

**TRABAJO PRÁCTICO INTEGRADOR**

**Asignatura:** Base de Datos Aplicadas.

**Tema:** Gestión de Proyectos Informáticos.

**Grupo Nº:** 2 (Dos)

**Integrantes:** Imfeld, Facundo Nicolas

Lopez, William Juan Jose

Nasir, Khalil Abdul

Ojeda Nuñez, Pablo Nicolas

Troncoso, Clarise

**Profesores:** Ing. Fantin, Andres Pablo

Ing. Fernandez, Juan Carlos

**Año:** 2020

**Segunda etapa**

**Programabilidad**

**Triggers**

1. Incorporar, a partir de una fundamentación específica en cada caso, al menos tres disparadores en tablas de la base de datos de manera que cada uno cumpla con alguno de los siguientes requerimientos:

1. Control de la integridad, coherencia y/o consistencia de los datos.

2. Actualización automática de datos (puede ser necesario agregar algún campo o tabla

para cumplir el requerimiento).

3. Algún tipo de auditoría (puede ser necesario agregar algún campo o tabla para cumplir el requerimiento).

/\* -- El criterio que usamos para el nombre de los triggers es: -- \*/

/\* -- nombreTabla\_descripcion\_evento\_restriccion -- \*/

/\*

nombreTabla: Bueno, es bastante descriptivo, es el nombre de la tabla

descripcion: Viene siendo el campo que se actualizará o qué cosas se van a validar, en fin, una descripcion de lo que se hará

evento: Seria el insert, update, o delete

restriccion: Serian los SS8, SS9, SS10, SS11, etc etc. Esto con la finalidad de validar la correcta creacion del trigger mas adelante

\*/

/\* -- RESTRICCIÓN SS8 -- \*/

/\* -- SS8 PROYECTO.Horas estimadas <= PROYECTO.Fecha\_fin - PROYECTO. Fecha\_inicio -- \*/

DROP TRIGGER IF EXISTS proyectos\_horas\_estimadas\_insert\_SS8;

DELIMITER //

CREATE TRIGGER proyectos\_horas\_estimadas\_insert\_SS8 BEFORE INSERT ON proyectos FOR EACH ROW

BEGIN

IF(NEW.horas\_estimadas > TIMESTAMPDIFF(HOUR, NEW.fecha\_inicio, NEW.fecha\_fin))THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El campo horas\_estimadas debe ser menor o igual a la diferencia entre fecha\_fin y fecha\_inicio';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS proyectos\_horas\_estimadas\_update\_SS8;

DELIMITER //

CREATE TRIGGER proyectos\_horas\_estimadas\_update\_SS8 BEFORE UPDATE ON proyectos FOR EACH ROW

BEGIN

IF(NEW.horas\_estimadas > TIMESTAMPDIFF(HOUR, NEW.fecha\_inicio, NEW.fecha\_fin))THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El campo horas\_estimadas debe ser menor o igual a la diferencia entre fecha\_fin y fecha\_inicio';

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS10 -- \*/

/\* -- SS10 Comprobar que los intervalos de fechas en la entidad FASE estén incluidos en el intervalo

compuesto por la fecha de inicio y la fecha de fin del proyecto -- \*/

DROP TRIGGER IF EXISTS fases\_rango\_fechas\_insert\_SS10;

DELIMITER //

CREATE TRIGGER fases\_rango\_fechas\_insert\_SS10 BEFORE INSERT ON fases FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

IF(NEW.fecha\_inicio < @fecha\_inicio OR NEW.fecha\_fin > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas de ésta fase, está fuera del rango de fechas del proyecto asociado';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS fases\_rango\_fechas\_update\_SS10;

DELIMITER //

CREATE TRIGGER fases\_rango\_fechas\_update\_SS10 BEFORE UPDATE ON fases FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

IF(NEW.fecha\_inicio < @fecha\_inicio OR NEW.fecha\_fin > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas de ésta fase, está fuera del rango de fechas del proyecto asociado';

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS11 -- \*/

/\* -- SS11. Comprobar que la suma de todas las fechas correspondientes al conjunto de fases de un determinado proyecto, no exceda la fecha de finalización del mismo -- \*/

DROP TRIGGER IF EXISTS fases\_control\_suma\_fechas\_insert\_SS11;

DELIMITER //

CREATE TRIGGER fases\_control\_suma\_fechas\_insert\_SS11 BEFORE INSERT ON fases FOR EACH ROW

BEGIN

SET @cantidad\_horas\_proyecto := (SELECT TIMESTAMPDIFF(HOUR, fecha\_inicio, fecha\_fin) FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

SET @cantidad\_horas\_fases := (

SELECT SUM(TIMESTAMPDIFF(HOUR, fecha\_inicio, fecha\_fin))

FROM fases

WHERE NEW.cod\_proyecto = cod\_proyecto

);

IF(@cantidad\_horas\_fases > @cantidad\_horas\_proyecto)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El numero de horas total de fases excede el numero de horas del proyecto';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS fases\_control\_suma\_fechas\_update\_SS11;

DELIMITER //

CREATE TRIGGER fases\_control\_suma\_fechas\_update\_SS11 BEFORE UPDATE ON fases FOR EACH ROW

BEGIN

IF(NEW.fecha\_inicio != fecha\_inicio OR NEW.fecha\_fin != fecha\_fin)THEN

SET @cantidad\_horas\_proyecto := (SELECT TIMESTAMPDIFF(HOUR, fecha\_inicio, fecha\_fin) FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

SET @cantidad\_horas\_fases := (

SELECT SUM(TIMESTAMPDIFF(HOUR, fecha\_inicio, fecha\_fin))

FROM fases

WHERE NEW.cod\_proyecto = cod\_proyecto

);

IF(@cantidad\_horas\_fases > @cantidad\_horas\_proyecto)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El numero de horas total de fases excede el numero de horas del proyecto';

END IF;

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS12 -- \*/

/\* -- SS12 Dirigido.Num\_horas = PROYECTO.Fecha\_fin - PROYECTO.Fecha\_inicio. -- \*/

/\* -- Considero que el campo num\_horas NO será ingresado por el usuario y será el sistema quien lo ingresará, es decir,

el trigger en este caso -- \*/

DROP TRIGGER IF EXISTS dirigido\_num\_horas\_insert\_SS12;

DELIMITER //

CREATE TRIGGER dirigido\_num\_horas\_insert\_SS12 BEFORE INSERT ON dirigido FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET NEW.num\_horas = TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin);

END//

DELIMITER ;

/\* -- Controla el caso de que por las dudas se intente manipular el numero de horas -- \*/

DROP TRIGGER IF EXISTS dirigido\_num\_horas\_update\_SS12;

DELIMITER //

CREATE TRIGGER dirigido\_num\_horas\_update\_SS12 BEFORE UPDATE ON dirigido FOR EACH ROW

BEGIN

IF (NEW.num\_horas != num\_horas) THEN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET NEW.num\_horas = TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin);

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS proyectos\_num\_horas\_dirigido\_y\_trabaja\_update\_SS12\_SS13;

DELIMITER //

CREATE TRIGGER proyectos\_num\_horas\_dirigido\_y\_trabaja\_update\_SS12\_SS13 AFTER UPDATE ON proyectos FOR EACH ROW

BEGIN

IF (NEW.fecha\_inicio != fecha\_inicio OR NEW.fecha\_fin != fecha\_fin) THEN -- Pregunta antes si cambio algun valor en la fecha, sino no la ejecuta

