BDII 2 – SEGURIDAD

AUTORIZACIÓN

- Los PRIVILEGIOS son los diferentes tipos de **autorización** para diferentes **partes de la bd** que se le pueden asignar a los usuarios.
  Cuando un usuario envía una sentencia, primero se comprueba si tiene los permisos necesarios para ejecutarla. De no ser así, se **rechaza**.
- El usuario que **crea** un objeto recibe de manera automática **todos los privilegios** sobre él.
- El **administrador** del sistema tiene **todos los privilegios posibles.**

– Autorizaciones sobre **datos**: { **lectura** de datos $(select)$ | **inserción** de datos $(insert\ into)$ | **actualización** de datos $(update)$ | **borrado** de datos $(delete)$ }.

– Autorizaciones sobre **esquemas**: { **creación** de esquemas $(create)$ | **modificación** de esquemas $(alter)$. | **borrado** de esquemas $(drop)$ }

CONCESIÓN Y REVOCACIÓN DE PRIVILEGIOS

- La instrucción $grant$ se usa para **conceder** un conjunto de privilegios a un conjunto de usuarios o roles.

$$\boldsymbol{grant}\ autorización_1, autorización_2\ ....$$
$$\boldsymbol{on}\ relacion/vista$$
$$\boldsymbol{to}\ usuario/rol_1, usuario/rol_2, ...$$

  - La autorización $all\ privileges$ reúne **todos los privilegios que se pueden conceder**.
  - La autorización $select$ permite **leer** tuplas de la relación (realizar consultas) → $\boldsymbol{grant}\ select$ ...
  - La autorización $update$ permite **actualizar** tuplas de la relación → $\boldsymbol{grant}\ update\ [(a_1, a_2, ...)]$ ...
    ↳ Puede concederse sobre todos los atributos de la relación o sobre algunos. Si no se especifica, se concede sobre todos.
  - La autorización $insert$ permite **insertar** tuplas en la relación → $\boldsymbol{grant}\ insert\ [(a_1, a_2, ...)]$ ...
    ↳ Puede concederse sobre todos los atributos de la relación o sobre algunos. Si no se especifica, se concede sobre todos.
    Si se especifica, las tuplas insertadas tendrán el resto de atributos con valores **predeterminados** o **nulos**.
  - La autorización $delete$ permite **borrar** tuplas de la relación → $\boldsymbol{grant}\ delete$ ...
  - La autorización $execute$ permite **ejecutar** una función o procedimiento → $\boldsymbol{grant}\ execute$ ...
  - La autorización $references$ permite **declarar claves** externas o predicados de $\boldsymbol{check}$ → $\boldsymbol{grant}\ references\ [(a_1, a_2, ...)]$ ...
    ↳ Puede concederse sobre todos los atributos de la relación o sobre algunos. Si no se especifica, se concede sobre todos.

- La instrucción $revoke$ se usa para **retirar** un conjunto de privilegios a un conjunto de usuarios o roles.

$$\boldsymbol{revoke}\ autorización_1, autorización_2\ ....$$
$$\boldsymbol{on}\ relacion/vista$$
$$\boldsymbol{to}\ usuario/rol_1, usuario/rol_2, ...$$

ROLES

- Los ROLES agrupan usuarios para poder concederles y revocarles autorizaciones a todos a la vez.
  ↳ El rol $public$ agrupa a todos los usuarios actuales y futuros del sistema.

$$\boldsymbol{create\ role/group/user}\ nombre\_rol;$$
$$\boldsymbol{alter\ role/group/user}\ nombre\_rol;$$
$$\boldsymbol{drop\ role/group/user}\ nombre\_rol;$$

- Todas las autorizaciones que se concedan a un usuario se pueden conceder a un rol.
- Los roles se conceden a los usuarios igual que las autorizaciones → $\boldsymbol{grant}\ rol_1, rol_2, ...\ \boldsymbol{to}\ usuario_1, usuario_2, ...$
- Se pueden conceder roles a otros roles → $\boldsymbol{grant}\ rol_1, rol_2, ...\ \boldsymbol{to}\ rol_1, rol_2, ...$

