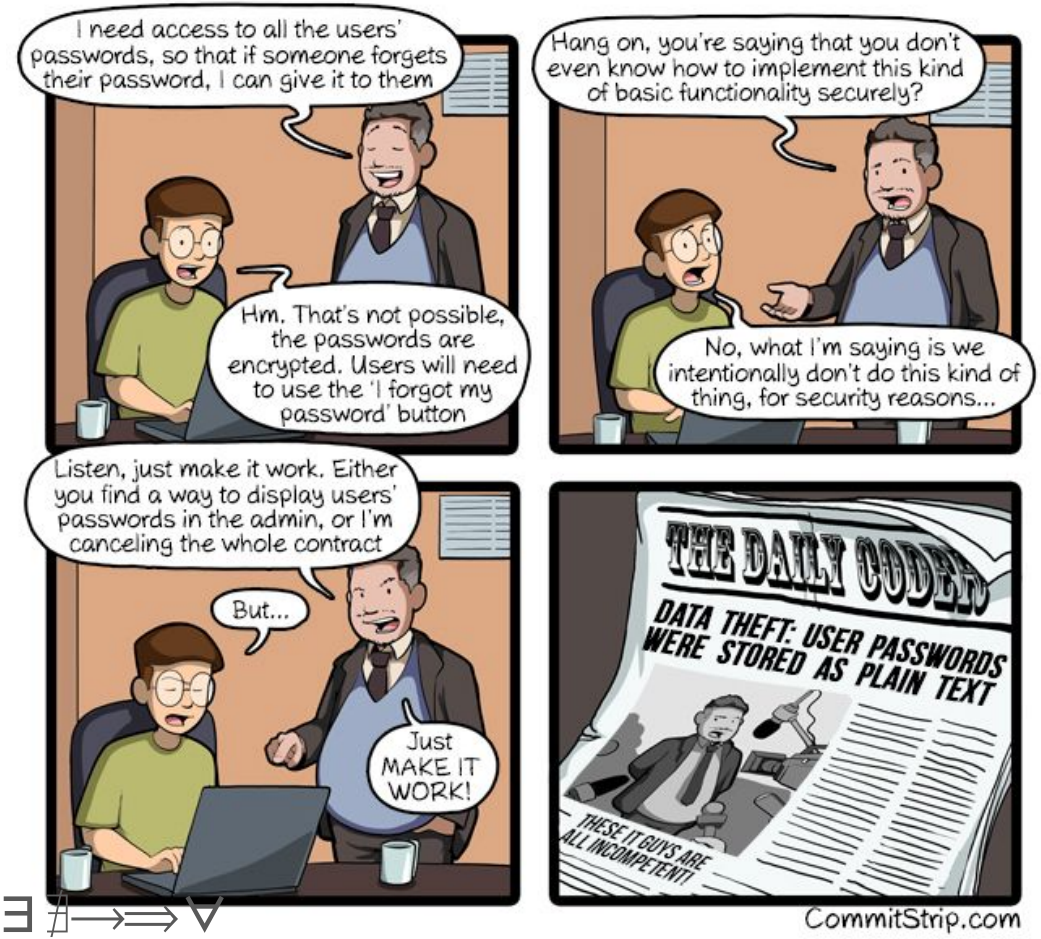
