Unit 6: Security management

- Security management
- Tools
- Users (creation, modification, deletion)
- Privileges (assignment, withdrawal)
- Views
- Roles (creation, deletion)
- Views with information about roles
- Profiles

Database security

- Database security has to do with protection against potential unauthorized access
- Sometimes, it may interfere with the concept of privacy
- Aspects to consider:
  - Legal and ethical
  - Public and private information levels
  - Physical controls
  - User identification
  - Operating system control

Database security

- The DBMS must keep control and record of:
  - Users
  - Access
  - Allowed operations
- Among the different users we can find:
  - Database administrator → all privileges
  - Programmers and managers → create, delete, apply privileges on created objects
  - Normal users → query some data, and maybe update

Database security

- All in all, security in a database is based on the following features:
  - Confidentiality → only certain users can access certain data, and perform certain operations on them
  - Integrity → when an operation is carried out on the data, these must be preserved without losing information, and kept being stored represented without significant problems
  - Availability → data must be available to whoever requires, as long as they have the authority to access them

Tools

- The security measures and services may be diverse:
  - Physical: access control to the equipment or physical system. Example: access card
  - Personnel: restrict access to authorized personnel. For instance: eye recognition, fingerprint, etc.
  - Operating system and/or network: user accounts, passwords, restricted access to certain areas of information, etc.

Tools

- In addition to the previously mentioned, when somebody has access to a system, a new level of security is to be considered: the database management system.
- The DBMS has operations to manage security through permissions about users and files audit (logs)
- Usual tools in relational DBMS:
  - Integrity constraints
  - User profiles
  - Orders and permissions
  - Views

Tools

- How is security guaranteed with the dedicated DBMS module?
  - Identification and authorization of users
  - Authorization depending on other factors (terminal, time of day, …)
  - Data encryption
  - Accounts
  - Audit and record of accesses (log)

Services

- Security can be preserved both in discretionary way (on users) and in mandatory way (through access levels)
- The services for doing so are:
  - Authentication → correct identification and validation
  - Cryptography → encrypt the data
  - Accounts → limit operations and access
  - IP security → network defence and protection

Integrity constraints

- Integrity constraints are already known, from tables creation in relational systems
- In SQL we have a wide range of mechanisms to add constraints that do not corrupt the data
- Some instructions are inherent to the relational model, others to the standard language (SQL) itself, and finally the last ones are specific to specific DBMS (Oracle, MySQL, PostgreSQL, ...)

Integrity constraints

- Integrity constraints in SQL:
  - Keys:
    - Primary → PRIMARY KEY
    - Foreign → FOREIGN KEY … REFERENCES ...
  - Data types: BOOLEAN, (VAR)CHAR, NUMERIC, INTEGER, DATE, TIME, interval, array, ...
  - Non-null values → NOT NULL
  - Non-repeating values → UNIQUE
  - Delimited or restricted values → CHECK
  - User/programmer defined data types

Integrity constraints

- Delimited or restricted values → via the CHECK clause and a condition
- Example:

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR (50),
    last_name VARCHAR(50),
    birth_date DATE CHECK (birth_date > '1950-01-01'),
    joined_date DATE CHECK (joined_date > birth_date),
    salary numeric CHECK(salary > 0)
);
```

- An error will occur when entering data that do not meet the above conditions

Integrity constraints

- Conditions can be multiple, or as complex as desired
- Example:

```
ALTER TABLE prices
ADD CONSTRAINT price_discount_check
CHECK(
    price_value > 0
    AND discount >= 0
    AND price_value > discount
);
```

Integrity constraints

User/programmer defined data types:

- CREATE DOMAIN: instruction that defines a new domain (understood as a subset of values of an existing data type)
- CREATE TYPE: statement that allows to create a new data type (generally used within stored procedures for database programming)

Integrity constraints

- Example: add a new domain that validates the non-use of spaces and tabs within a text string:

  ```
  CREATE DOMAIN no_spaces_text AS
  VARCHAR NOT NULL CHECK (value !~ '\s');
  ```

- Domain use:

