  */
    quantity smallint not null default 1;
    timestamp timestamp not null default now();
);
```

4.4.1 Altering DEFAULT

```
-- afterwards
ALTER TABLE tasks ALTER COLUMN description SET default 'xyz'
ALTER TABLE tasks ALTER COLUMN description DROP default
```

5 Primary key

Primary key value is a **unique** AND **not null**.

Unambiguously identifies each row.

Every table should have 1 primary key. Cannot have more than 1.

Most DBMS do not enforce this, but you should consider it mandatory. Use PRIMARY KEY if column is the primary key.

Example

In our example, we *could* use the timestamp as the primary key.
But it would be much better to make an autoincrementing integer `id` column.

```
-- specifying primary key
CREATE TABLE accounts (
    account_number bigint primary key
    /* other columns as necessary */
);

-- primary key covering two columns
CREATE TABLE accounts (
    branch bigint, -- not null automatic
    account_number bigint,  -- not null automatic
    /* other columns */
    primary key (branch, account_number)
    /* other constraints etc. */
)
```

5.1 Auto-increment column

We can use the column types SERIAL or BIGSERIAL for auto-incrementing columns. Generally prefer SERIAL or BIGSERIAL.

Shorthand for default value based on *sequence generator* (later on).

```
CREATE TABLE sales (
    id BIGSERIAL PRIMARY KEY,
    /* other columns & constraints */
);
```

5.2 DML operations

The main **Data Manipulation Language** operations you'll use are grouped into:

INSERT rows populating specified columns with data given.

UPDATE table, optionally select rows with `WHERE`.

DELETE rows from a table, optionally select rows with `WHERE`.

We'll mainly cover these by example!

6 Transaction control

PostgreSQL transaction control can be used in simplest form to give basic “undo” capability:

```
-- start a new transaction
BEGIN;

/* execute SQL statements then either: */

COMMIT; /* save the changes */
-- or
ROLLBACK; /* undo the changes */
```

6.1 Error handling

If we don’t BEGIN a transaction we implicitly BEGIN before the statement and COMMIT after it. Errors (e.g. syntax) will abort the transaction, requiring a ROLLBACK before any more statements will be accepted.

6.2 Savepoints

```
BEGIN;

/* sql statement block 1 */

-- define a savepoint
SAVEPOINT sp1;

/* sql statement block 2 */

-- rollback to the savepoint undoes block 2;
ROLLBACK TO sp1;

-- then either:
COMMIT; /* or ROLLBACK */
```

7 Multi-table databases

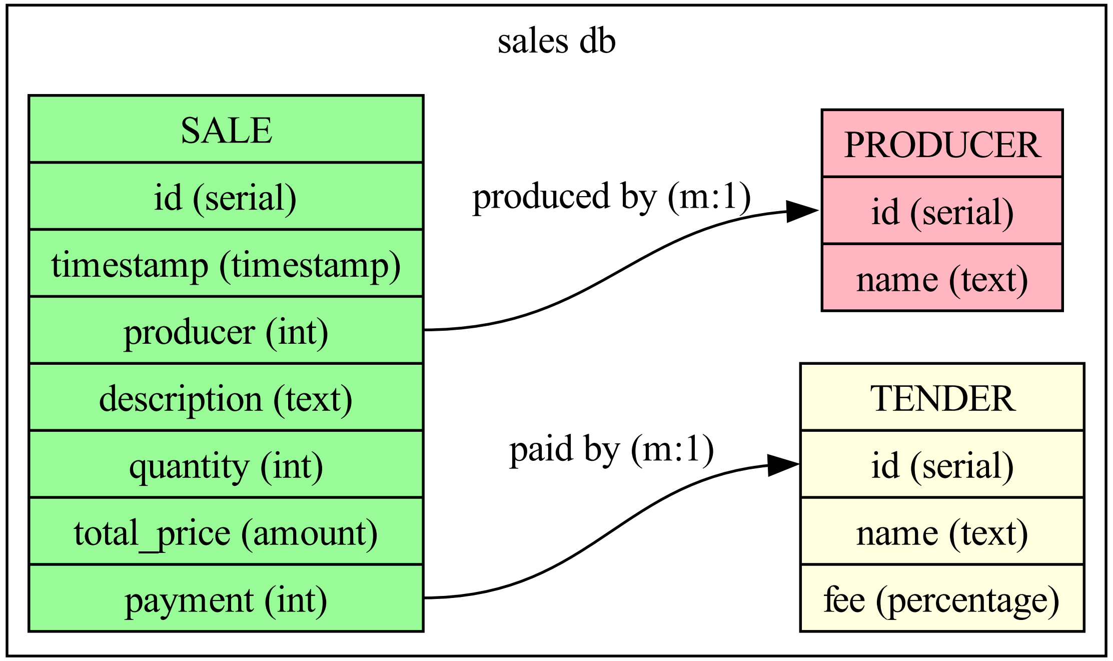
7.1 Scenario

A group of local artisan producers operate a popup stall for a 3-month period. They contract a local IT expert to build a simple sales tracking and reporting system.

Main requirements

1. Sales must be numbered and timestamped.
2. Every sale is separate (no concept of a “basket” for simplicity!)
3. Stall accepts cash and card tender.
    - Card payments have a 1.68% processor fee (which may vary).
    - May be requirement to accept additional payment types during period.
4. Sale must capture producer, description, quantity sold, amount paid and tender.
5. Everyone will be paid at the end of the 3-month period.
6. Database must be able to produce the following reports:

7.2 Implementation with 3 tables



7.3 Key decisions

1. **Auto-numbered** `id` column using `SERIAL`.
2. **Primary key** will be the auto-numbered `id` column.
3. All fields will be **not null**.
4. Every sale must match a valid:

   producer from the `producer` table.

   tender from the `tender` table.
5. Storage of numerical values:

   Amounts as `NUMERIC(6,2)` typed `amount`.

   Rate as `NUMERIC(6,2)` typed `rate`.

8 Domains

Domains are user-defined types based on underlying type. Defaults to NULL allowed, best to define any NOT NULL conditions on the underlying columns. CHECK constraints can be defined.

