EI1052 - Sistemas de gestión de bases de datos

Examen. Primera convocatoria. Junio 2019

Nombre y Apellidos: Joshua Garcia Olucha

RESULTADOS DE APRENDIZAJE

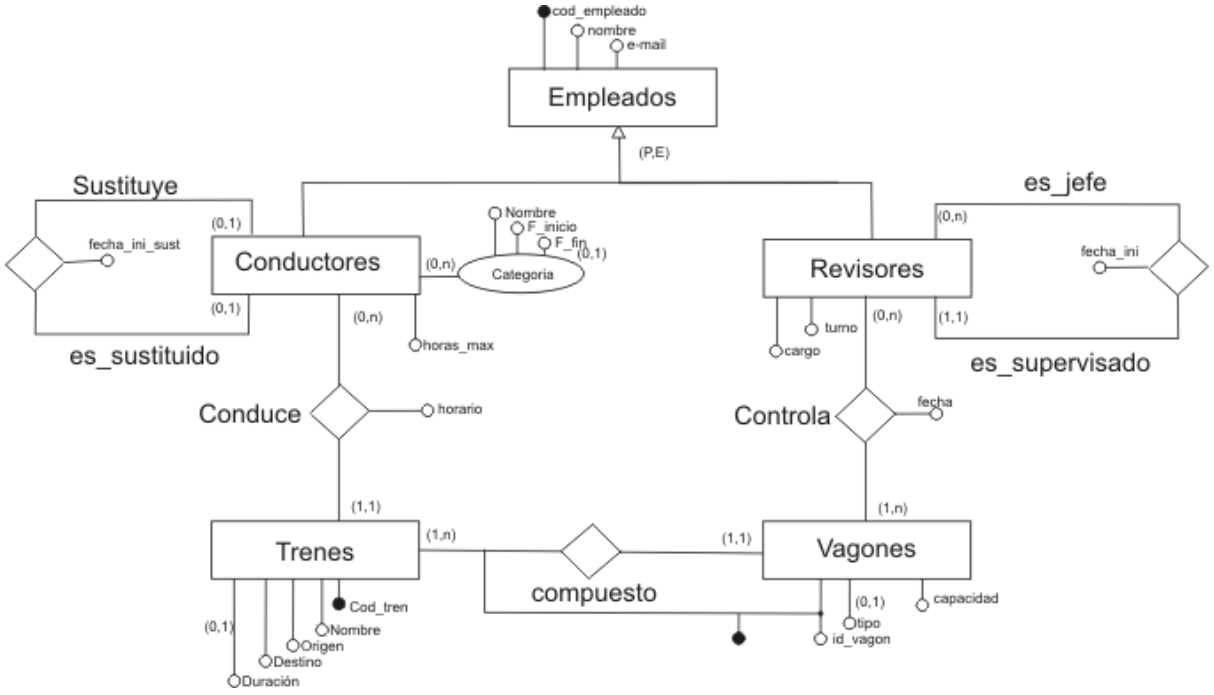
- ETI.7.1 - Realizar un diseño físico de una base de datos que permita el acceso eficiente.
- ETI.7.3 - Planificar la implantación de un sistema de bases de datos.
- ETI.7.4 - Medir los parámetros de rendimiento de un sistema de base de datos.
- ETI.7.5 - Realizar tareas de administración de sistemas de bases de datos.

INSTRUCCIONES

- Crea una copia del documento compartido para contestar al examen. Compartela con el profesor. Una vez acabado el plazo, crea un pdf y súbelo al Aula virtual. No borres el *gdoc* donde has trabajado, ni lo modifiques después de la fecha de entrega del examen.
- El plazo para subir al Aula Virtual el documento pdf con las respuestas finaliza el 25/6/2019 a las 19:00.
- Solo se evaluará el contenido del documento pdf.
- La nota de este examen supone un 45% de la nota final de la asignatura.
- Para sumar la nota de este examen a la evaluación continua hay que obtener, al menos, un 50% de la nota del examen.
- Durante la realización del examen puedes preguntar dudas sobre el enunciado al profesor. Recuerda, solo el profesor.
- Existen diferentes modelos de exámenes. La realización del examen es individual, por lo que cada estudiante deberá hacer su examen en solitario, sin solicitar ayuda a ninguna otra persona.
- Al realizar este examen el estudiante se compromete a **no plagiar** el trabajo de otras personas y a no **permitir que otras personas plagien** su trabajo.
- Si fuera necesario, el alumno podría tener que pasar por una revisión para defender su examen.

ENUNCIADO

El esquema conceptual que se muestra continuación es de una empresa de trenes y se utiliza para almacenar la información sobre sus empleados y los trenes que conducen y revisan. Se quiere diseñar una base de datos para almacenar la información de esta empresa.



Ejercicio 1.

