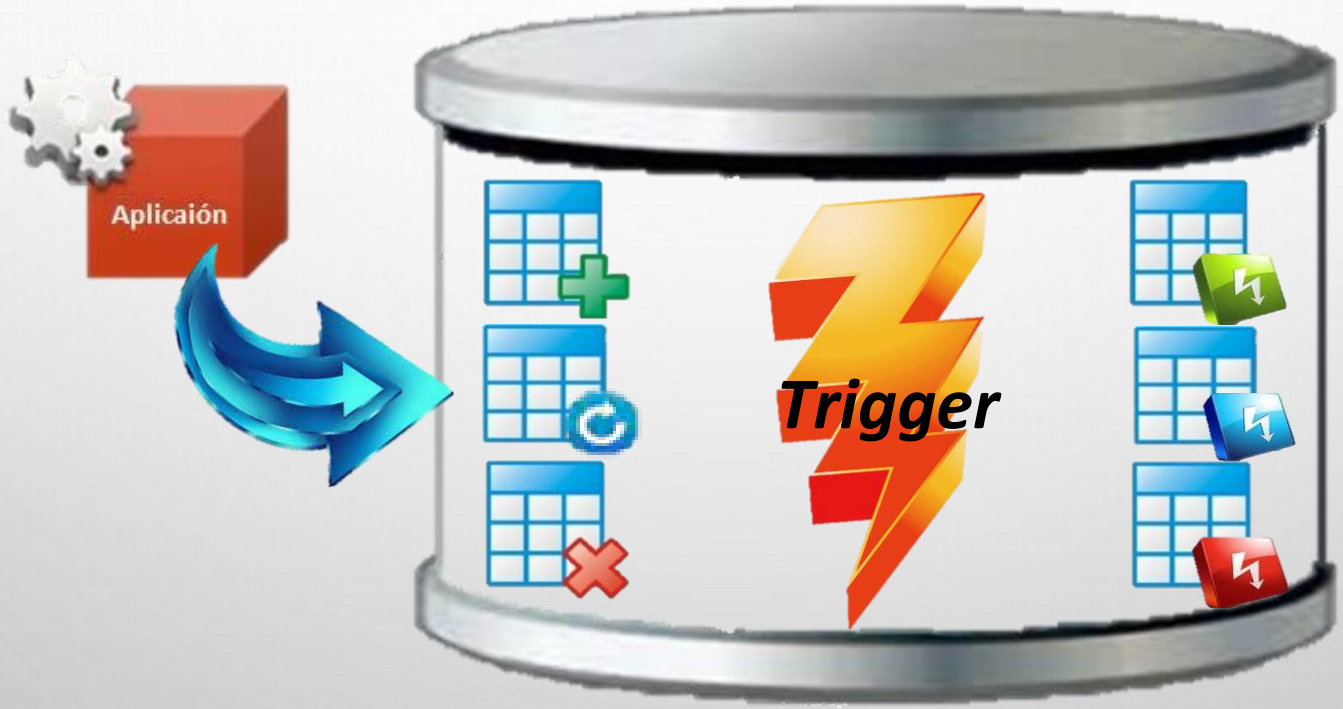
Conceptos de Bases de Datos II

Triggers

Triggers



Triggers - Auditoria

Auditoria de Base de Datos: Es el proceso que permite medir, asegurar, demostrar, monitorear y registrar los accesos a la información almacenada en las bases de datos incluyendo la capacidad de determinar:

- Quién accede a los datos.
- Cuando se accedió a los datos.
- Desde qué tipo de dispositivo/aplicación.
- Desde que ubicación en la Red.
- Cuál fue la sentencia SQL ejecutada.
- Cuál fue el efecto del acceso a la base de datos.

Auditoria - Objetivos

Disponer de mecanismos que permitan capturar de una auditoría la relación del personal con el acceso a las bases de datos incluyendo la capacidad de generar, modificar y/o eliminara datos.