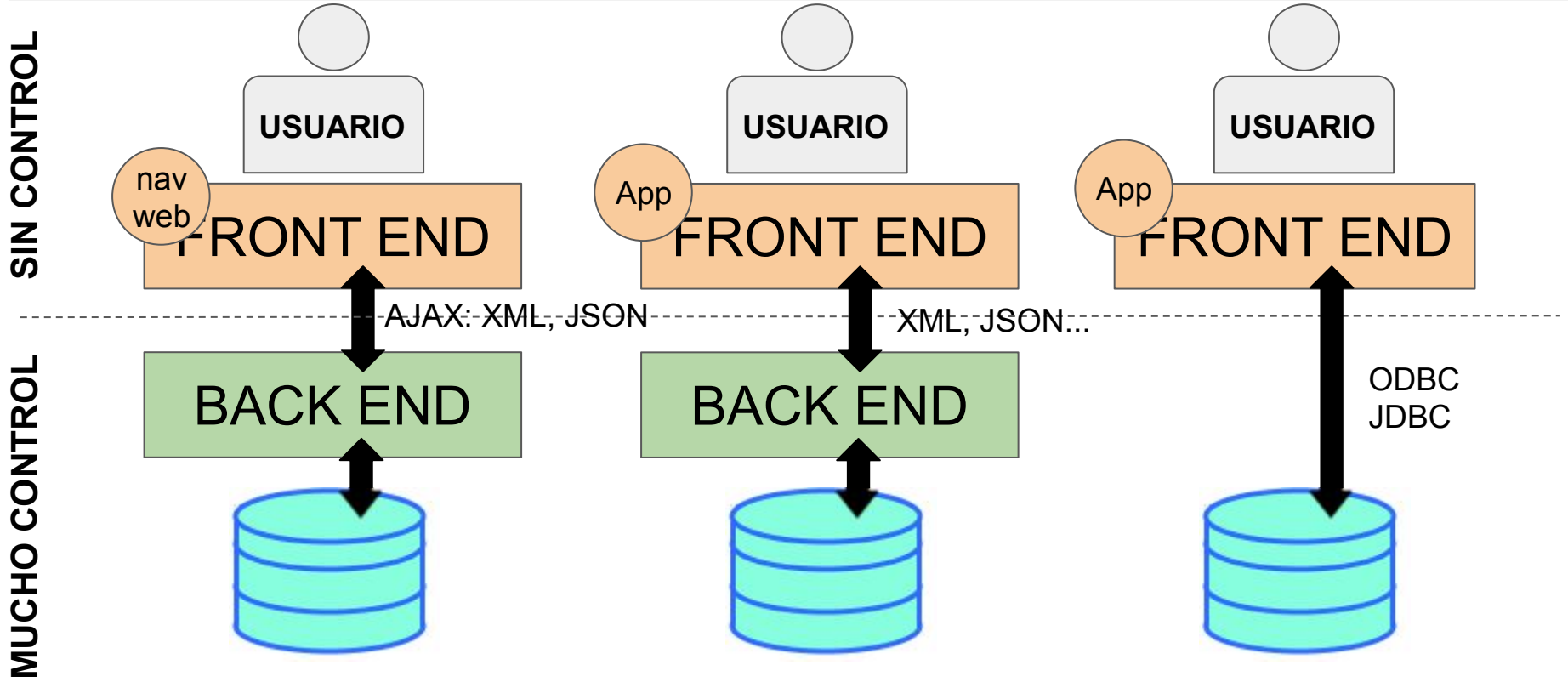