SET @num\_horas = TIMESTAMPDIFF(HOUR, NEW.fecha\_inicio, NEW.fecha\_fin);

UPDATE dirigido SET num\_horas = @num\_horas WHERE cod\_proyecto = NEW.cod\_proyecto;

UPDATE trabaja SET num\_horas = @num\_horas WHERE cod\_proyecto = NEW.cod\_proyecto;

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS13 -- \*/

/\* -- SS13 Trabaja.Num\_horas = PROYECTO.Fecha\_fin - PROYECTO.Fecha\_inicio. -- \*/

/\* -- Considero que el campo num\_horas NO será ingresado por el usuario y será el sistema quien lo ingresará, es decir,

el trigger en este caso -- \*/

DROP TRIGGER IF EXISTS trabaja\_num\_horas\_insert\_SS13;

DELIMITER //

CREATE TRIGGER trabaja\_num\_horas\_insert\_SS13 BEFORE INSERT ON trabaja FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET NEW.num\_horas = TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin);

END//

DELIMITER ;

/\* -- Controla el caso de que por las dudas se intente manipular el numero de horas -- \*/

DROP TRIGGER IF EXISTS trabaja\_num\_horas\_update\_SS13;

DELIMITER //

CREATE TRIGGER trabaja\_num\_horas\_update\_SS13 BEFORE UPDATE ON trabaja FOR EACH ROW

BEGIN

IF (NEW.num\_horas != num\_horas) THEN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE cod\_proyecto = NEW.cod\_proyecto);

SET NEW.num\_horas = TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin);

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS14 -- \*/

/\* -- SS14 Involucrado.Num\_horas =< FASE.Fecha\_fin - FASE.Fecha\_inicio. -- \*/

DROP TRIGGER IF EXISTS involucrado\_en\_num\_horas\_insert\_SS14;

DELIMITER //

CREATE TRIGGER involucrado\_en\_num\_horas\_insert\_SS14 BEFORE INSERT ON involucrado\_en FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM fases WHERE cod\_fase = NEW.cod\_fase LIMIT 1);

SET @fecha\_fin := (SELECT fecha\_fin FROM fases WHERE cod\_fase = NEW.cod\_fase LIMIT 1) ;

IF(NEW.num\_horas > TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin))THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El campo num\_horas debe ser menor o igual a la diferencia entre fecha\_fin y fecha\_inicio asociada a la fase correspondiente';

END IF;

END//

DELIMITER ;

/\* -- Contemplamos el caso de que intente actualizarse el numero de horas -- \*/

DROP TRIGGER IF EXISTS involucrado\_en\_num\_horas\_update\_SS14;

DELIMITER //

CREATE TRIGGER involucrado\_en\_num\_horas\_update\_SS14 BEFORE UPDATE ON involucrado\_en FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM fases WHERE cod\_fase = NEW.cod\_fase LIMIT 1);

SET @fecha\_fin := (SELECT fecha\_fin FROM fases WHERE cod\_fase = NEW.cod\_fase LIMIT 1) ;

IF(NEW.num\_horas > TIMESTAMPDIFF(HOUR, @fecha\_inicio, @fecha\_fin))THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El campo num\_horas debe ser menor o igual a la diferencia entre fecha\_fin y fecha\_inicio asociada a la fase correspondiente';

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS16 -- \*/

/\* -- SS16 Comprobar que el intervalo de tiempo de la interrelación Se\_asigna está contenido en el

compuesto por las fechas de inicio y de fin de la fase correspondiente -- \*/

DROP TRIGGER IF EXISTS se\_asigna\_intervalo\_tiempo\_insert\_SS16;

DELIMITER //

CREATE TRIGGER se\_asigna\_intervalo\_tiempo\_insert\_SS16 BEFORE INSERT ON se\_asigna FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM fases WHERE NEW.cod\_fase = cod\_fase);

SET @fecha\_fin := (SELECT fecha\_fin FROM fases WHERE NEW.cod\_fase = cod\_fase);

IF(NEW.fecha\_i < @fecha\_inicio OR NEW.fecha\_f > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas debe estar comprendido en el rango de fechas de la fase correspondiente';

END IF;

END//

DELIMITER ;

-- insert into se\_asigna values (NOW(), "2021-11-30", 1, 1, 1, "Nombre");

DROP TRIGGER IF EXISTS se\_asigna\_intervalo\_tiempo\_update\_SS16;

DELIMITER //

CREATE TRIGGER se\_asigna\_intervalo\_tiempo\_update\_SS16 BEFORE UPDATE ON se\_asigna FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM fases WHERE NEW.cod\_fase = cod\_fase);

SET @fecha\_fin := (SELECT fecha\_fin FROM fases WHERE NEW.cod\_fase = cod\_fase);

IF(NEW.fecha\_i < @fecha\_inicio OR NEW.fecha\_f > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas debe estar comprendido en el rango de fechas de la fase correspondiente';

END IF;

END//

DELIMITER ;

/\* -- RESTRICCIÓN SS17 -- \*/

/\* -- SS17 Los periodos de tiempo de la interrelación Se\_asigna no pueden solaparse para el mismo

recurso -- \*/

DROP TRIGGER IF EXISTS se\_asigna\_solapamiento\_fechas\_insert\_SS17;

DELIMITER //

CREATE TRIGGER se\_asigna\_solapamiento\_fechas\_insert\_SS17 BEFORE INSERT ON se\_asigna FOR EACH ROW

BEGIN

SET @cantidad := (SELECT COUNT(\*) FROM se\_asigna WHERE ((NEW.fecha\_i >= fecha\_i AND NEW.fecha\_i <= fecha\_f) OR (NEW.fecha\_f <= fecha\_f AND NEW.fecha\_f >= fecha\_i)) AND NEW.cod\_recurso = cod\_recurso);

IF(@cantidad > 0)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas no debe solaparse para el mismo recurso';

END IF;

END//

DELIMITER ;

/\*

(cod\_recurso, fecha\_i, fecha\_f)

(1, "29-11-2020", "02-12-2020") EJEMPLO

(1, "01-12-2020", "05-12-2020") SE SOLAPA

(1, "20-11-2020", "30-11-2020") SE SOLAPA

\*/

DROP TRIGGER IF EXISTS se\_asigna\_solapamiento\_fechas\_update\_SS17;

DELIMITER //

CREATE TRIGGER se\_asigna\_solapamiento\_fechas\_update\_SS17 BEFORE UPDATE ON se\_asigna FOR EACH ROW

BEGIN

SET @cantidad := (SELECT COUNT(\*) FROM se\_asigna WHERE ((NEW.fecha\_i >= fecha\_i AND NEW.fecha\_i <= fecha\_f) OR (NEW.fecha\_f <= fecha\_f AND NEW.fecha\_f >= fecha\_i)) AND NEW.cod\_recurso = cod\_recurso);

IF(@cantidad > 0)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas no debe solaparse para el mismo recurso';

END IF;

END//

DELIMITER ;

/\* AMBOS ESTAN PENDIENTES DE TESTEO \*/

/\* -- RESTRICCIÓN SS18 -- \*/

/\* -- SS18 Un informático ha de aparecer en la interrelación Trabaja para todos los proyectos en los que

aparece como Involucrado en alguna de sus fases -- \*/

/\* Considero que NO puede cargarse datos en la tabla involucrado\_en, si previamente no se cargaron en trabaja\*/

DROP TRIGGER IF EXISTS involucrado\_en\_controlar\_involucrado\_insert\_SS18;

DELIMITER //

CREATE TRIGGER involucrado\_en\_controlar\_involucrado\_insert\_SS18 BEFORE INSERT ON involucrado\_en FOR EACH ROW