- **(2 puntos)** Realiza el modelo lógico del modelo conceptual presentado (en primera forma normal). No hace falta indicar las restricciones de modificación y borrado. En base a tu modelo deberás realizar el resto de apartados del problema.

Ningún atributo acepta nulos a menos que se indique

CONDUCTORES(cod_empleado, nombre, edad, e-mail, horas_max)

CATEGORIA(cod_empleado, nombre, f_inicio, f_fin)
f_fin acepta nulos
CATEGORIA → cod_empleado → CONDUCTORES nulos: no

SUSTITUCIONES(id_sustitucion, cod_empleado_sustituido, cod_empleado_sustituye, fecha_ini_sust)
cod_empleado_sustituido y cod_empleado_sustituye aceptan nulos

SUSTITUCIONES → cod_empleado_sustituido → CONDUCTORES nulos: no
SUSTITUCIONES → cod_empleado_sustituye → CONDUCTORES nulos: no

TRENES(cod_tren, nombre, origen, destino, duracion, horario, cod_empleado)
TRENES → cod_empleado → CONDUCTORES nulos: no

REVISORES(cod_empleado, nombre, e-mail, turno, cargo, jefe, fecha_ini)
REVISORES→ jefe → REVISORES nulos: no

VAGONES(cod_tren, id_vagon, tipo, capacidad)
tipo acepta nulos
VAGONES → cod_tren → TRENES nulos: no

CONTROLES(id_control, cod_empleado, cod_tren, id_vagon, fecha)
CONTROLES → cod_empleado → REVISORES nulos: no
CONTROLES → cod_tren → VAGONES nulos: no
CONTROLES → id_vagon → VAGONES nulos: no

- **(1,5 punto)** Crea el/los disparador/es necesarios para mantener la siguiente restricción “Un conductor de un tren no puede controlar los vagones que componen ese tren”.

Según el modelo conceptual la clasificación es parcial y exclusiva, y por lo tanto un conductor no debería ser revisor, pero el disparador que podría regular esta restricción es el siguiente:

```
CREATE OR REPLACE FUNCTION funConductoresDeVagones() RETURNS TRIGGER AS
'
	BEGIN
		IF new.cod_empleado = (select con.cod_empleado FROM conductores as con JOIN trenes as t USING (cod_empleado) JOIN vagones as v USING (cod_tren) WHERE t.cod_tren = new.cod_tren)
		THEN
			RAISE NOTICE "No puedes añadir este control porque el conductor del tren es el revisor del vagon";
			RETURN OLD;
		ELSE
			INSERT INTO controles VALUES (new.id_control, new.cod_empleado, new.cod_tren, new.id_vagon, new.fecha);
			RETURN NEW;
		END IF;
	END;
	' LANGUAGE 'plpgsql';

CREATE TRIGGER trgConductoresDeVagones
	BEFORE INSERT OR UPDATE ON controles
```

FOR EACH ROW
EXECUTE PROCEDURE funConductoresDeVagones () ;

- **(2 puntos)** Se desea modificar el diseño propuesto para añadir un nuevo atributo que almacene el número de revisores que un revisor supervisa (sin recursividad).
  - Escribe la instrucción para modificar la tabla que incorpore el atributo *n_supervisados*.

ALTER TABLE revisores ADD COLUMN n_supervisados INTEGER;
Añadir el campo n_supervisados en revisores siendo de tipo integer.

  - Una vez añadido el atributo, escribe la sentencia que permitiría actualizar este campo.

UPDATE revisores SET n_supervisados = 5 WHERE cod_empleado = 2;
Actualizar el campo n_supervisores a 5 supervisados en el empleado jefe con codigo 2.

  - Implementa mediante el/los disparador/es necesarios que permitan mantener este atributo actualizado.

```
CREATE TRIGGER trgActualizacionNSupervisados
	BEFORE INSERT OR DELETE OR UPDATE ON revisores
	FOR EACH ROW
	EXECUTE PROCEDURE funActualizarNSupervisados () ;

CREATE OR REPLACE FUNCTION funActualizarNSupervisados() RETURNS TRIGGER
AS '
	BEGIN
		IF TG_OP = "INSERT" THEN
			INSERT INTO revisores VALUES (new.cod_empleado, new.nombre,
new.e_mail, new.turno, new.cargo, new.jefe, new.fecha_ini);
			UPDATE revisores SET n_supervisados = n_supervisados+1 WHERE
cod_empleado = new.jefe;
			RETURN NEW;
		ELSEIF TG_OP = "DELETE" THEN
			DELETE FROM revisores WHERE cod_empleado =
old.cod_empleado;
			UPDATE revisores SET n_supervisados = n_supervisados-1 WHERE
cod_empleado = old.jefe;
```

```
                RETURN OLD;
        ELSE
                IF new.jefe = old.jefe THEN
                        RETURN NEW;
                ELSE
                        UPDATE revisores SET n_supervisados = n_supervisados+1
WHERE cod_empleado = new.jefe;
                        UPDATE revisores SET n_supervisados = n_supervisados-1
WHERE cod_empleado = old.jefe;
                        RETURN NEW;
                END IF;
        END IF;
    END;
    ' LANGUAGE 'plpgsql';
```