Base de Datos

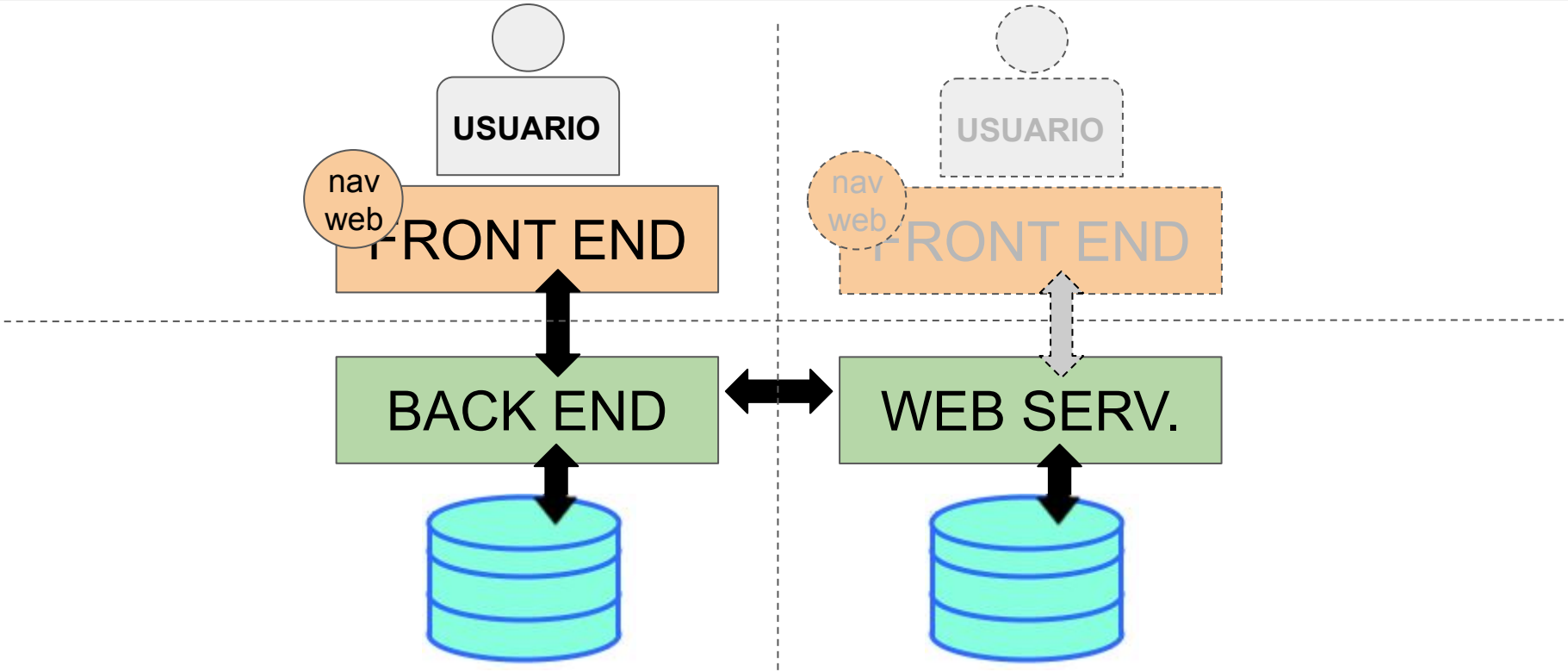
DC. FCEyN
2024-11-06

Emilio Platzer
tute@dc.uba.ar

Arquitectura

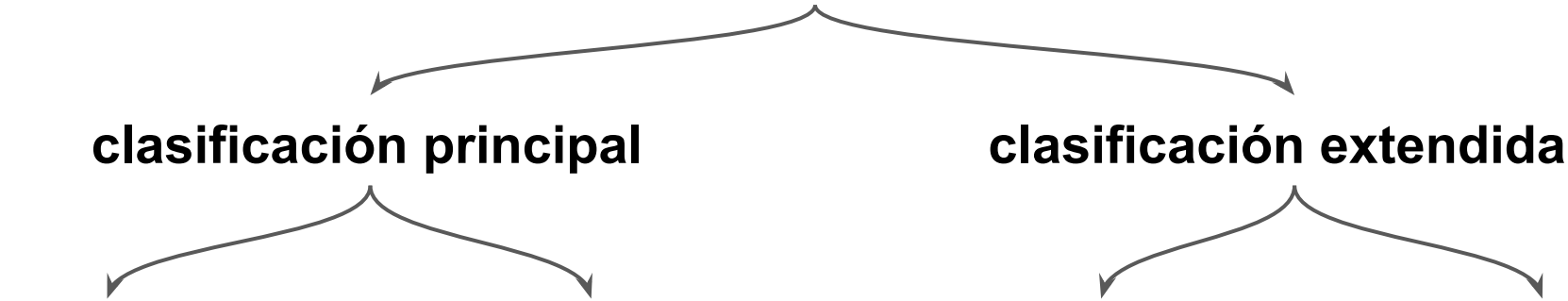


Arquitectura - web services



SQL

el lenguaje



SQL

el lenguaje

clasificación principal

DML
manipulation

instrucciones

SELECT	obtiene datos de la base de datos
INSERT INTO	agregar registros nuevos en una tabla
UPDATE	cambia registros existentes de una tabla
DELETE	borra registros de una tabla
TRUNCATE	vacía el contenido de una tabla

cláusulas

JOIN	junta tablas en un registro
UNION	agrega registros a un resultado
GROUP BY	agrupa registros
ORDER BY	ordena registros