Mitigar los riesgos asociados con el manejo inadecuado de los datos.

Evitar acciones criminales.

Evitar multas por incumplimiento.

Auditoria - Importancia

La Auditoría de BD es importante porque:

- Toda la información financiera de la organización reside en bases de datos y deben existir controles relacionados con el acceso a las mismas.
- Se debe poder demostrar la integridad de la información almacenada en las bases de datos.
- Las organizaciones deben mitigar los riesgos asociados a la pérdida de datos y a la fuga de información.
- La información confidencial de los clientes, son responsabilidad de las organizaciones.
- Las organizaciones deben tomar medidas mucho más allá de asegurar sus datos.

Auditoria - Seguridad

Mediante la auditoría de bases de datos se evaluará:

- Definición de estructuras físicas y lógicas de las bases de datos.
- Control de carga y mantenimiento de las bases de datos.
- Integridad de los datos y protección de accesos.
- Estándares para análisis y programación en el uso de bases de datos.
- Procedimientos de respaldo y de recuperación de datos.

Auditoria - Seguridad

Proteger los datos de accesos no autorizados

Quién puede estar interesado en atacar el sistema?

- Un antiguo empleado recientemente despedido.
- Alguien externo a la organización.
- Un usuario del sistema que trata de obtener mayores privilegios.
- Un hacker profesional con un propósito específico.

Auditoria - Tipos de Ataques

Desbordamiento de memoria (buffer overflow).
Se basa en pasar como entrada a un programa más datos de los que espera recibir.

Un desbordamiento de búfer (del inglés buffer overflow) es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer)

Si dicha cantidad supera la capacidad pre-asignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes. Esto constituye un fallo de programación

Inyección de Código SQL.
Consiste en **incluir/modificar** comando SQL al enviarlos al servidor.

Se conoce como Inyección SQL, indistintamente, al tipo de vulnerabilidad, al método de infiltración, al hecho de incrustar código SQL intruso y a la porción de código incrustado.

Auditoria

Para evitar posibles ataques deben realizarse auditorías.

Una auditoría es una revisión de los permisos de cada usuario y de los archivos del sistema con el propósito de detectar agujeros de seguridad.

Una auditoría completa no es un conjunto estricto de validaciones a realizar, sino que evoluciona según los riesgos detectados.

Es una tarea ardua, consume mucho tiempo y no asegura al 100% que el sistema está “limpio”.
La recopilación de información de auditoría puede centrarse en distintos elementos:

Sentencias SQL: Registro de los intentos de conexión con la base de datos.

Privilegios: Consiste en recopilar las operaciones que se han efectuado sobre la base de datos (inserciones, borrados, modificaciones, etc.) y por qué usuarios.

Objetos: Se pueden recoger información de las operaciones realizadas sobre algunos objetos determinados de la base de datos.

Metodologías para auditoría

Metodología Tradicional

El auditor revisa el entorno con la ayuda de una lista de control (Checklist), que consta de una serie de cuestiones a verificar, registrando los resultados de su investigación. En esta investigación se confecciona una lista de control de todos los aspectos a tener en cuenta.

Metodología de evaluación de riesgos

Este tipo de metodología, conocida también como Enfoque orientado al riesgo (Risk oriented approach), comienza fijando los objetivos de control que minimizan los riesgos potenciales a los que está sometido el entorno.

Considerando los riesgos de:

✓ Dependencia por la concentración de Datos
✓ Accesos no restringidos en la figura del DBA

Metodologías para auditoría

✓ Incompatibilidades entre el sistema de seguridad de accesos del SGBD y el general de instalación

✓ Impactos de los errores en Datos y programas

✓ Rupturas de enlaces o cadenas por fallos del software.

✓ Impactos por accesos no autorizados

✓ Dependencias de las personas con alto conocimiento técnico.