▸ Por tanto, los privilegios de un usuario o rol serán { todos los que se concedan directamente al usuario o rol. | todos los que se concedan a roles que se han concedido al usuario o rol. }

AUTORIZACIÓN SOBRE VISTAS Y FUNCIONES

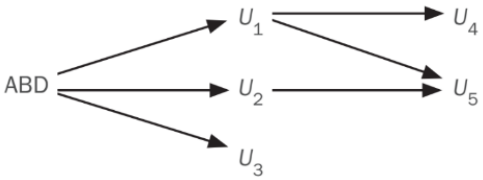
- Las autorizaciones sobre **vistas** se deben conceder como si fueran una relación normal.
- Un usuario que crea una **vista** no recibe necesariamente todos los privilegios sobre ella, sólo aquellos que no proporcionan autorización adicional fuera de la que ya tiene.
  ↳ Si un usuario intenta crear una vista sobre la que no se le puede conceder ninguna autorización, el sistema rechazará la creación.
- Las **funciones** y **procedimientos** también tienen los privilegios que tiene su creador, es decir, se ejecutan como si lo invocase el creador.
  ↳ Si la definición de la función tiene la cláusula $\boldsymbol{sql\ security\ invoker}$ se ejecuta con los privilegios del usuario que la invoca.

AUTORIZACIÓN SOBRE ESQUEMAS

- Sólo el propietario del esquema de la bd puede realizar modificaciones sobre él (crear o borrar tablas, crear o eliminar atributos y crear o añadir índices).

TRANSFERENCIA DE PRIVILEGIOS

- A un usuario puede permitírsele **trasladar una autorización a otros usuarios** añadiendo la cláusula $\boldsymbol{with\ grant\ option}$ al final de la instrucción $grant$ correspondiente para esa autorización.
  - ↳ El **creador** de un objeto mantiene todos los privilegios sobre este, incluyendo el de conceder privilegios a otros.
  - ↳ El permiso para trasladar una autorización se puede revocar con $\boldsymbol{revoke}\ grant\ option$ ...
- El paso de una determinada autorización de un usuario a otro se representa mediante un **grafo de autorización** para ese privilegio.
  - El administrador de la bd es la raíz.
  - Los usuarios son nodos.
  - Habrá una arista $U_i \rightarrow U_j$ si el usuario U_i le concedió la autorización al U_j.
  - ▸ Un usuario cualquiera tendrá esa autorización si existe un camino desde la raíz hasta el nodo que lo representa.



REVOCACIÓN DE PRIVILEGIOS CONCEDIDOS POR OTRO USUARIO

- La revocación de un privilegio a un usuario o rol implica que este se **elimina del grafo de autorización**.
- Esto provoca una **revocación en cascada**, pues los usuarios cuyo único predecesor directo en el grafo era este usuario también perderán el privilegio.
  - ↳ Se puede evitar añadiendo la cláusula $restrict$ al final de la instrucción $revoke$ correspondiente, de manera que las revocaciones que provocarían una revocación en cascada serán rechazadas.
- Restringir la revocación en cascada a veces no es apropiado. Por esto, SQL tiene la noción de **rol actual asociado a una sesión**.
  - Por defecto, el rol actual asociado a una sesión es $null$, pero se puede cambiar con $\boldsymbol{set\ role}\ nombre\_rol$.
    - ↳ Si $nombre\_rol$ no es uno de los roles concedidos al usuario, la sentencia será rechazada.
  - Así, se pueden conceder privilegios en nombre del rol actual asociado a una sesión añadiendo la cláusula $\boldsymbol{granted\ by\ current\_role}$ al final de la instrucción $grant$ correspondiente.