FROM	tablas involucradas
WHERE	filtro de registros
SELECT	sub-consultas

SQL

el lenguaje

clasificación principal

DDL
definition

crea, modifica o elimina tablas, funciones o vistas

TRUNCATE

`CREATE TABLE` crea una tabla a partir de la definición o de un `SELECT`

`ALTER TABLE` modifica la estructura de una tabla agregando, quitando o modificando:

- campos (nombres y tipos) `COLUMN`
- la clave primaria `PRIMARY KEY`
- claves secundarias `UNIQUE`
- relaciones con otras tablas `REFERENCES`
- restricciones `CONSTRAINT`

SQL

el lenguaje

clasificación extendida

TCL
task control

comienza una transacción	`BEGIN TRANSACTION`
finaliza una transacción	`COMMIT`
aborta una transacción	`ROLLBACK`

SQL

el lenguaje

clasificación extendida

DCL
data control

`GRANT` otorga
`REVOKE` quita
→ permisos de acceso y modificación de los objetos de la base de datos

Protección de datos personales

- Protegido por una ley especial
- Aplicado especialmente a bases de datos y procesamiento informático
- Las personas tienen derecho a saber qué datos se conocen sobre sí mismos y a pedir la corrección
- Excluidas las fuentes periodísticas
- Datos personales, datos sensibles (prohibidos), datos de salud

- Cuestiones prácticas
  - Hay que registrar el origen y transferencia de los datos
  - Hay que aplicar medidas de seguridad
  - No se puede recopilar datos porque sí. Tiene que haber un fin lícito

Seguridad

Aspectos

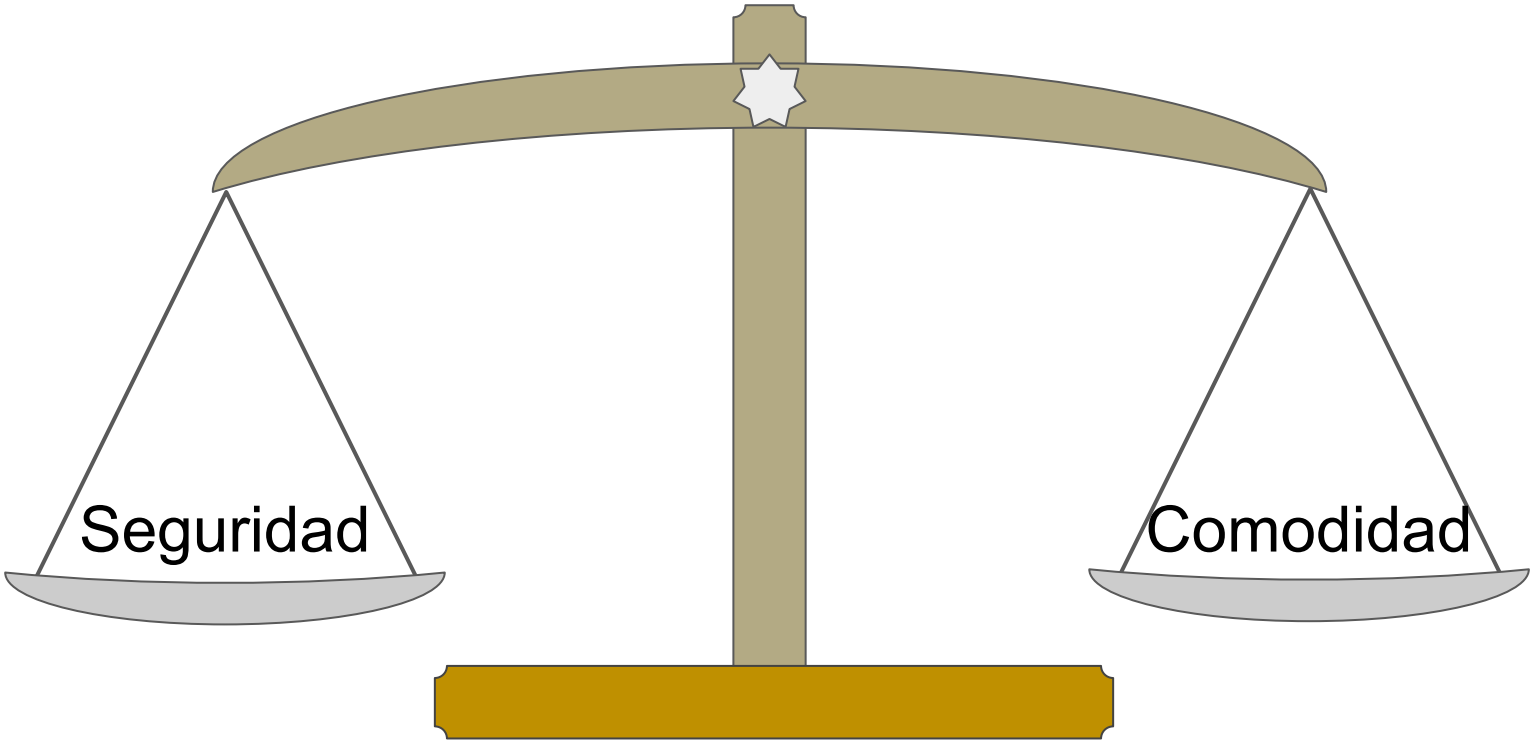
- Integridad
- Confidencialidad
- Disponibilidad