Planificación de Auditoria de BD

1. Identificar todas las bases de datos de la organización
2. Clasificar los niveles de riesgo de los datos en las bases de datos
3. Analizar los permisos de acceso
4. Analizar los controles existentes de acceso a las bases de datos
5. Establecer los modelos de auditoría de BD a utilizar
6. Establecer las pruebas a realizar para cada BD, aplicación y/o usuario

Tipos de Auditorias

Existen tres tipos de auditoria.

- Por aplicación
- Por triggers
- Y la combinación de ambas

Auditorias por Aplicación

Paso 1: Crear la tabla auditoria con la siguiente cabecera:

```
create table aud_nombretabla(
Id  serial not null,
FechaAud date not null,
UsuarioAud varchar(30) not null,
IP varchar(40) not null,
UserAgent varchar(255) null,
Aplicacion varchar(50) not null,
Motivo varchar(100) null,
TipoAud varchar(10) not null,
```

Agregar todos los campos de la tabla auditada

Auditorias por Aplicación

Paso 2: Identificar SP's. Agregar a la **entrada si no tiene:**

pIP **varchar**(40),
pUserAgent **varchar**(255),
pAplicacion **varchar** (50)

Si al motivo lo tiene que agregar el usuario, agregar:
pMotivo **varchar** (100)

La tabla aud_nombretabla no debe tener **CONSTRAINT,** solo la clave primaria (tipo serial) y si la tabla a ser auditada tiene índices, la tabla auditoria también debe tenerlos

Auditorias por Aplicación

En caso de INSERT dentro del sp…

```
declare
tabla MiTabla%rowtype;

begin
pUsuario:= (select Usuario from Usuarios where IdUsuario=pIdUsuario);


for tabla in select * from MiTabla where idMiTabla = pId loop
   insert into aud_MiTabla values
   (default, now(), pUsuario, pIP, pUserAgent, pAplicacion, null, 'INSERT', tabla.*)
 …
```

Auditorias por Aplicación

En caso de DELETE dentro del sp…

```
declare

tabla MiTabla%rowtype;

begin
pUsuario:= (select Usuario from Usuarios where IdUsuario=pIdUsuario);

for tabla in select * from MiTabla where idMiTabla = pId loop

insert into aud_MiTabla values
 (default, now(), pUsuario, pIP, pUserAgent, pAplicacion, null, 'DELETE', tabla.*)

delete from MiTabla where idMiTabla = pParametro
...
```

Auditorias por Aplicación

En caso de UPDATE dentro del sp…

```
declare
tabla MiTabla%rowtype;


begin
pUsuario:= (select Usuario from Usuarios where IdUsuario=pIdUsuario);

for tabla in select * from MiTabla where Parametro = pParametro loop

 -- Aud Antes
      insert into aud_MiTabla values (default, now(), pUsuario, pIP, pUserAgent, pAplicacion, null, 'ANTES', tabla.*)

  -- Modifico parámetro
update MiTabla set Valor = pValor where Parametro = pParametro;

 -- Aud Después
for tabla in select * from MiTabla where Parametro = pParametro loop
 insert into aud_MiTabla values (default, NOW(), pUsuario, pIP, pUserAgent, pAplicacion, null, 'DESPUES', tabla.*)
...
```

Auditorias por Triggers

Se debe crear la tabla aud_nombretabla, ésta debe tener todos los campos de la tabla a auditar**, no debe tener constraint**, solo la clave primaria (tipo serial) y si la tabla a ser auditada tiene índices, la tabla auditoria también debe tenerlos

Supongamos que la tabla a ser auditada sea la de empleados

```
create table audita_emp    (Id serial not null,
                            FechaAud date not null,
                            Usuario varchar(50) not null,
                            Operacion varchar(10) not null,
                            Id_Emp integer not null,
                            Nom_Emp varchar(50) not null,
                            Sueldo numeric(8,2));
```