BEGIN

SET @existe := (SELECT TRUE FROM trabaja WHERE NEW.cod\_informatico = cod\_informatico AND NEW.cod\_proyecto = cod\_proyecto);

IF(@existe IS NULL)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El informatico no trabaja en este proyecto';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS involucrado\_en\_controlar\_involucrado\_update\_SS18;

DELIMITER //

CREATE TRIGGER involucrado\_en\_controlar\_involucrado\_update\_SS18 BEFORE UPDATE ON involucrado\_en FOR EACH ROW

BEGIN

IF(NEW.cod\_informatico != OLD.cod\_informatico OR NEW.cod\_proyecto != OLD.cod\_proyecto)THEN

SET @existe := (SELECT TRUE FROM trabaja WHERE NEW.cod\_informatico = cod\_informatico AND NEW.cod\_proyecto = cod\_proyecto);

IF(@existe IS NULL)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El informatico no trabaja en este proyecto';

END IF;

END IF;

END//

DELIMITER ;

/\* AMBOS ESTAN PENDIENTES DE TESTEO \*/

/\* -- RESTRICCIÓN SS20 -- \*/

/\* -- SS20. PROYECTO.Fecha\_inicio =< GASTO.Fecha =< PROYECTO.Fecha\_fin -- \*/

DROP TRIGGER IF EXISTS gastos\_control\_fecha\_insert\_SS20;

DELIMITER //

CREATE TRIGGER gastos\_control\_fecha\_insert\_SS20 BEFORE INSERT ON gastos FOR EACH ROW

BEGIN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

IF(NEW.fecha < @fecha\_inicio OR NEW.fecha > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas debe estar comprendido en el rango de fechas del proyecto correspondiente';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS gastos\_control\_fecha\_update\_SS20;

DELIMITER //

CREATE TRIGGER gastos\_control\_fecha\_update\_SS20 BEFORE UPDATE ON gastos FOR EACH ROW

BEGIN

IF(NEW.fecha != OLD.fecha)THEN

SET @fecha\_inicio := (SELECT fecha\_inicio FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

SET @fecha\_fin := (SELECT fecha\_fin FROM proyectos WHERE NEW.cod\_proyecto = cod\_proyecto);

IF(NEW.fecha < @fecha\_inicio OR NEW.fecha > @fecha\_fin)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El rango de fechas debe estar comprendido en el rango de fechas del proyecto correspondiente';

END IF;

END IF;

END//

DELIMITER ;

/\* AMBOS ESTAN PENDIENTES DE TESTEO \*/

/\* -- RESTRICCIÓN SS21 -- \*/

/\* -- SS21. Controlar que el empleado que realiza el gasto esté trabajando en el proyecto al que se carga dicho gasto -- \*/

/\* -- Consideraciones: será necesario hacer un query sobre la tabla trabaja y dirigido. Ya que en la

tabla trabaja solo estarán los empleados de tipo informatico y en la tabla dirigido estarán

los empleados de tipo jeye de proyectos. Con que se encuentre en cualqueira de estas dos,

será suficiente para validar la transacción --\*/

DROP TRIGGER IF EXISTS gastos\_empleado\_involucrado\_insert\_SS21;

DELIMITER //

CREATE TRIGGER gastos\_empleado\_involucrado\_insert\_SS21 BEFORE INSERT ON gastos FOR EACH ROW

BEGIN

SET @empleado\_trabaja := (SELECT TRUE FROM trabaja WHERE NEW.cod\_empleado = cod\_informatico AND NEW.cod\_proyecto = cod\_proyecto);

SET @empleado\_dirigido := (SELECT TRUE FROM dirigido WHERE NEW.cod\_empleado = cod\_jefe AND NEW.cod\_proyecto = cod\_proyecto);

IF(@empleado\_trabaja IS NULL AND @empleado\_dirigido IS NULL)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El empleado no trabaja en este proyecto';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS gastos\_empleado\_involucrado\_update\_SS21;

DELIMITER //

CREATE TRIGGER gastos\_empleado\_involucrado\_update\_SS21 BEFORE UPDATE ON gastos FOR EACH ROW

BEGIN

IF(NEW.cod\_empleado != OLD.cod\_empleado OR NEW.cod\_proyecto != OLD.cod\_proyecto)THEN

SET @empleado\_trabaja := (SELECT TRUE FROM trabaja WHERE NEW.cod\_empleado = cod\_informatico AND NEW.cod\_proyecto = cod\_proyecto);

SET @empleado\_dirigido := (SELECT TRUE FROM dirigido WHERE NEW.cod\_empleado = cod\_jefe AND NEW.cod\_proyecto = cod\_proyecto);

IF(@empleado\_trabaja IS NULL AND @empleado\_dirigido IS NULL)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El empleado no trabaja en este proyecto';

END IF;

END IF;

END//

DELIMITER ;

/\* AMBOS ESTAN PENDIENTES DE TESTEO \*/

/\* -- RESTRICCIÓN SS22 -- \*/

/\* -- SS22 El presupuesto de un proyecto siempre ha de ser mayor o igual a la suma de todos los costes de sus empleados más los gastos que generan -- \*/

/\* -- TABLA GASTOS -- \*/

DROP TRIGGER IF EXISTS gastos\_control\_suma\_insert\_SS22;

DELIMITER //

CREATE TRIGGER gastos\_control\_suma\_insert\_SS22 BEFORE INSERT ON gastos FOR EACH ROW

BEGIN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS gastos\_control\_suma\_update\_SS22;

DELIMITER //

CREATE TRIGGER gastos\_control\_suma\_update\_SS22 BEFORE UPDATE ON gastos FOR EACH ROW

BEGIN

IF(NEW.monto != OLD.monto OR NEW.cod\_proyecto != OLD.cod\_proyecto)THEN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END IF;

END//

DELIMITER ;

/\* -- TABLA TRABAJA -- \*/

DROP TRIGGER IF EXISTS trabaja\_control\_suma\_insert\_SS22;

DELIMITER //

CREATE TRIGGER trabaja\_control\_suma\_insert\_SS22 BEFORE INSERT ON trabaja FOR EACH ROW

BEGIN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS trabaja\_control\_suma\_update\_SS22;

DELIMITER //

CREATE TRIGGER trabaja\_control\_suma\_update\_SS22 BEFORE UPDATE ON trabaja FOR EACH ROW

BEGIN

IF(NEW.coste\_total != OLD.coste\_total OR NEW.cod\_proyecto != OLD.cod\_proyecto)THEN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END IF;

END//

DELIMITER ;

/\* -- TABLA DIRIGIDO -- \*/

DROP TRIGGER IF EXISTS dirigido\_control\_suma\_insert\_SS22;

DELIMITER //

CREATE TRIGGER dirigido\_control\_suma\_insert\_SS22 BEFORE INSERT ON dirigido FOR EACH ROW

BEGIN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END//

DELIMITER ;

DROP TRIGGER IF EXISTS dirigido\_control\_suma\_update\_SS22;

DELIMITER //

CREATE TRIGGER dirigido\_control\_suma\_update\_SS22 BEFORE UPDATE ON dirigido FOR EACH ROW

BEGIN

IF(NEW.coste\_total != OLD.coste\_total OR NEW.cod\_proyecto != OLD.cod\_proyecto)THEN

SET @presupuesto := (

SELECT presupuesto

FROM proyectos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_gatos := (

SELECT SUM(monto)

FROM gastos

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_trabaja := (

SELECT SUM(coste\_total)

FROM trabaja

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_dirigido := (

SELECT SUM(coste\_total)

FROM dirigido

WHERE NEW.cod\_proyecto = cod\_proyecto

);

SET @suma\_total := (@suma\_gatos + @suma\_trabaja + @suma\_dirigido);

IF(@presupuesto < @suma\_total)THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Con este gasto se excede el presupuesto del proyecto asociado';

END IF;

END IF;

END//

DELIMITER ;

**Programas almacenados**

1. Escribir un procedimiento almacenado que reciba como parámetro algún valor compatible con un dato en la base de datos a partir del cual se emita un informe para ese dato en particular. El informe debe requerir al menos una consulta avanzada de las vistas en cátedra y debe hacer uso de al menos una variable definida en el procedimiento.