Seguridad

Tipos

- Física
- Lógica
- Administrativa

Seguridad

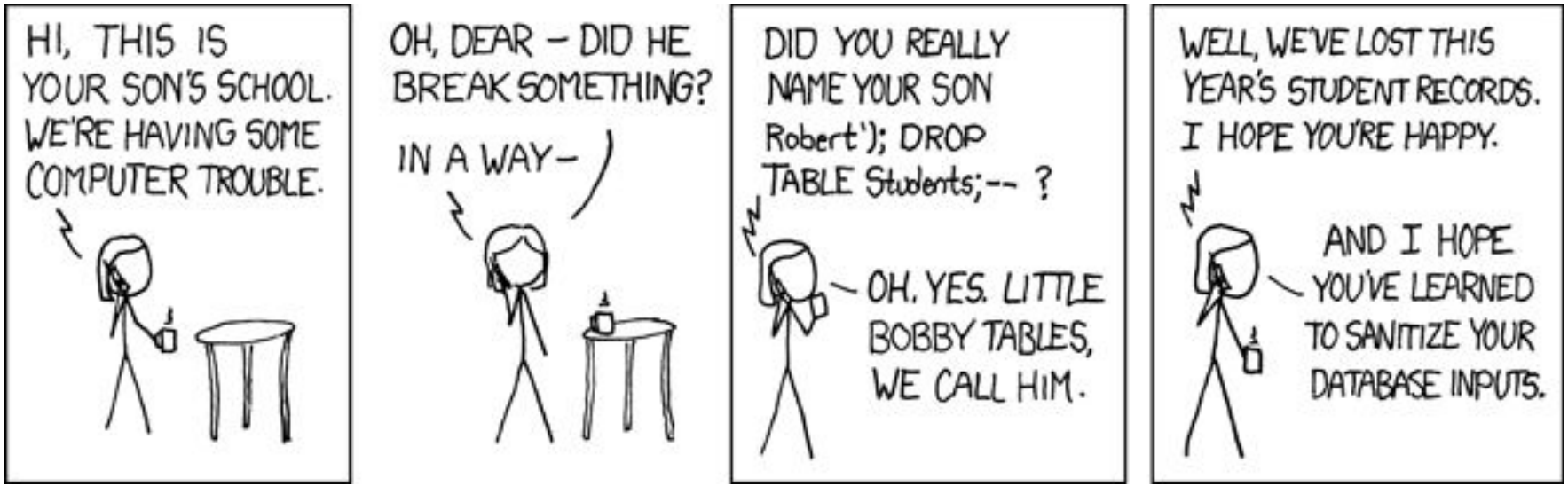


Seguridad - Vulnerabilidades

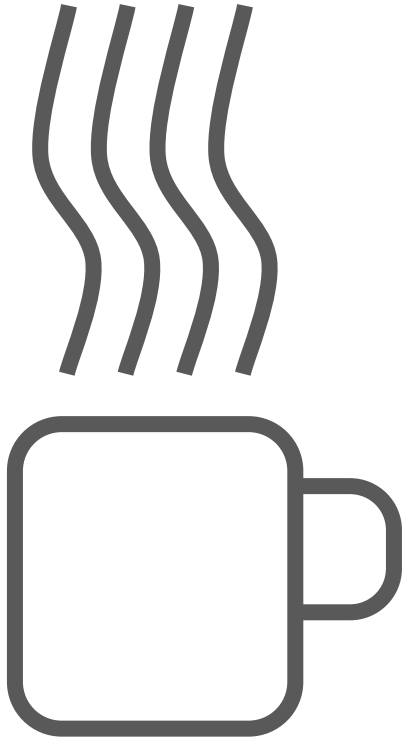
- soft: back-end, base de datos, front-end, navegadores
- hardware: pc, servidores, internet de las cosas
- red: paquetes interceptados o desviados
- entorno / humana / administrativa

Seguridad - Vulnerabilidades

inyección SQL



https://xkcd.com/327/



usuarios

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

Seguridad del Backend vs de la Base de Datos

Cada usuario debe tener el mínimo privilegio que le permita realizar su tarea.

- El usuario del sistema
- El usuario de la base de datos:
  - un usuario db por usuario del sistema
  - un usuario db por rol en el sistema
  - un usuario db para todo el sistema
  - **el usuario que crea el esquema debe ser otro.**

permisos en tablas

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

where *role_specification* can be:

```
    [ GROUP ] role_name
  | PUBLIC
  | CURRENT_ROLE
  | CURRENT_USER
  | SESSION_USER
```

permisos en columnas

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

permisos en base de datos y esquemas

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

```
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

permisos en funciones y procedimientos

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE } routine_name
                [ ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] ) ] [, ...]
          | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA s [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