  ```
  CREATE TABLE mailing_list (
      id serial PRIMARY KEY,
      first_name no_spaces_text,
      last_name no_spaces_text,
      email VARCHAR NOT NULL
  );
  ```

Integrity constraints

- Type creation example:

```
CREATE TYPE film_summary AS (
    film_id INT,
    title VARCHAR,
    release_year SMALLINT
);
```

Users

- A user is anyone who has contact with the database system
- It corresponds to an entity (which could perfectly be another software) or a person who has the ability to access the database
- Each user has different permissions and a different scope to perform operations
- We will be able to create and enable users in a customized way, without necessarily having to stick to the standard profiles (administrator, programmer, end user,...)

User creation

- Creating users specifically in PostgreSQL is very easy → there is a create user application in the bin directory
- Apart from the above, we are going to see how to do the same thing with a standard SQL statement
- Once a user is created, they can be granted certain permissions
- SELECT username FROM pg_user;

User creation

- With SQL statements:

```
CREATE USER paca WITH PASSWORD 'password';

GRANT ALL PRIVILEGES ON DATABASE jardineria TO paca; -- paca now has super powers!

\q -- only for Postgres
```

- Connecting to the database with the new user:

```
psql -d jardineria -U paca
```

User Modification

- If at any time we change our needs or opinion… → ALTER USER statement:

  ALTER USER <user> [WITH] <options>;

- Examples:
  - ALTER USER paca WITH PASSWORD 'newpassword';
  - ALTER USER ambrosio VALID UNTIL 'May 4 12:00:00 2025 +1';
  - ALTER USER anselma CREATEUSER CREATEDB;

User deletion

- Users are born (sometimes they grow up) and at some point their useful life comes to an end
- In that case… → DROP USER statement
- Example:

  DROP USER lucas;

  -- See you later!

Privileges

- Possible permissions to be granted (to users):
  - SELECT → allows to query data
  - INSERT → new data can be entered
  - UPDATE → existing data can be updated
  - DELETE → possibility to delete existing data
  - REFERENCES → allows the creation of foreign keys
  - TRIGGER → possibility to define triggers
  - RULE → allows the creation of rules

Privilege assignment

- GRANT <privileges> | ALL

  ON <table(s)>

  TO <user> | <role>;
- Examples:
  - GRANT ALL ON film TO juan;
  - GRANT SELECT ON employees TO ambrosio;
  - GRANT INSERT, UPDATE ON employees TO anselma;

Privileges withdrawal

- REVOKE <privilege> | ALL

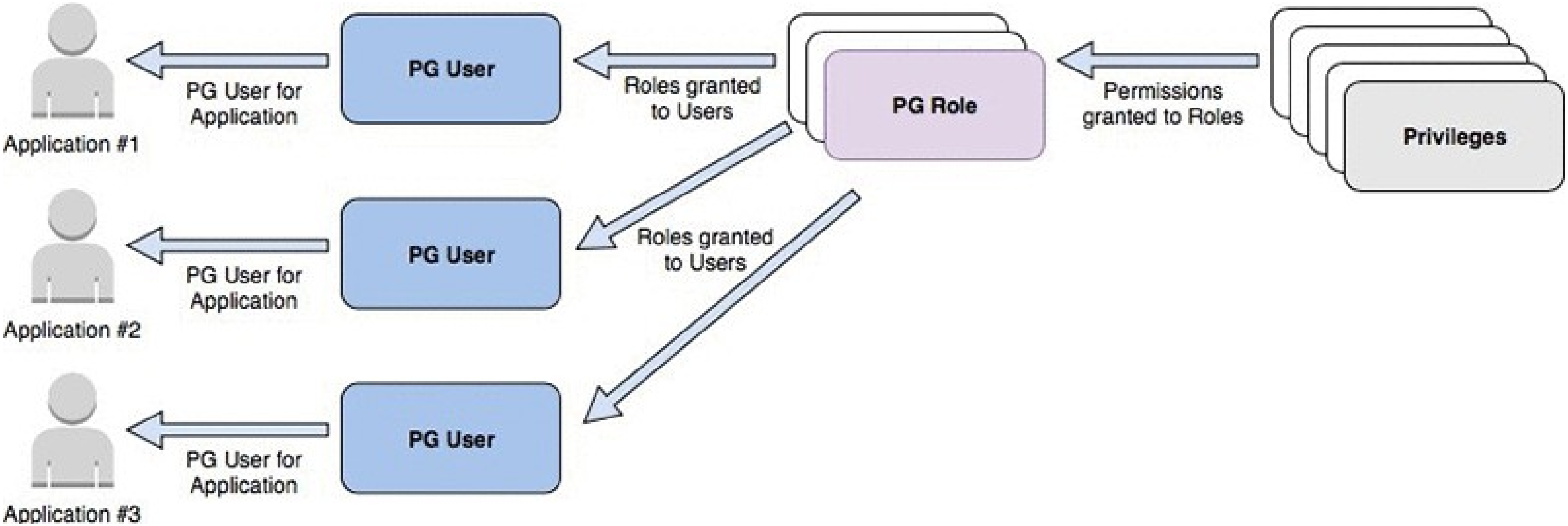
  ON <TABLE> <table> |ALL TABLES

  FROM <user> | <role>;
- Examples:
  - REVOKE CREATEUSER FROM anselma;
  - REVOKE ALL ON TABLE customers FROM paca;
  - REVOKE SELECT ON employees FROM ambrosio;

Roles

- Roles are basically permission feature sets tagged with a name that can be applied to database users
- By using roles, the management of permissions by groups is greatly facilitated, since it allows different privileges to be assigned to multiple users according to the group to which they belong
- Roles can be equivalent to users or groups depending on how they are used

Roles



Application #1
PG User for Application
PG User
Roles granted to Users
PG Role
Permissions granted to Roles
Privileges
Application #2
PG User for Application
PG User
Roles granted to Users
Application #3
PG User for Application
PG User

Create and delete roles

- CREATE ROLE <role> [WITH <options>];
- Examples:
  - CREATE ROLE bank_manager;
  - CREATE ROLE admin WITH CREATEDB CREATEROLE;
- To view existing roles:
  - SELECT rolename FROM pg_roles;
  - \du -- Postgres only

Create and delete roles

- How to create a super-user (with all powers):
- CREATE ROLE god_mode

  SUPERUSER

  LOGIN

  PASSWORD 'vamosrafa';

Create and delete roles

- Example:

```
CREATE ROLE dev_api WITH
LOGIN
PASSWORD 'dev_api_pass'
VALID UNTIL '2030-01-01';
```

Create and delete roles

- The CREATE GROUP statement is equivalent to CREATE ROLE → it is an alias
- CREATE GROUP <name>

  [WITH <options>]
- The difference is that it is a non-standard statement in SQL (it is exclusive to Postgres systems)

Assign roles to users

- Syntax:

  GRANT <role> TO <user(s)>;

- Examples:
  - GRANT god_mode TO anselma, ambrosio;
  - GRANT customer_support TO paca;

Views

- When we perform an operation on a table in the relational model, the result is always another table
- Likewise, when a query is carried out, the result is also a table
- A view is simply a given query that is stored with a given name → useful when dealing with recurring data or operation, or the query presents some complexity

Views

- Therefore, when creating a view we are storing a query and also a table
- The main practical utility of views, apart from efficiency issues, is to hide part of the conceptual model from some users
- Example: an employee of the advertising department of a bank should only have access to a list of clients
- However, a branch manager will have access to more customer data if it belongs to that branch, but less information from other customers

Views

- In short → any relationship that is NOT part of the original conceptual model but is visible to some users (as if it was a "virtual" table) is a view
- Therefore, given a set of relations, it is possible to define and have a large number of views on that set

Views

- The application of the views is done mainly for security reasons → they restrict access to certain users and data
- Normally views are NOT stored, but recalculated for each query when required
- However, a series of techniques are applied to them that improve their efficiency and performance so that the user does not perceive this difference
- Apart from the above, they are manipulated like any other table in the database

Views

- Syntax:

```
CREATE VIEW <view name>
AS <subquery>;
```

- Example:

```
CREATE_VIEW customers_zaidin AS (
    SELECT name, address
    FROM customers
    WHERE branch = 'Zaidín'
);
```

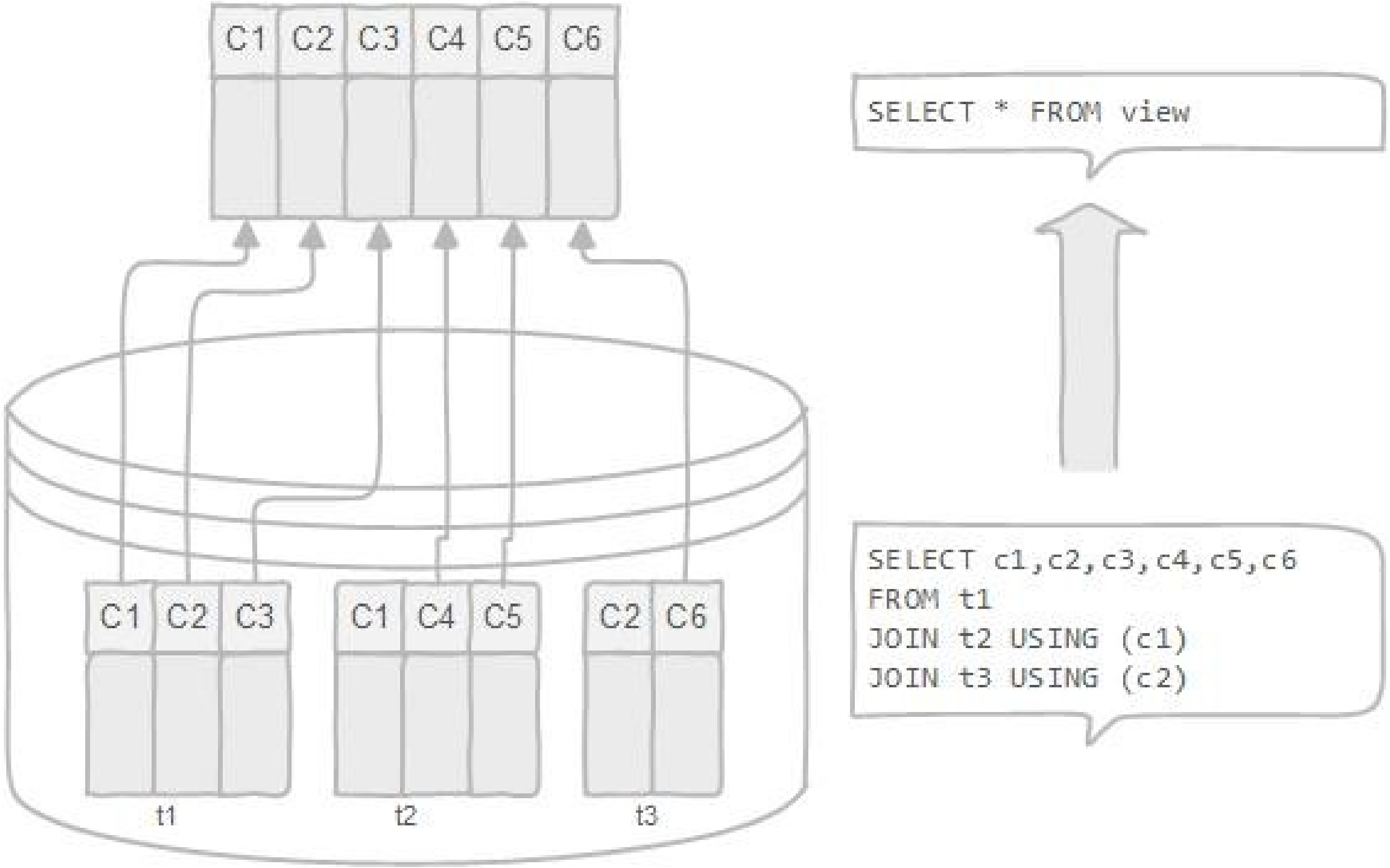
Views

- Once the view is defined, it can be used like any other table:
- SELECT address

  FROM customers_zaidin

  WHERE name LIKE '%Fernández%';

Views