#Escriba un procedimiento

#almacenado que para un codigo de proyecto, imprima su nombre, el total de gastos realizados para ese proyecto,

# la fecha en que se realizó

#y el nombre del empleado que hizo ese gasto.

DELIMITER //

CREATE PROCEDURE listar\_proyecto(codigo\_proyecto int)

BEGIN

SET @total\_gastos = (SELECT g.monto FROM gastos g WHERE g.cod\_proyecto = codigo\_proyecto);

SELECT p.nombre,total\_gastos,g.fecha,e.nombre

FROM proyectos p INNER JOIN gastos g ON g.cod\_proyecto = codigo\_proyecto

INNER JOIN empleado e ON g.cod\_empleado = e.cod\_empleado;

END //

DELIMITER ;

2. Escribir una función almacenada que reciba como parámetro algún valor compatible con un dato en la base de datos a partir del cual se calcule un valor resumido para ese dato en particular. En la función se debe realizar al menos una consulta avanzada de las vistas en cátedra y debe hacerse uso de al menos una variable definida localmente.

#Escriba una función que retorne en forma de tupla, para un codigo de informatico,

#la cantidad de proyectos en los que trabajó y la suma de horas.

DELIMITER //

CREATE FUNCTION listar\_proyectos\_y\_horas(codigo\_empleado int) RETURNS VARCHAR(100)

BEGIN

SET @cod\_empleado = (SELECT t.cod\_informatico FROM trabaja t WHERE t.cod\_informatico = codigo\_empleado);

SET @total\_horas = (SELECT SUM(t.num\_horas) FROM trabaja t INNER JOIN empleado e ON e.cod\_empleado = @cod\_empleado AND e.tipo LIKE '%Informático%' );

SET @total\_proyectos = (SELECT COUNT(t.cod\_proyecto) FROM trabaja t INNER JOIN empleado e ON e.cod\_empleado = @cod\_empleado AND e.tipo LIKE '%Informático%' );

RETURN CONCAT(total\_horas,total\_proyectos);

END //

#Para que cree la funcion

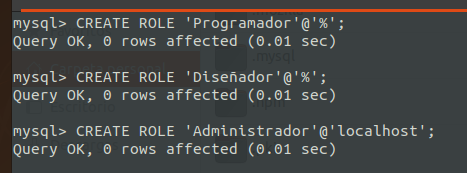
SET GLOBAL log\_bin\_trust\_function\_creators = 1;

**Seguridad**

Este tema requiere la creación de roles y cuentas de usuario, con asignación de políticas de seguridad y uso de recursos, y la asignación de privilegios. Deberán debatir en el grupo y fundamentar cada una de las tareas realizadas. Se pide:

1. Crear tres roles para que en la base de datos se pueda agrupar a los usuarios según los siguientes perfiles:

* Administrador
  + CREATE ROLE 'Administrador'@'localhost';
* Programador
  + CREATE ROLE 'Programador'@'%';
* Diseñador
  + CREATE ROLE 'Diseñador'@'%';



2. Debatir y fundamentar sobre los permisos necesarios para cada uno de los roles creados y realizar la asignación de los mismos.

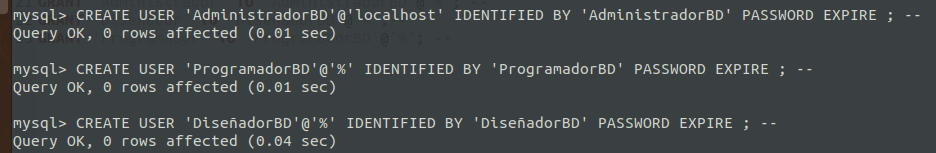
* Administrador: son responsables de muchas tareas críticas:
  + Diseño de los esquemas conceptual y físico
  + Seguridad y autorización
  + Disponibilidad de datos y recuperación en caso de fallos
  + Ajuste de las Bases de Datos

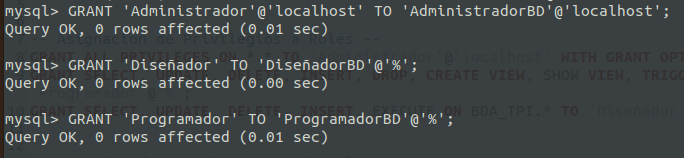
Debido a lo expresado anteriormente en cuanto a las tareas del administrador, se decide asignar los siguientes permisos a este rol:

* + GRANT ALL PRIVILEGES ON \*.\* TO 'Administrador'@'localhost' WITH GRANT OPTION;
* Programador: los programadores de aplicaciones de las bases de datos desarrollan paquetes que facilitan el acceso a los datos por parte de los usuarios finales mediante lenguajes anfitriones o de datos y herramientas software que proporcionan los fabricantes de SGBD. Los programadores de aplicación deberían tener acceso a los datos, en teoría, mediante el esquema externo. Debido a esto se considera la asignación de los siguientes permisos al rol:
  + GRANT SELECT, UPDATE, DELETE, INSERT, DROP, CREATE VIEW, SHOW VIEW, TRIGGER, EXECUTE,CREATE ROUTINE, ALTER ROUTINE ON BDA\_TPI.\* TO 'Programador'@'%';
* Diseñador: El diseñador de aplicaciones tiene como principal enfoque la Usabilidad y la Experiencia de Usuario. A grandes rasgos, hacer la aplicación más intuitiva y fácil de usar para que el usuario pueda llevar a cabo la tarea que quiere realizar (ya sea leer un artículo o comprar un libro) sin sentirse frustrado en el intento. Debido a esto, se considera las siguiente asignación de permisos al rol:
  + GRANT SELECT, UPDATE, DELETE, INSERT, EXECUTE ON BDA\_TPI.\* TO 'Diseñador'@'%';

3. Crear al menos una cuenta para cada rol, estableciendo y fundamentando las políticas utilizadas para nombres de los usuarios, host desde los que se pueden conectar, políticas de contraseñas, utilización de recursos.