permisos en permisos

```
GRANT role_name [, ...] TO role_specification [, ...]
    [ WITH { ADMIN | INHERIT | SET } { OPTION | TRUE | FALSE } ]
    [ GRANTED BY role_specification ]

where role_specification can be:

    [ GROUP ] role_name
  | PUBLIC
  | CURRENT_ROLE
  | CURRENT_USER
  | SESSION_USER
```

roles en el backend

Transparencia:

Tiene que haber una manera de simular tener un menor rol

Tengo que poder ver lo mismo que ven mis subordinados:

- No ver menos que ellos
- Decidir ver lo mismo que ellos (para comprobarlo)

esas actividades deben estar garantizadas y auditadas

Los roles de admin o superuser no deben estar activos todo el tiempo

Permisos por fila

```
CREATE POLICY fp_s ON information FOR SELECT
  USING (level <= (SELECT level FROM users WHERE user_name = current_user));

CREATE POLICY fp_u ON information FOR UPDATE
  USING (level <= (SELECT level FROM users WHERE user_name = current_user));
```

Quitar permisos

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM role_specification [, ...]
    [ GRANTED BY role_specification ]
    [ CASCADE | RESTRICT ]
```

Funciones

```
CREATE FUNCTION check_password(uname TEXT, pass TEXT)
RETURNS BOOLEAN AS $$
DECLARE passed BOOLEAN;
BEGIN
    SELECT  (pwd = $2) INTO passed
        FROM    pwds
        WHERE   username = $1;
    RETURN passed;
END;
$$  LANGUAGE plpgsql
    SECURITY DEFINER ~~INVOKER~~
    SET search_path = admin, pg_temp
    ~~IMMUTABLE~~ STABLE ~~VOLATILE~~
    NOT LEAKPROOF;
```

las contraseñas

Son privadas, el admin no debería poder verlas:

- se pueden encriptar con un algoritmo fuerte (no debería ser reversible ni abordarse por fuerza bruta)
- Los hash de las contraseñas deben protegerse al máximo posible.
- Las contraseñas no se deben poner en duro en el código
  - ni en .conf en el repositorio de código (git, svn, etc…)
  - si hay contraseñas de test comunes en producción deben prohibirse
  - los .conf en el backend deben estar protegidos al máximo a nivel del S.O.
- No mandar de ejemplo contraseñas que no queremos (predicar con el ejemplo)
- Tener política de contraseñas (no son cepillos de dientes)

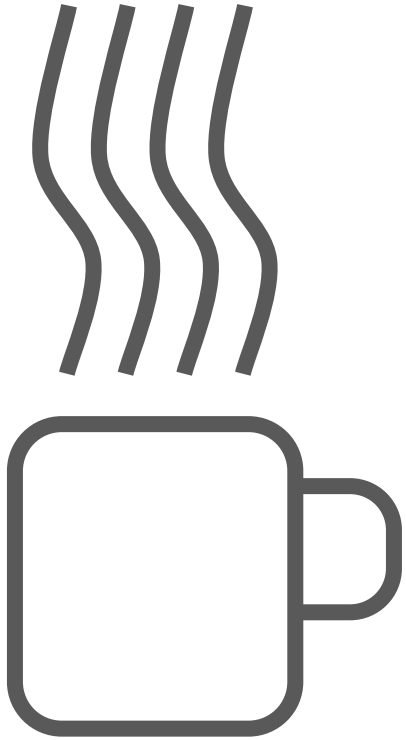
el control de usuarios

- Usar módulos estándar (no inventar la pólvora).
- Evaluar el uso de SSO
- Controlar que el usuario esté activo y tenga el rol correcto todo el tiempo (no solo al iniciar sesión).
- Tener logs o estadísticas de conexiones, intentos de acceso, etc…
-

Inyección de código

Si se almacena código en una variable (SQL, HTML, BASH, JS, JSON…):

- Enviar lo fijo (estructurado) separado de lo variable (lista de valores) y que en un solo punto se haga la unión, sanitización o quoting
  - Si no es posible utilizar una función de quoting unificada.
- Utilizar un único punto de contacto con otros sistemas (llamado a base de datos, generación de HTML, CSS, comandos del S.O., etc…)



TRANSACCIONES DE LARGA DURACIÓN

- Estamos en presencia de una “long duration transaction” en las siguientes situaciones
    - Una transacción que necesita modificar muchos registros de una base de datos , por ejemplo, 6 millones
    - Un workflow, que para ejecutarse necesita “cruzar” la frontera de diferentes bases de datos. Que pasa cuando recargamos la tarjeta SUBE desde el home banking del Banco Nación?

Long duration: TRANSACCIONES MASIVAS

- Cuando se insertan (populate) millones de registros.
    - Instrucciones especiales de bulk copy: **`COPY FROM`**
- A veces la base de datos no puede ejecutarlo como una única transacción por el tiempo que tarda en ejecutarse y por la cantidad de información que necesita almacenar para poder hacer rollback
- No hay instrucciones especiales para **`UPDATE`**
- **Cuidado con los triggers y los índices**