```
/* Number to hold 1-10 user rating */
CREATE DOMAIN rating AS integer CHECK ( VALUE >= 1 AND VALUE <=10 );

/* Just use the domain as type when creating table */
CREATE TABLE restaurants (
    id bigserial primary key,
    /* creating two columns using our domain: */
    visitor_rating rating not null,
    reviewer_rating rating,
    /* other columns */
);
```

9 Foreign keys

Foreign keys require that values in a column (or a group of columns) must match the values appearing in some row of another table. This maintains the referential integrity.

```
/* each employee must be in a valid department */

CREATE TABLE department (
id bigserial primary key,
name text not null unique
/* other fields as required */
);

CREATE TABLE employee (
id bigserial primary key,
surname text not null,
firstname text not null,
department bigint not null REFERENCES department
/* other fields as required */
)
```

9.1 DELETE / UPDATE behaviour

Possible behaviours: NO ACTION, CASCADE, SET NULL.

```
CREATE TABLE product (
/* other fields */

department bigint references departments,
/* NO ACTION is the default, prohibits conflicting delete */

supplier bigint references suppliers ON DELETE CASCADE,
/* DELETE in suppliers deletes linked products */

policy bigint references policies ON DELETE SET NULL,
/* SETS product.policy to NULL when row in policies deleted  */

);
```

9.2 JOIN

The JOIN operation permits queries across more than one table. See both the JOIN tutorial and the Table expressions section from Postgres manual for full details.

Assume R1 to be a row of Table T1. Similarly R2 for T2. Normally should explicitly specify columns required and use table prefix to avoid ambiguity.

9.3 INNER JOIN

For each row R1 of T1, the joined table has a row for each row in T2 that satisfies the join condition with R1.

9.4 LEFT JOIN

Same as INNER JOIN except that output also includes any row in T1 that does not match one or more rows in T2. Null values are substituted for T2 in the output row.

9.5 RIGHT JOIN

Similar to LEFT JOIN. Same as INNER JOIN, except any row in T2 that does not match ≥ 1 rows in T1 will be output. Null values are subtituted for T1 columns in the otuput row.

9.6 FULL JOIN

Similar to combination of LEFT and RIGHT JOIN. INNER JOIN performed. Then rows in T1 without corresponding T2 output with nulls for T2. Same again, rows in T2 without corresponding T1 rows output with nulls for T1.

10 Views

Views are defined by @connolly:2015:database as:

> The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a virtual relation that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of the request.

```sql
/* Creation syntax: */
CREATE VIEW my_view AS
SELECT ... ;
/* select statement can be any valid select */

/* VIEW can be selected like any other table */
SELECT * FROM my_view ;
```