* Administrador: se establece un nombre de usuario referido al rol que se le asignará posteriormente, incitando al cambio de contraseña después del primer inicio de sesión, ya que la misma por defecto será igual al nombre de usuario. Debido a que las tareas de administración conlleva muchas responsabilidades solo se podrá acceder de manera local para realizar esas acciones.
  + CREATE USER 'AdministradorBD'@'localhost' IDENTIFIED BY 'AdministradorBD' PASSWORD EXPIRE ;
  + GRANT 'Administrador'@'localhost' TO 'AdministradorBD'@'localhost';
* Programador: se establece un nombre de usuario referido al rol que se le asignará posteriormente, incitando al cambio de contraseña después del primer inicio de sesión, ya que la misma por defecto será igual al nombre de usuario. Debido a que un programador puede trabajar de manera remota, tendrá acceso a las funciones asignadas sobre la base de datos desde cualquier lugar.
  + CREATE USER 'ProgramadorBD'@'%' IDENTIFIED BY 'ProgramadorBD' PASSWORD EXPIRE ;
  + GRANT 'Diseñador' TO 'DiseñadorBD'@'%';
* Diseñador: se establece un nombre de usuario referido al rol que se le asignará posteriormente, incitando al cambio de contraseña después del primer inicio de sesión, ya que la misma por defecto será igual al nombre de usuario. Debido a que un diseñador puede trabajar de manera remota, tendrá acceso a las funciones asignadas sobre la base de datos desde cualquier lugar.
  + CREATE USER 'DiseñadorBD'@'%' IDENTIFIED BY 'DiseñadorBD' PASSWORD EXPIRE ;
  + GRANT 'Programador' TO 'ProgramadorBD'@'%';





4. Probar mediante distintas acciones que las cuentas de usuarios se comportan de acuerdo a lo planificado. Hacer capturas de pantalla de las pruebas realizadas.

**Acciones del Administrador**

CREATE ROLE 'Programador'@'%';

CREATE ROLE 'Diseñador'@'%';

GRANT SELECT, UPDATE, DELETE, INSERT, CREATE VIEW, SHOW VIEW, TRIGGER, EXECUTE,CREATE ROUTINE, ALTER ROUTINE ON BDA\_TPI.\* TO 'Programador'@'%';

GRANT ALL PRIVILEGES ON \*.\* TO 'Diseñador'@'%';

CREATE USER 'DiseñadorBD'@'%' IDENTIFIED BY 'DiseñadorBD' PASSWORD EXPIRE ;

CREATE USER 'ProgramadorBD'@'%' IDENTIFIED BY 'ProgramadorBD' PASSWORD EXPIRE ;

GRANT 'Diseñador' TO 'DiseñadorBD'@'%';

GRANT 'Programador' TO 'ProgramadorBD'@'%';

**Acciones del Diseñador**

SELECT \* FROM empleados;

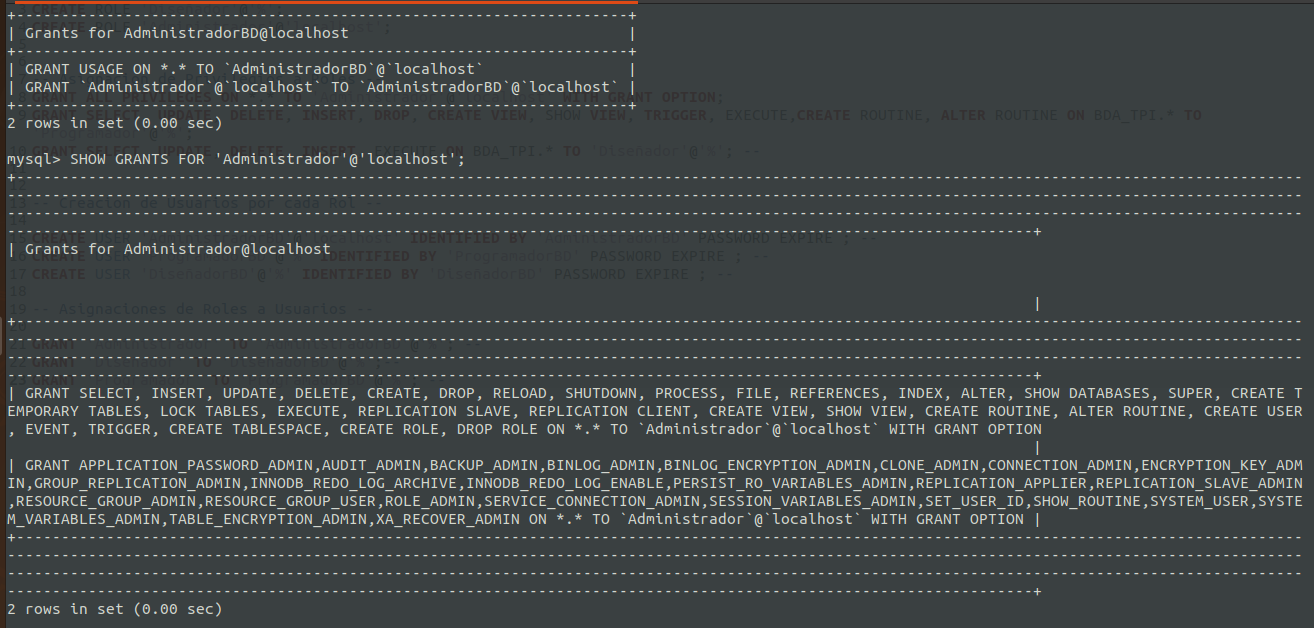
UPDATE empleados

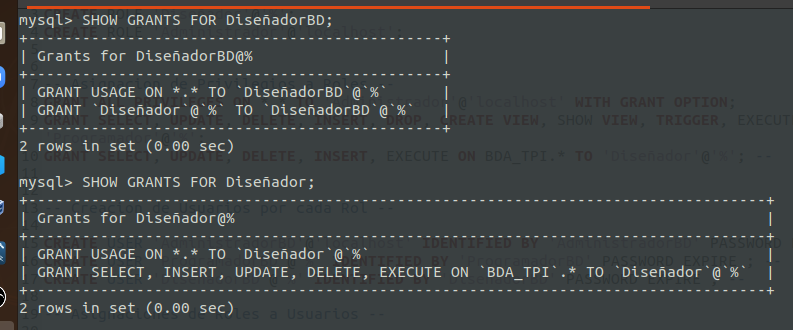
SET titulacion= 'Ingeniero'

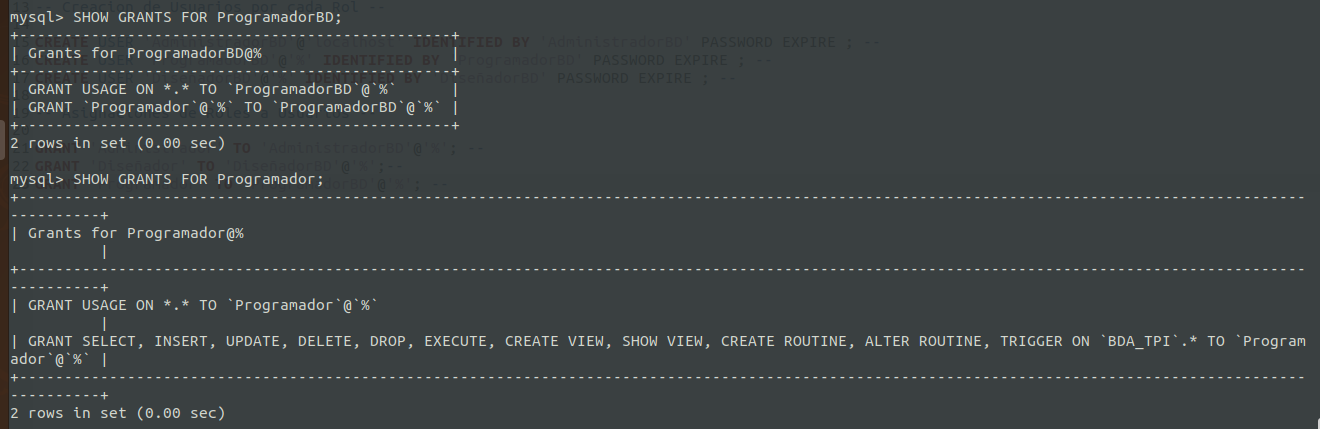
WHERE titulacion= 'Dr';