Auditorias por Triggers

```
create or replace function audita_emp() returns trigger as $audita_emp$
begin
if (tg_op = 'delete')  then
 insert into audita_emp values (default, now(), user, 'delete', old. Id_empleado,
   (select concat(apellido, nombre) from persona where  id_persona = old.id_empleado), old.sueldo);
  return old;
elsif (tg_op = 'insert')  then
   insert into audita_emp values (default, now(), user, 'insert', new.Id_empleado,
  (select concat(apellido, nombre) from persona where id_persona = new.id_empleado),new.sueldo);
 return new;
elsif (tg_op = 'update') then
 insert into audita_emp values (default, now(), user, 'antes', old. Id_empleado,
   (select concat(apellido, nombre) from persona where  id_persona = old.id_empleado), old.sueldo);
 insert into audita_emp values (default, now(), user, 'despues', new.Id_empleado,
  (select concat(apellido, nombre) from persona where id_persona = new.id_empleado),new.sueldo);
 return new; --  return old;  en este caso se puede retornar cualquiera de las dos variables (solo una, no ambas)
endif;
end;
$audita_emp$ language plpgsql;
```

Auditorias por Triggers

```
create trigger audita_emp
after insert or  delete or update on empleado – se dispara con cualquier modificación
for each row execute procedure audita_emp();
```

Para que solo se dispare cuando se modifique el sueldo y ademas sea diferente

```
create trigger audita_emp
after insert or  delete or update on empleado
for each row when(old.sueldo < new.sueldo) execute procedure audita_emp();
```

Data Output | Messages | Notifications

```
ERROR:  la condición WHEN de un "trigger" en INSERT no puede referirse a valores OLD
LINE 4: for each row when(old.sueldo < new.sueldo)
                          ^
```

Auditorias por Triggers

```
create trigger audita_emp
 after insert or  delete on empleado
for each row execute procedure audita_emp();

create trigger audita_emp_updte
 after update on empleado
for each row when (old.sueldo < new.sueldo)
execute procedure audita_emp();

-- prueba
insert into empleado values (999,1,1,'2025-01-01',150000.23,null)

update empleado set sueldo = 302512 where id_empleado = 999

update empleado set fecha_baja = '2025-12-30' where id_empleado = 999

delete from empleado where id_empleado = 999

select * from aud_empleado
```

Auditorias por Triggers

Consulta para saber en que tablas es PK

```
select tc.table_name as tabla, kcu.column_name as columna
from  information_schema.table_constraints as tc -- vista que contiene las constraint definida en una tabla.
inner join   information_schema.key_column_usage AS kcu --detalle de las columnas involucradas en las constraint.
on tc.constraint_name = kcu.constraint_name
where   tc.constraint_type = 'PRIMARY KEY' --Solo queremos constraints que sean claves primarias.
  and kcu.column_name = 'id_empleado'; -- muestra donde id_empleado forma parte de esa clave primaria.
```

Consulta para saber en que tablas es fk obligatoria

```
select  kcu.table_name as tabla, kcu.column_name as columna, ccu.table_name as tabla_referenciada,
      ccu.column_name as columna_referenciada, cols.is_nullable
from   information_schema.table_constraints as tc
join  information_schema.key_column_usage as kcu on tc.constraint_name = kcu.constraint_name
join   information_schema.constraint_column_usage as ccu  on ccu.constraint_name = tc.constraint_name
join information_schema.columns as cols
on kcu.table_name = cols.table_name and kcu.column_name = cols.column_name
where   tc.constraint_type = 'FOREIGN KEY'   and kcu.column_name = 'id_empleado';
```

Auditorias por Triggers

```
create or replace function borra_empleado()  returns trigger as $borra_empleado$

begin

  delete from  diagnostico where id_empleado = old.id_empleado;
  delete from  compra where id_empleado = old.id_empleado;
  delete from  consulta where id_empleado = old.id_empleado;
  delete from  mantenimiento_equipo where id_empleado = old.id_empleado;
  delete from  estudio_realizado where id_empleado = old.id_empleado;
  delete from  trabajan where id_empleado = old.id_empleado;

  return old;
end;
$borra_empleado$ language plpgsql;

create trigger borra_empleado
before delete on empleado for each row
execute procedure borra_empleado();
```