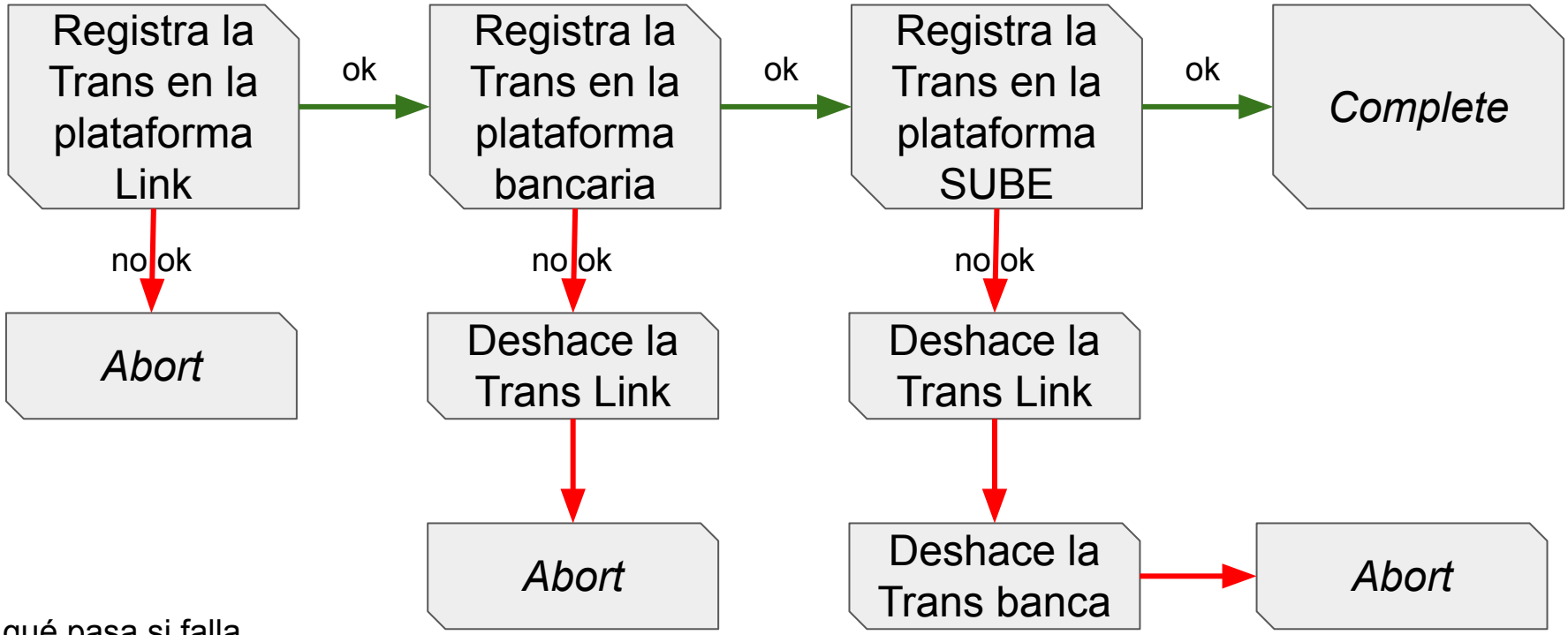
Long duration: TRANSACCIONES INTER SISTEMAS

- Para poder implementar este tipo de transacciones necesitamos de un mecanismo externo u orquestador que construya las “operaciones de compensación” en una “saga” de operaciones para cada sistema.
- Una saga es un conjunto de acciones que juntas conforman una “long duration transaction”. Está formada por:
  - Las acciones
  - Un grafo dirigido, que tiene como nodos las acciones + Abort +Complet
  - Una marca de cual es el primer nodo

TRANSACCIONES INTER SISTEMAS

- Los caminos del grafo representan posible cursos de acción
- Para asegurar la “atomicidad” cada acción se considera una transacción y para conseguir la atomicidad global se usan transacciones de compensación
- Para cada acción A, se define su “compensación”, A^{-1}
- El objetivo es que si una base de dato está en un estado S, y le aplicó en forma sucesiva la acción A y su acción de compensación (A^{-1}), vuelvo al mismo estado S
- Si necesito revertir una saga tengo que ejecutar las acciones de compensación en el orden inverso al que fueron ejecutadas las acciones.

SAGA DE LA SUBE



¿qué pasa si falla el deshacer?

```
async function registrarSube(cuenta, monto){
    try {
        var linkId = dbLink.command(
            "INSERT INTO servicios (servicio, cuenta, monto) VALUES ($1, $2, $3) RETURNING id",
            ["CARGA SUBE", cuenta, monto]
        ).fetchOneRow();
        try {
            dbSube.command(
                "UPDATE saldos SET saldo = saldo - $2 WHERE cuenta = $1",
                [cuenta, monto]
            ).execute();
        } catch {
            dbSube.command(
                "UPDATE saldos SET saldo = saldo + $2 WHERE cuenta = $1",
                [cuenta, monto]
            ).execute();
        }
    } catch {
        dbLink.command(
            "DELETE FROM SERVICIOS WHERE id = $1",
            [linkId]
        ).execute();
    }
}
```