**Acciones del Programador**

CALL listar\_proyecto (3);



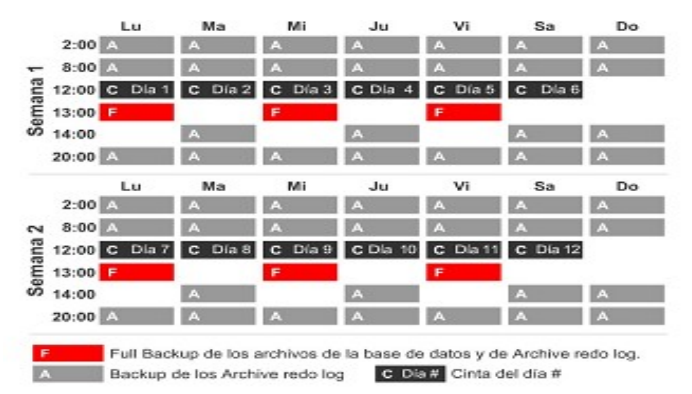




**Respaldo y recuperación**

Para un volumen de transacciones diarias (120.000 inserciones aprox.) y una disponibilidad requerida de la base de datos 24 horas al día, se debe implementar una política de backup soportada sobre un mecanismo de respaldo físico en caliente, con el fin de garantizar la menor pérdida de datos y su mayor disponibilidad, de acuerdo al tiempo que la aplicación debe estar en producción. Analizando la carga de la base de datos y las horas pico de consulta y transaccionalidad, se fijó como hora cero para backup, las 2:00 PM, ya que hacía las horas de la tarde la base de datos se encuentra con un nivel de carga bajo. Los intervalos de tiempo, en los cuales se hace el respaldo de los datos, son fijados de acuerdo al crecimiento en volumen de datos y el nivel de dinamismo que presenta la base de datos. Cuando se fija como política de respaldo el backup físico en línea, se corre el riesgo de provocar una caída en la base de datos, si no se garantiza espacio suficiente para el copiado de los archivos necesarios. Teniendo en cuenta lo anterior, se definió un esquema de respaldo con cintas diarias, realizando Full Backup de la base de datos los días lunes, miércoles y viernes, y backup de archivos de Log cada seis horas, eliminando estos archivos después de realizado el backup automáticamente, garantizando disponibilidad de espacio en disco para estos tipos de archivos. Se programó la toma del backup a través de tareas automáticas del sistema operativo, necesitando sólo la intervención del usuario, para el cambio de cinta y la validación de los backups. Se debe manejar un pool (conjunto) de doce (12) cintas, las cuales se deben intercambiar diariamente, rotándolas cada dos (2) semanas, es decir que se contará con el backup de las dos últimas semanas. Por fines prácticos y para evitar inconvenientes en el proceso de intercambio de rotación de cintas, se fijó las doce

del mediodía (12:00 PM), como la hora en que se debe realizar el intercambio de la cinta en el servidor, siguiendo la secuencia determinada. Este proceso se debe realizar todos los días de lunes a sábado y debe hacerlo la persona responsable de los backups. Por mantenimiento, confiabilidad y seguridad se recomienda cambiar el pool de cintas por unas nuevas cada seis meses. La Figura muestra el diagrama resultante de la esquematización de una estrategia de backup según el escenario planteado.



**Aclaración:** Tomaremos el Backup de Archivos "redo log" como backups tipo incremental. Si bien no es lo mismo, lo hacemos para no complicar el escenario.

Tareas:

a) Investigue ¿Qué es un redo log?. ¿Su base de datos lo soporta o es algo exclusivo de un motor en particular? En caso afirmativo ¿cómo lo llama? (Véase Aclaración)

La recuperación incremental es la recuperación de los cambios realizados durante un período de tiempo determinado. Esto también se denomina recuperación en un momento dado porque hace que el estado de un servidor sea actual hasta un momento determinado. La recuperación en un momento determinado se basa en el registro binario y, por lo general, sigue a una recuperación completa de los archivos de respaldo que restaura el servidor a su estado cuando se realizó el respaldo. Luego, los cambios de datos escritos en los archivos de registro binarios se aplican como recuperación incremental para rehacer las modificaciones de datos y llevar el servidor al punto deseado en el tiempo.

En el caso del escenario serían los backup designados con la letra A, que llevan a cabo diariamente, de lunes a domingos cada 6 horas de diferencia, a excepción de los días que se llevan a cabo el full backup, donde se puede ver una diferencia de 12hs, ya que el full backup incluye también el backup incremental.

b) Deberá buscar y comparar soluciones de hardware para dar soporte a esta planificación. Al menos deberán ser tres opciones como mínimo con sus respectivos análisis de costos y de las tres proponer la mejor opción.

NAS (Network-attached storage):

Pros:

+ Los backups automáticos no se interrumpirán por que te olvides

+ Las soluciones inalámbricas también funcionan en teléfonos y tablets

Contras:

– Puede ser costoso

– Puede ser incómodo para configurar y mantener

Raid:

• Conectar muchos discos en un mismo sistema.

• Es una herramienta de conectividad de varios discos independientes para que trabajen en un sistema por Hardware o Software que mejore la velocidad y la confiabilidad.

• Confiabilidad (redundancia).

• Velocidad de lectura escritura (paralelismo). Técnica de Distribución de Bandas (striping)

• Si un disco falla los datos no se pierden.

• Si los discos están escribiendo y se corta la energía antes de que ambos estén completamente escritos, éstos pueden quedar en estado incoherente.

1. Escribir una copia y luego la siguiente.

2. Usar como memoria caché una NVRAM que conserva los datos ante esta falla.

Discos de estados solidos (SSD):

Ventajas

• Tamaño y peso

• Resistencia

• Velocidad

• Consumo energético.

• No generan ruido ni vibraciones

• Mejora el MTBF

• No sufren por estar fragmentados

• Rendimiento constante y sin deterioros cuando se llena

Desventajas:

• Costo

• No hay estándares

• Necesitan recibir energía periódicamente

c) Describa el procedimiento de restauración que utilizaría si la base de datos se dañase el primer Jueves de la 2da semana a las 11 de la mañana. ¿Tiene un plan "B" por si falla alguna restauración?

Se podría ejecutar mysqlbinlog con una fecha y hora de inicio:

shell> mysqlbinlog --start-datetime="2020-11-19 10:59:59" \

/var/log/mysql/bin.123456 | mysql -u root -p

Otra opción (mediante posiciones):

shell> mysqlbinlog --start-datetime="2020-11-19 8:01:00" \

--stop-datetime="2020-11-19 10:59:59" \

/var/log/mysql/bin.123456 > /tmp/mysql\_restore.sql

Después de restaurar el archivo de respaldo anterior, use los números de posición para procesar el archivo de registro binario:

shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \

| mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \

| mysql -u root -p

Si algún fallo se debiera por alguna tabla se puede utilizar myisamchk para comprobar, reparar u optimizar tablas de bases de datos.

Vease en <https://dev.mysql.com/doc/refman/8.0/en/myisam-repair.html>

d) ¿Cómo debería modificar la planificación si en vez de dos semanas se necesitarán conservar al menos 3 meses? Esto implicaría inversión adicional en equipamiento / software? Justifique.

Si necesitara conservar los últimos 3 meses debería hacer 3 backup total mensual.

(Más cintas en el esquema de rotación no significa un mayor gasto. Las cintas se gastan en función a sus horas de uso)

Para esta empresa con alto volumen de transacciones conviene incluir un software de respaldo que permita administrar y controlar diversas copias de archivos importantes en dispositivos de almacenamiento.