Auditorias por Triggers

```
select  event_object_table as tabla,
        trigger_name,
        action_timing as momento,     -- BEFORE o AFTER
        event_manipulation as evento, -- INSERT, UPDATE, DELETE, etc.
        action_orientation as nivel,  -- ROW o STATEMENT
        action_statement as funcion
from  information_schema.triggers
where   event_object_table = 'empleado';
```

Data Output | Messages | Notifications

	tabla name	trigger_name name	momento character varying	evento character varying	nivel character varying	funcion character varying
1	empleado	borra_empleado	BEFORE	DELETE	ROW	EXECUTE FUNCTION borra_empleado()
2	empleado	audita_emp	AFTER	INSERT	ROW	EXECUTE FUNCTION audita_emp()
3	empleado	audita_emp	AFTER	DELETE	ROW	EXECUTE FUNCTION audita_emp()

Auditorias por Triggers

Auditar las tablas que tienen valores, por ejemplo, precio, costo, sueldo

```
create or replace function fn_audita_valores() returns trigger as $$
begin
if (tg_table_name = 'tratamiento')  then
  insert into aud_valores values (default, now(), user, tg_table_name, 'costo', old.costo);
  insert into aud_valores values (default, now(), user, tg_table_name, 'costo', new.costo);
   return old;
 elsif (tg_table_name = 'internacion')  then
   insert into aud_valores values (default, now(), user, tg_table_name, 'costo', old.costo);
  insert into aud_valores values (default, now(), user, tg_table_name, 'costo', new.costo);
   return new;
   elsif (tg_table_name = 'estudio_realizado') then
       insert into aud_valores values (default, now(), user, tg_table_name, 'precio', old.precio);
  insert into aud_valores values (default, now(), user, tg_table_name, 'precio', new.precio);
       return new;
end if;
end;
$$ language plpgsql;
```

Auditorias por Triggers

```
create table   aud_valores(Id serial not null,
                           fechaAud date not null,
                           usuario varchar(50) not null,
                           tabla varchar(20) not null,
                           campo varchar(20) not null,
                            valor numeric(9,2));

create trigger audita_tratamiento
after update on tratamiento
for each row when(old.costo < new.costo) execute procedure fn_audita_valores();

 create trigger audita_internacion
after update on internacion
for each row when(old.costo < new.costo)  execute procedure fn_audita_valores();

 create trigger audita_estudio_realizado
after update on estudio_realizado
for each row when(old.precio < new.precio) execute procedure fn_audita_valores();
```

Auditorias por Triggers

```
update tratamiento set costo = 100000
where id_paciente = 1000
and id_medicamento = 10
and fecha_indicacion = '2025-06-03'

update internacion set costo = 1000000
where id_paciente = 1147
and id_cama = 107
and fecha_inicio = '2022-06-26'

update estudio_realizado set precio = 5500
where id_paciente = 87752
and id_estudio = 108
and fecha = '2022-03-28’


select * from aud_valores
```

Auditorias por Triggers

```
create table aud_eventos_criticos (id serial primary key,
                                   fecha timestamp,
                                     usuario text,
                                     tabla_afectada text,
                                     operacion text,
                                     comentario text);

create or replace function auditar_truncate() returns trigger as $$
begin

 insert into aud_eventos_criticos values (default, now(), user, tg_table_name, tg_op,
                                          concat('ejecutado ',tg_op,' sobre una tabla ',tg_table_name));

  return null;
end;
$audita_emp$ language plpgsql;
```

