EI1052 Sistemas de gestión de bases de datos

Examen. Primera convocatoria. Enero 2015

Nombre y Apellidos: Mihaita Alexandru Lupoiu

RESULTADOS DE APRENDIZAJE

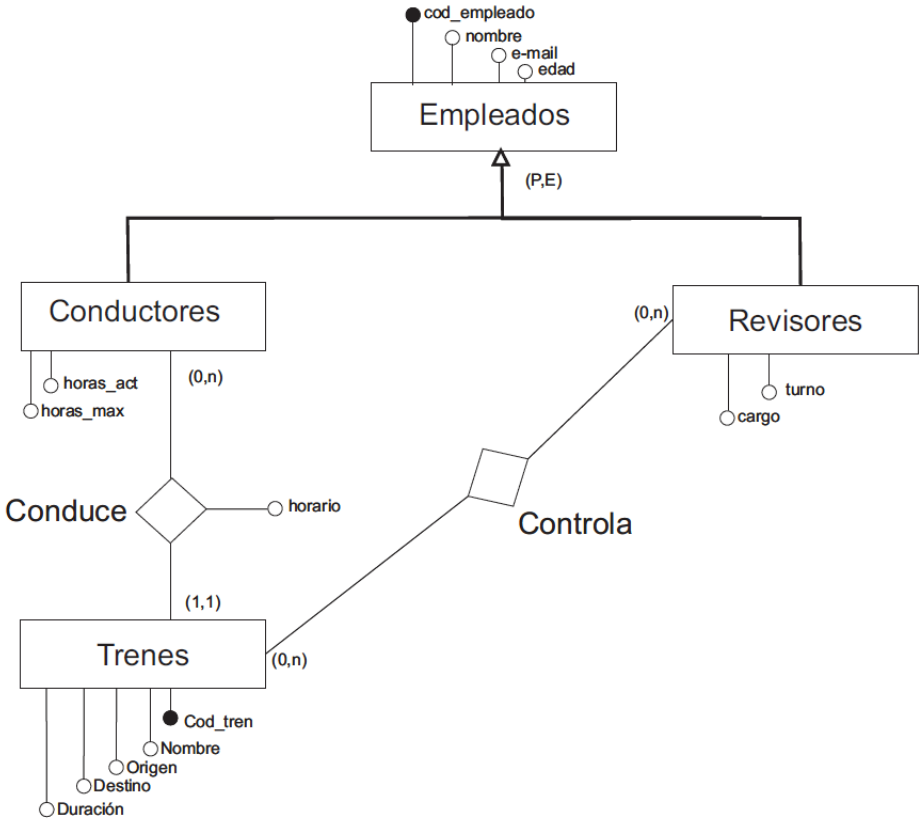
- ETI.7.1 - Realizar un diseño físico de una base de datos que permita el acceso eficiente.
- ETI.7.3 - Planificar la implantación de un sistema de bases de datos.
- ETI.7.4 - Medir los parámetros de rendimiento de un sistema de base de datos.
- ETI.7.5 - Realizar tareas de administración de sistemas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

INSTRUCCIONES

- Copia o descarga este documento para contestar al examen.
- El plazo para subir al Aula Virtual el documento con las respuestas finaliza el 14/1/2015 a las 13:00.
- 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.
- 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.

ENUNCIADO

El siguiente modelo conceptual representa parte del sistema de información de una empresa de trenes. Los empleados pueden clasificarse en empleados, conductores y revisores. Los conductores pueden conducir uno o varios trenes. Los revisores pueden ser revisores o responsables del tren, en este caso su cargo es revisor o responsable.



El diseño lógico realizado es el siguiente:

Empleados(cod_empleado, nombre, e-mail, edad)
Conductores(cod_empleado, horas_Act,horas_max)
Revisores (cod_empleado, turno, cargo)
Trenes (Cod_tren, nombre, origen, destino, duración, cod_empleado, horario)
Controla (cod_tren, cod_empleado)