- **(1 punto)** Crea la vista *TrenesValencia* que muestre los trenes con origen en Valencia cuyos conductores tengan más de 55 años. La vista debe mostrar el código de tren, el destino, la duración, el tipo de viaje y el código y nombre del conductor, además del nombre de su categoría actual (su categoría actual es aquella donde fecha_fin es nulo). Tipo de viaje es: Local si la duración es menor de 1 hora, Regional si la duración es mayor que 1 hora y menor que 3 y Nacional si la duración es mayor que 3.

```
CREATE VIEW TrenesValencia as
    SELECT t.cod_tren, t.destino, t.duracion, con.cod_empleado, con.nombre,
    (SELECT nombre FROM categoria WHERE f_fin = NULL  AND cod_empleado =
con.cod_empleado) AS nombre_categoria,
    (SELECT
        (CASE WHEN t.duracion < 60 THEN 'Local' WHEN t.duracion BETWEEN 60
AND 180 THEN 'Regional' WHEN t.duracion > 180 THEN 'Nacional' END) as tipo_de_viaje
    FROM categoria WHERE cod_empleado = con.cod_empleado)
    FROM trenes AS t JOIN conductores AS con USING (cod_empleado) JOIN
categoria AS cat USING (cod_empleado)
    WHERE t.origen = 'Valencia' AND con.edad > 55;
```

- **(1,5 puntos)** Cuando los usuarios de la aplicación utilizan la vista *TrenesValencia* pueden detectar que un tren con un determinado destino tiene una duración errónea. ¿es actualizable la vista? ¿por qué? Si no lo es, escribe el/los disparador/es necesarios que permitan modificar la duración a través de la vista.

No es actualizable ya que la vista afecta a varias tablas y para su actualización es necesario el uso de reglas para los privilegios de borrado y actualización o el uso de disparadores. El disparador para actualizar la duracion de los trenes es el siguiente:

```
CREATE TRIGGER trgActualizacionDuracion
        INSTEAD OF INSERT OR DELETE OR UPDATE ON trenesvalencia
        FOR EACH ROW
        EXECUTE PROCEDURE funActualizarNSupervisados () ;

CREATE OR REPLACE FUNCTION funActualizarNSupervisados() RETURNS TRIGGER AS '
        BEGIN
                IF TG_OP = "UPDATE" THEN
                        UPDATE trenes SET duracion = new.duracion WHERE cod_tren = new.cod_tren;
                        RETURN NEW;
                END IF;
        END;
        ' LANGUAGE 'plpgsql';
```

Ejercicio 2. (2 puntos)

En la base de datos de una bombonería tenemos la tabla en donde se indica con qué chocolate hacer cada bombón que se coloca en las cajas: BOMBONES(id, chocolate). En el estado inicial del que partimos la tabla contiene las siguientes filas:

```
id chocolate
---- ----------
 1 blanco
 2 negro
 3 blanco
 4 negro
```

Dos maestros chocolateros que trabajan en la misma bombonería están preparándose para ejecutar las transacciones T1 y T2. Como verás, al maestro que va a ejecutar T1 solo le gusta el chocolate blanco y mientras que al maestro que va a ejecutar T2 solo le gusta el negro.

```
T1: START TRANSACTION;
  UPDATE bombones SET chocolate=’blanco’ WHERE chocolate=’negro’;
  COMMIT;
```

```
T2: START TRANSACTION;
  UPDATE bombones SET chocolate=’negro’ WHERE chocolate=’blanco’;
  COMMIT;
```

Tras la ejecución concurrente de T1 y T2 (ambos maestros intentan hacer todos los bombones a su gusto a la vez), hemos llegado al siguiente estado:

```
 id chocolate
---- ----------
  1 negro
  2 blanco
  3 negro
  4 blanco
```

Indica en qué nivel de aislamiento se han ejecutado ambas transacciones y da argumentos que justifiquen adecuadamente porqué es ese el nivel de aislamiento y no otro diferente. Si lo necesitas, puedes suponer que estamos trabajando con las versiones de los SGBD de las que disponemos en db-aules.uji.es.

El nivel de aislamiento es READ COMMITTED ya que ocurren lecturas irrepetibles, como se ve que al leer la tabla antes y después cambia el resultado, y no es READ UNCOMMITTED porque no ocurren lecturas sucias ya que el resultado de una lectura sucia hubiese sido todo los chocolates a negro o blanco según el orden de ejecución.