**Datos semiestructurados**

Dadas las modificaciones al escenario inicial, que implican el cumplimiento de nuevos

requerimientos, deberán:

1. Analizar los nuevos requerimientos de registro de datos y obtener en base a la estructura del documento propuesta un esquema JSON a partir de alguna de las herramientas mostradas por la Cátedra.

2. Hacer una carga masiva de documentos utilizando alguna herramienta de generación de datos.

3. Resolver las consultas planteadas a continuación del escenario.

Escenario

Por el crecimiento exponencial que esta teniendo la empresa, decidieron implementar un sitio web con el objetivo que de capturar capital humano de valor actual y potencial. Se tendrá en cuenta la experiencia, los trabajos realizados y las necesidades de desarrollo personal que tengan. Esa información se volcará en formato JSON a la tabla de empleados de la base de datos desarrollada, de acuerdo a la siguiente estructura:

{"postulante" : {

"dni" : 40404040,

"nombre" : "juan paredez",

"edad" : 24,

"ciudad" : "Madrid",

"nivel estudio" : "universitario",

"email" : "juanpa24@correos.com",

"redes sociales" : [

{"instagram" : "juanpa24"},

{"twitter" : "juanpa24"}],

"experiencia laboral" : [

{"empresa" : {

"nombre" : "Yahoo",

"lenguaje" : "php",

"años" : 4

"puesto" : "programador"

"tipo " : "presencial"}

},

{"empresa" : {

"nombre" : "Mercadolibre",

"lenguaje" : "",

"años" : 1

"puesto" : "Scrum Master"

"tipo " : "presencial"}

},

{"empresa" : {

"nombre" : "Tiendamia",

"lenguaje" : "java",

"años" : 1

"puesto": "Testing",

"tipo" : "virtual"}

}],

"educación formal" : [

{"tema" : "php"}, {"lugar" : "Instituto"},

{"tema" : "Scrum"}, {"lugar" : "Utn"},],

"aspiraciones" : [

{"lenguajes" : ["javascript", "android"]},

{"puestos" : ["DevOps"]},

{"conocimientos" : ["git", " redes"]}]

}

}

Consultas:

Mediante el soporte JSON del SGBD, realice las siguientes consultas:

1. Obtener el promedio de edad de los postulantes con nivel de estudio universitario.

SELECT AVG(JSON\_EXTRACT(cjson, '$.edad')) as PromedioEdad

FROM postulantes;



2. Listar los nombres de las aspiraciones de puesto a ocupar con mayor cantidad de

interesados.

SELECT x.puestos, count(\*) AS Cantidad

FROM postulantes p

JOIN JSON\_TABLE(

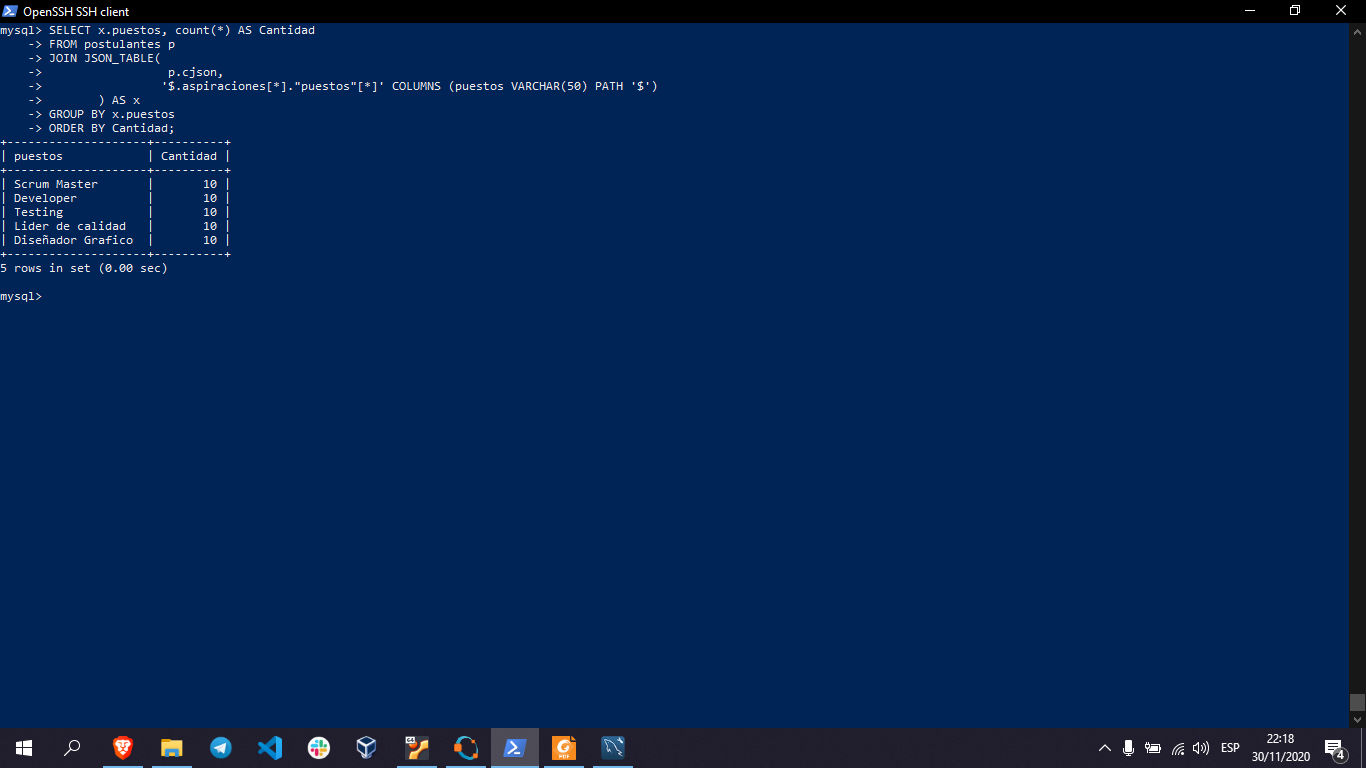
p.cjson,

'$.aspiraciones[\*]."puestos"[\*]' COLUMNS (puestos VARCHAR(50) PATH '$')

) AS x

GROUP BY x.puestos

ORDER BY Cantidad;