Ejercicio 1. (0,5 puntos). Crea una vista que se llame *Infotrenes* y que muestre la información de los trenes con origen en Castellón. El resultado debe ser similar al siguiente

tren	trayecto	cod_empleado	conductor	revisor
1	Castellón-Madrid	1	Jose Garcia	Jesus Gonzalez
2	Castellón-Valencia	1	Jose Garcia	Carlos remon
3	Castellón-Barcelona	2	Antonio Jose	Sin asignar

```
CREATE VIEW Infotrenes AS
        SELECT T.cod_tren as Tren, T.origen || ' - ' || T.destino AS trayecto,
        T.cod_empleado, E.nombre AS conductor,  COALESCE(R.nombre, 'Sin asignar') as revisor
        FROM Trenes AS T
                LEFT JOIN Empleados AS E USING (cod_empleado)
                LEFT JOIN
                (       SELECT CO.cod_tren, E.nombre
                        FROM Controla as CO
                                JOIN Empleados AS E USING (cod_empleado)
                        ) AS R USING (cod_tren)
        WHERE UPPER(T.origen) like 'CASTELLON';
```

Ejercicio 2. (2 puntos). Crea un disparador que permita actualizar el conductor del tren (solo el id) mediante la vista. Independientemente de cómo la hayas creado, el disparador mantener la restricción WITH CHECK OPTION a la vista. ¿Qué crees que pasará con el nombre del conductor? Razona tu respuesta.

```
CREATE OR REPLACE FUNCTION actualizarConductor() RETURNS TRIGGER AS '
        DECLARE
                id INTEGER;
                id_tren INTEGER;
        BEGIN
                SELECT COUNT(cod_empleado) INTO id
                FROM   conductores
                WHERE  cod_empleado = NEW.cod_empleado;

                SELECT tren INTO id_tren
                FROM   infotrenes
                WHERE  cod_empleado = OLD.cod_empleado;

                IF (id > 0) THEN
                        UPDATE trenes
                        SET cod_empleado = NEW.cod_empleado
                        WHERE cod_empleado = OLD.cod_empleado
                        AND cod_tren = id_tren;
                ELSIF (id < 1) THEN
                RAISE EXCEPTION ''El codigo empleado % no esta el la lista de conductores'',
NEW.cod_empleado;
                END IF;
                RETURN NEW;
         END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER actualizaConductorTrenes
        INSTEAD OF UPDATE ON infotrenes
        FOR EACH ROW
        EXECUTE PROCEDURE actualizarConductor();
```

Como el código del empleado cambiará, dado que la búsqueda del nombre del conductor se realiza sobre el campo cod_empleados de trenes. Al cambiar ese valor, se cambia el nombre del conductor también.

Ejercicio 3. Ante un nuevo requisito planteado por la empresa llegamos a la conclusión de que debemos añadir una nueva columna a la base de datos que refleje siempre, y de modo automático (lo haremos mediante disparadores), el número de trenes que conduce cada empleado. Este nuevo dato es un dato derivado ya que se obtiene a partir de datos que se encuentran en la base de datos.

1) **(0.1 puntos)** Determina la tabla a la que debes añadir la nueva columna y el tipo de datos que deberá tener. Da argumentos para justificar tu elección.

   En la tabla de conductores se crea la columna num_trenes y sería de tipo INTEGER, porque es la manera fácil de controlar el numero de trenes que tiene cada conductor alterando lo mínimo la base de datos.

2) **(0.5 puntos)** Teniendo en cuenta el dato derivado que se desea mantener siempre actualizado ¿cuál es el predicado que se debe respetar en todo momento? Usa el lenguaje SQL para expresar este predicado.

```
EXISTS( SELECT *
        FROM conductores AS C
        WHERE C.num_trenes = ( SELECT COUNT(*)
                               FROM trenes AS T
                               WHERE T.cod_empleado = C.cod_empleado); )
```

3) **(0.1 puntos)** ¿Ante qué eventos (INSERT, UPDATE, DELETE) sobre qué tablas se puede violar el predicado establecido en el apartado 2?

   INSERT trenes: actualizar numero de trenes de conductores.
   UPDATE trenes: cambiar el numero de trenes de cada conductor.
   DELETE trenes: descontar en numero de trenes de un conductor.

   INSERT conductores: numero de trenes en un principio es 0.
   UPDATE conductores: actualizar num_trenes en función de los trenes.

4) **(0.1 puntos)** ¿Qué acciones se han de realizar ante los eventos establecidos en el apartado 3? Describe las acciones a realizar usando el lenguaje natural.

   Insertar en trenes un tren y asignarle un conductor. Actualizar el numero de trenes del conductor.
   Actualizar un tren y cambiar un conductor. Actualiza el numero de trenes del conductor.
   Eliminar un tren. Actualiza el numero de trenes del conductor.