```
SELECT * FROM view
```

```
SELECT c1,c2,c3,c4,c5,c6
FROM t1
JOIN t2 USING (c1)
JOIN t3 USING (c2)
```

Views

- In order to change the definition of a view:

  ```
  ALTER VIEW <view name>
  RENAME TO <new name>;
  ```

- To delete a view the statement is:

  ```
  DROP VIEW [ IF EXISTS ] <view name>
  [CASCADE];
  ```

- The CASCADE clause is used to delete referenced data recursively

Views

The drawbacks that views can pose:

- Data update → is it changed in the original table?
- Insertion → if the view includes a piece of data but not a field that is part of the key, inserting data into the view leads to the problem of not having all the data for a new record in the source table
- Deletion → should data in the original table be deleted if a part of it is deleted in the view?

Views

- To solve the problems raised, database management systems impose a number of conditions
- For example (Postgres):
  - The record of a view must correspond exactly to another in an original table and be equally updatable
  - Updating is not allowed if the view contains certain clauses: GROUP BY, HAVING, LIMIT, OFFSET, DISTINCT, WITH, UNION, INTERSECT, EXCEPT
  - The view selection also cannot make use of the aggregation operators: SUM, COUNT, AVG, MIN, MAX

Views

- One potential problem with views is a user trying to add data (if allowed) that they can't even see
- The WITH CHECK OPTION clause will allow us to avoid this situation
- Example:
- CREATE VIEW usa_city AS

  SELECT city_id, city, country_id

  FROM city WHERE country_id = 103;

Views

- If the user tries something like...

```
INSERT INTO usa_city (city, country_id)
VALUES('Birmingham', 102);
```

- ...the result will be introduced → unwanted effect
- Fix: add WITH CHECK OPTION to the end of the view definition
- Now the user will not be able to add the previous row as the system will give a validation error on country_id

Views with information about roles

- As with the users, there is a view with information about the roles and the users that are assigned to each one → pg_group

-
  ```
  SELECT username
  
  FROM pg_user, pg_group
  
  WHERE pg_user.usesysid = ANY(pg_group.grolist)
  
  AND pg_group.groname = 'my_role';
  ```

Profiles

- Some database management systems offer this tool that allows to restrict and optimize database system resources
- Through profiles the amount of system and database resources available to a given user can be limited
- If no profiles are defined for a user, the default profile is used, which specifies unlimited resources

Profile creation and deletion

- Profiles can be created using the CREATE PROFILE command, and they can be modified with the ALTER PROFILE statement.
- Syntax:

  CREATE PROFILE <profile name> LIMIT {LIMIT NAME}

  {INTEGER [K | M] | UNLIMITED | DEFAULT };
- UNLIMITED = no limits on a particular resource
- DEFAULT = default profile limit

Profile creation and deletion

- The definition of profiles may consider the following aspects:
  - Sessions per user: maximum number of sessions a user can open
  - CPU per session: expressed in processor time per operation
  - Connection time: maximum duration of a session
  - Idle time: maximum inactive time per session
  - Reads: maximum number of data block read operations per session
  - Composed limit: total allowed cost of resources per session

Profile creation and deletion

- Example:

-
  ```
  CREATE PROFILE my_profile LIMIT
  sessions_per_user 2
  connect_time 5
  idle_time 3
  failed_login_attempts 2;
  ```

Profile creation and deletion

- Delete a profile:
- DROP FILE <profile name> [CASCADE];
- In general, the default profile should be suitable for normal users
- Users with special requirements should have special profiles

Profile enabling / disabling

- To use a profile, the following syntax is used:
- ALTER USER <username>

  PROFILE <profile name>;
- To deactivate a profile, the systems provide more or less complex solutions without the need to delete them, for example using procedures

Profile enabling / disabling

- Profile disabling example:

```
BEGIN
DBMS_SQLTUNE.ALTER_SQL_PROFILE(
    name => 'name_of_my_profile',
    attribute_name => 'STATUS',
    value => 'DISABLED'
);
END;
```