SEGURIDAD DE LAS APLICACIONES

- Hay muchas formas en que se puede ver comprometida la seguridad de una aplicación, aunque el **sgbd** sea **seguro**, debido a un **código mal escrito**.
- La SEGURIDAD DE LAS APLICACIONES tiene que tratar con distintos temas que van más allá de los que manejan las autorizaciones SQL.

INYECCIÓN SQL

- La INYECCIÓN SQL es una fuga de seguridad en la que el atacante consigue que una aplicación ejecute una consulta SQL creada por él debido a que en ella se **concatena la entrada del usuario con cadenas SQL**.
- Estas sentencias pueden causar un daño importante pues pueden realizar cualquier acción en la bd **ignorando** todas las **medidas de seguridad del código**.

Para evitar estos ataques se puede:

- Usar **sentencias preparadas**, en las que JDBC añade automáticamente caracteres de escape a los *inputs* de los usuarios.
- Crear **funciones** que incluyan manualmente **caracteres de escape** en los *inputs* antes de concatenar la cadena con la consulta SQL.
- Comprobar que los **valores** que introduce el usuario son los **esperados** (por ejemplo, que coinciden con un atributo de la relación).

FUGA DE CONTRASEÑAS

- La FUGA DE CONTRASEÑAS sucede cuando se almacenan las contraseñas en **texto claro en el código de la aplicación**, de forma que, si este código se guarda en un directorio accesible por el servidor web, un usuario externo puede acceder a él y conseguir las contraseñas de la bd.

Para evitar este problema se puede:

- **Cifrar** las contraseñas antes de almacenarlas en el servidor y que este las descifre antes de pasárselas a la bd.
  - ↳ Las contraseñas seguirán siendo vulnerables si se descubre la clave de cifrado.
- Restringir el acceso a la bd a ciertas direcciones **IP** conocidas.

AUTENTICACIÓN DE LAS APLICACIONES

- La AUTENTICACIÓN se refiere a la tarea de **verificar la identidad** de una persona o software que se conecta a una aplicación.
- La autenticación más simple es la **contraseña**, pero estas se comprometen fácilmente (adivinándolas o leyéndolas de paquetes de red no cifrados).
- Una alternativa es la **autenticación en dos pasos**, que consiste en usar dos piezas de información o procesos para identificar al usuario.
  - ↳ Ambos factores no deberían compartir una vulnerabilidad.

1. El primer paso suele ser una **contraseña**.
2. El segundo paso puede ser
   - medidas **biométricas** (huella dactilar, escáneres de cara, etc.).
   - **tarjetas inteligentes** u otros dispositivos cifrados.
   - dispositivos de **contraseñas de un sólo uso** que generen un nuevo número aleatorio cada cierto tiempo.
   - envío de un **SMS** con un número aleatorio.

- Incluso así los usuarios siguen siendo vulnerables a un ataque de **hombre en el medio**, en el que un usuario que se intenta conectar a la aplicación es desviado a una web falsa, que acepta la contraseña (incluyendo la del 2º paso) y la usa inmediatamente para autenticarse en la aplicación original.
  - ↳ Para solucionar esto, HTTPS autentica a los usuarios de una web y cifra los datos.

SERVICIOS DE AUTENTICACIÓN CENTRALIZADOS

- Identificarse por separado en todas las webs a las que se accede es muy tedioso. Los SERVICIOS DE AUTENTICACIÓN CENTRALIZADOS como LDAP permiten que el usuario se autentique en un servicio central, de manera que esta autenticación le sirva para acceder a varias webs.
  - ↳ El servicio central de autenticación también puede proporcionar información del usuario (correo, dirección, teléfono, etc.) a la aplicación.
- El **SAML** (lenguaje de marcado para confirmaciones de seguridad) es un estándar para intercambiar información de autenticación y autorización que proporciona acceso de firma única entre organizaciones.
- La norma **OpenID** es una forma alternativa de firma única entre organizaciones en el que sitios web (Google, Microsoft, etc.). actúan como proveedores de autenticación permitiendo que cualquier aplicación cliente OpenID los pueda usar para autenticar un usuario.