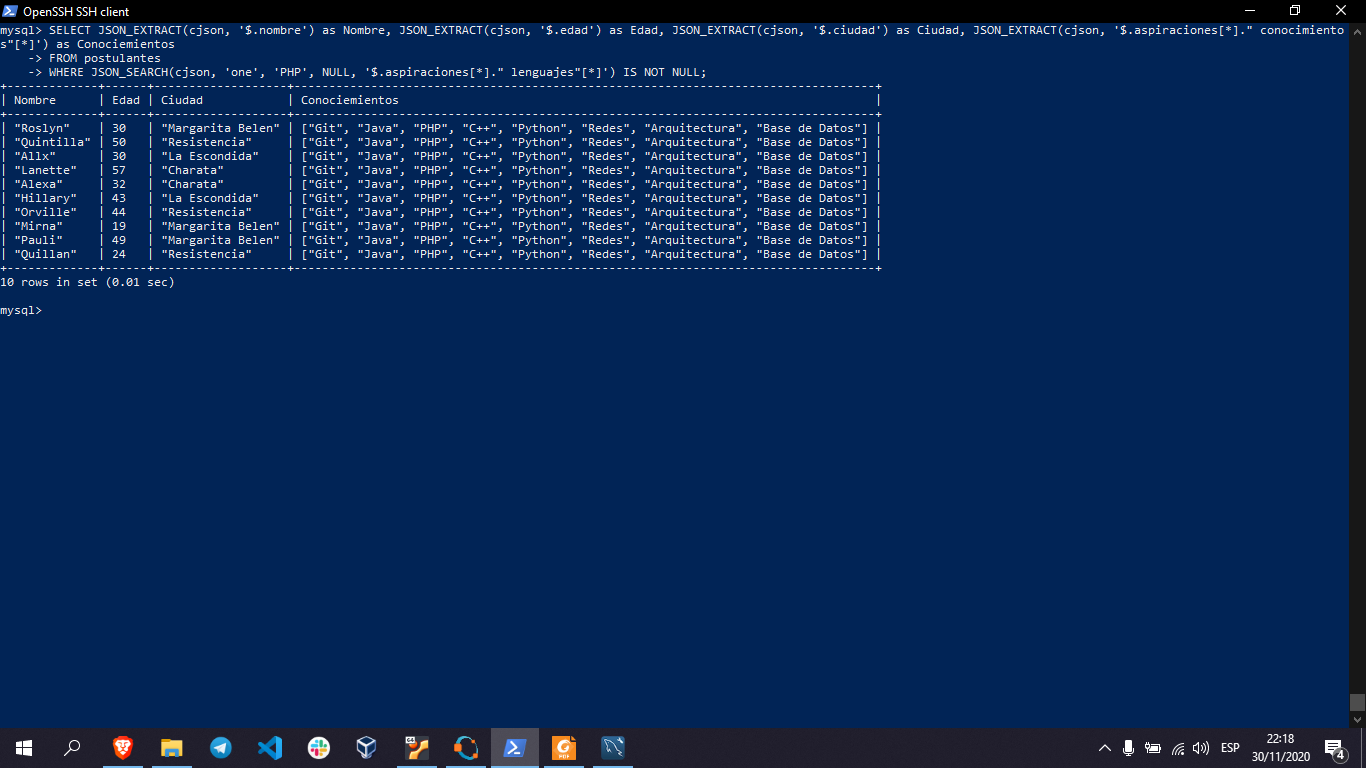


3. Listar los nombres, edades, ciudad de residencia y los conocimientos que aspiran aprender aquellos que tienen interes en aprender lenguaje:php.

SELECT JSON\_EXTRACT(cjson, '$.nombre') as Nombre, JSON\_EXTRACT(cjson, '$.edad') as Edad, JSON\_EXTRACT(cjson, '$.ciudad') as Ciudad, JSON\_EXTRACT(cjson, '$.aspiraciones[\*]." conocimientos"[\*]') as Conociemientos

FROM postulantes

WHERE JSON\_SEARCH(cjson, 'one', 'PHP', NULL, '$.aspiraciones[\*]." lenguajes"[\*]') IS NOT NULL;

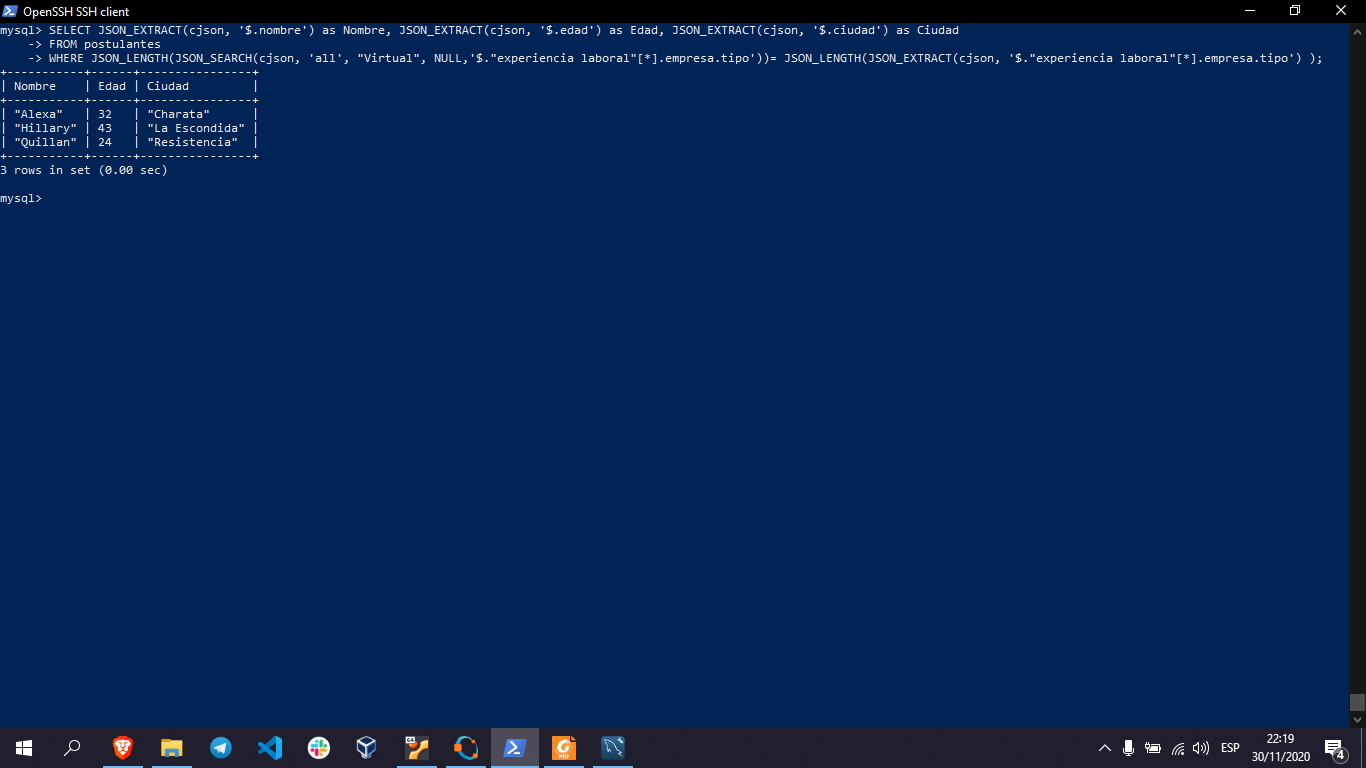


4. Listar los nombres, edades y ciudad de residencia de quines solamente trabajaron de manera virtual (ninguna presencial).

SELECT JSON\_EXTRACT(cjson, '$.nombre') as Nombre, JSON\_EXTRACT(cjson, '$.edad') as Edad, JSON\_EXTRACT(cjson, '$.ciudad') as Ciudad

FROM postulantes

WHERE JSON\_LENGTH(JSON\_SEARCH(cjson, 'all', "Virtual", NULL,'$."experiencia laboral"[\*].empresa.tipo'))= JSON\_LENGTH(JSON\_EXTRACT(cjson, '$."experiencia laboral"[\*].empresa.tipo') );



5. Listar los nombres y redes sociales de quienes estudiaron el lenguaje:java y les interesa aprender conocimiento:git.

SELECT JSON\_EXTRACT(cjson, '$.nombre') as Nombre, JSON\_EXTRACT(cjson, '$."redes sociales"[\*]') as Redes

FROM postulantes

WHERE (JSON\_SEARCH(cjson, 'one', 'Java', NULL, '$."educación formal"[\*].tema') IS NOT NULL) AND (JSON\_SEARCH(cjson, 'one', 'Git', NULL, '$.aspiraciones[\*]." conocimientos"[\*]') IS NOT NULL);