Auditorias por Triggers

```
create trigger tr_audita_eventos_criticos after  truncate or insert  on factura
for each statement execute function fn_auditar_eventos_criticos();

create trigger tr_audita_eventos_criticos after  truncate or insert  on pago
for each statement execute porcedure fn_auditar_eventos_criticos();

create table factura1 as  select * from facturacion.factura;
create table pago1  as select * from pago;


truncate table pago1
select * from pago1
truncate table factura1
select * from factura1

select * from aud_eventos_criticos

insert into factura1   select * from facturacion.factura;
insert into pago1   select * from pago;
```

Triggers – Observaciones

Cuando se diseñan disparadores es necesario tomar en cuenta las siguientes consideraciones:

No se debe usar para hacer cumplir restricciones de integridad que puedan ser definidas a nivel de esquema. Por ejemplo, verificar al insertar una tupla en la tabla Empleado que su fecha de nacimiento no sea mayor a la fecha actual. Esta restricción puede definirse mediante CHECK (fecha_nac < NOW())

Hay que evitar crear disparadores recursivos. Por ejemplo, el crear un disparador que se active después de actualizar la tabla Empleado, que a su vez realiza una actualización de la tabla Empleado, provoca una ejecución recursiva del disparador que agota la memoria.

Triggers – Observaciones

Características y reglas más importantes cuando se define un trigger y/ó una función que se vaya a utilizar por un trigger:

1. El procedimiento almacenado que se vaya a utilizar por el trigger debe de definirse e instalarse antes de definir el propio disparador.

2. Un procedimiento que se vaya a utilizar por un disparador no puede tener argumentos y tiene que devolver el tipo "trigger".

3. Un mismo procedimiento almacenado se puede utilizar por múltiples disparadores en diferentes tablas.

4. Procedimientos almacenados utilizados por disparadores que se ejecutan una sola vez per comando SQL (statement-level) tienen que devolver siempre NULL.

5. Procedimientos almacenados utilizados por disparadores que se ejecutan una vez por línea afectada por el comando SQL (row-level) pueden devolver una fila de tabla.

Triggers – Observaciones

6. Procedimientos almacenados utilizados por disparadores que se ejecutan una vez per fila afectada por el comando SQL (row-level) y ANTES de ejecutar el comando SQL que lo lanzó, pueden:

- Retornar NULL para saltarse la operación en la fila afectada.
- Ó devolver una fila de tabla (RECORD)

7. Procedimientos almacenados utilizados por disparadores que se ejecutan DESPUES de ejecutar el comando SQL que lo lanzó, ignoran el valor de retorno, así que pueden retornar NULL sin problemas.

8. En resumen, independientemente de como se defina un disparador, el procedimiento almacenado utilizado por dicho disparador tiene que devolver ó bien NULL, ó bien un valor RECORD con la misma estructura que la tabla que lanzó dicho disparador.

9. Si una tabla tiene más de un disparador definido para un mismo evento (INSERT,UPDATE,DELETE), estos se ejecutarán en orden alfabético por el nombre del disparador. En el caso de disparadores del tipo ANTES / rowlevel, la file retornada por cada disparador, se convierte en la entrada del siguiente. Si alguno de ellos retorna NULL, la operación será anulada para la fila afectada.

Triggers – Observaciones

10.Procedimientos almacenados utilizados por disparadores pueden ejecutar sentencias SQL que a su vez pueden activar otros disparadores. Esto se conoce como disparadores en cascada. No existe límite para el número de disparadores que se pueden llamar pero es responsabilidad del programador el evitar una recursión infinita de llamadas en la que un disparador se llame asi mismo de manera recursiva.

Otra cosa que tenemos que tener en cuenta es que, por cada trigger que definamos en una tabla, nuestra base de datos tendrá que ejecutar la función asociada a dicho trigger. El uso de triggers de manera incorrecta ó inefectiva puede afectar el rendimiento de nuestra base de datos de manera significativa. Los principiantes deberían de usar un tiempo para entender como funcionan y así poder hacer un uso correcto de los mismos antes de usarlos en sistemas en producción

Gracias