AUTORIZACIÓN A NIVEL DE APLICACIÓN

- El modelo de **autorización de SQL es muy limitado** para gestionar las autorizaciones de usuario en una aplicación típica, debido a:
  - Falta de **información sobre el usuario final** ⟶ los usuarios finales normalmente no tienen identificadores de usuario individuales en la bd.
  - Falta de **autorización de grado fino** ⟶ la autorización no se puede realizar a nivel de las tuplas individuales.
    - ↳ Se puede intentar solucionar creando vistas para cada usuario, pero eso sería muy costoso.
- Por tanto, la autenticación normalmente **se realiza a nivel de aplicación**, lo cual aporta **flexibilidad** a los desarrolladores, pero tiene ciertos problemas:
  - × El código de la autorización se **mezcla** con el resto del código de la aplicación, dificultando su comprensión.
  - × Es difícil asegurar que no aparezcan **agujeros de seguridad** ⟶ una aplicación con problemas puede comprometer la seguridad de todas, pero verificar que todos los programas de aplicación realicen todas las comprobaciones necesarias implica un gran esfuerzo.
- Si la bd dispone de **autorizaciones de grado fino**, las políticas de autorización se pueden especificar y forzar a nivel SQL, especificando las políticas de autorización de forma declarativa, lo cual lleva a menos errores.
  - ▸ La bd privada virtual (VPD) de Oracle proporcionan estas autorizaciones permitiendo que el administrador del sistema asocie a cada relación una función que devuelva un predicado que se añade a la cláusula $where$ de todas las consultas la usen (también sirve para actualizaciones).

TRAZAS DE AUDITORÍA

- ○ Una TRAZA DE AUTIDORÍA es un **registro** de todos los cambios, inserciones, borrados y actualizaciones en los datos de la aplicación, junto con la información sobre **quién** y **cuándo** los realizó.
  - Si se **rompe la seguridad** de la app o se realiza alguna **actualización errónea**, puede ayudar a:
    - Descubrir **qué ha pasado**
    - **Reparar el daño**.
  - También se pueden usar para **detectar brechas de seguridad** cuando una cuenta de usuario se ve comprometida y un intruso accede usándola.
    - ↳ Por ejemplo, cuando un usuario inicia sesión se le puede informar del contenido de su traza para que detecte actualizaciones que no hizo.
- – Se pueden **crear** definiendo **disparadores** sobre las actualizaciones que usen variables del sistema (como $current\_user$ o $current\_time$).
- – En otros casos, los propios sgbd proporcionan **mecanismos predeterminados** más cómodos.
- Las trazas de auditoría a **nivel bd** suelen ser **insuficientes** pues:
  - Normalmente no son capaces de identificar al **usuario final** de la aplicación.
  - Las actualizaciones que almacena se realizan a **nivel bajo** (tuplas) en lugar de en términos de la lógica del negocio
- ▸ Por tanto, **las aplicaciones** normalmente **crean una traza de auditoría de alto nivel**.

PRIVACIDAD

- ○ La PRIVACIDAD DE LOS DATOS supone una preocupación cada vez mayor conforme la cantidad de datos personales accesibles online va creciendo.
- La mayoría de países tienen **leyes** sobre la privacidad de datos, que definen cuándo y quién puede acceder a ellos, por lo que las aplicaciones que acceden a datos privados se deben construir con sumo **cuidado**.
- Algunos datos privados agregados pueden ser interesantes para la **investigación**, por lo que se debe intentar que estén disponibles para los investigadores sin comprometer la privacidad de las personas.
- Los **sitios web** suelen recopilar datos personales de sus usuarios, por lo que le deben permitir a los clientes indicar sus preferencias de privacidad y asegurarse de que se respetan.