   Insertar un conductor. Numero de trenes asignado en un principio es 0.
   Actualizar un conductor. Actualizar numero de trenes asignados en función de los trenes.

5) **(0.5 puntos)** Escribe la sentencia ALTER TABLE que añade la nueva columna a la base de datos. Teniendo en cuenta que el mantenimiento de este dato derivado ha de ser transparente para la aplicación que actualmente se usa en la empresa de alquiler de vehículos (no vamos a modificar el código ya implementado puesto que habría que volver a instalarlo en múltiples ordenadores), explica si resulta necesario o no el asignar un valor por defecto (DEFAULT) a la nueva columna, y en su caso, tenlo en cuenta en la sentencia ALTER TABLE.

Se añadirá por defecto el valor 0 en ese campo, de esa manera en caso de añadir un conductor y no se especifica un valor, se pondrá el valor 0 y no afectará el funcionamiento de la base de datos. Además cuando se añade un conductor nuevo el numero de trenes que conduce es siempre 0, por lo que no hará hacer ningún caso especial cuando se realizan INSERTs.

```
ALTER TABLE conductores ADD COLUMN num_trenes INTEGER NOT NULL DEFAULT 0;
```

6) **(2 puntos)** Escribe el código de los disparadores correspondientes a los eventos y las acciones del apartado 4. Es importante que tengas en cuenta la respuesta que has dado en el apartado anterior. Para comprobar el correcto funcionamiento de los disparadores seguramente necesitarás inicializar la nueva columna con el valor correspondiente al estado actual de la base de datos. Una sola sentencia UPDATE te servirá para llevar a cabo esta inicialización.

```
-- TRENES:
CREATE OR REPLACE FUNCTION trenesConductor() RETURNS TRIGGER AS '
  DECLARE
    numero_trenes INTEGER;
  BEGIN

    IF TG_OP = ''INSERT'' OR TG_OP = ''UPDATE'' THEN

      SELECT COUNT(*) INTO numero_trenes
      FROM Trenes
      WHERE cod_empleado=NEW.cod_empleado;

      UPDATE conductores
      SET num_trenes=numero_trenes
      WHERE cod_empleado=NEW.cod_empleado;

      IF TG_OP = ''UPDATE'' THEN
        UPDATE conductores
        SET num_trenes=numero_trenes
        WHERE cod_empleado=OLD.cod_empleado;
      END IF;
    ELSE
      SELECT COUNT(*) INTO numero_trenes
      FROM Trenes
      WHERE cod_empleado=OLD.cod_empleado;

      UPDATE conductores
      SET num_trenes=numero_trenes
      WHERE cod_empleado=OLD.cod_empleado;
    END IF;
    RETURN NEW;
  END;
' LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER trenesConductor
   AFTER INSERT OR UPDATE OR DELETE ON Trenes
   FOR EACH ROW
   EXECUTE PROCEDURE trenesConductor();

-- CONDUCTORES:
CREATE OR REPLACE FUNCTION conductorTrenes() RETURNS TRIGGER AS '
   DECLARE
      numero_trenes INTEGER;
   BEGIN

      SELECT COUNT(*) INTO numero_trenes
      FROM TRENES
      WHERE cod_empleado = NEW.cod_empleado;

      IF NEW.num_trenes != numero_trenes THEN
         NEW.num_trenes = numero_trenes;
      END IF;

   RETURN NEW;
 END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER conductorTrenes
   BEFORE INSERT OR UPDATE ON Conductores
   FOR EACH ROW
   EXECUTE PROCEDURE conductorTrenes();
```

7) **(0.5 puntos)** Como sabes, cuando escribimos una sentencia INSERT podemos indicar después del nombre de la tabla (entre paréntesis) los nombres de las columnas a las que corresponden los valores que vamos a insertar. Por ejemplo:
INSERT INTO tabla(col1,col2,col3) VALUES (1,2,3);
Explica qué problemas se podrían producir al trabajar con la aplicación que está usando la empresa, tras ejecutar la sentencia ALTER TABLE del apartado 5 sobre la base de datos, si al escribir el código de la aplicación no hubiéramos especificado los nombres de las columnas en las sentencias INSERT. Por ejemplo:
INSERT INTO tabla VALUES (1,2,3)

El problema que existe de insertar los valores sin especificar las columnas es que el orden de las columnas sea alterado y por lo tanto insertar unos valores erróneos en la base de datos. Por ello se debe revisar e intentar especificar las columnas.

Ejercicio 4. (1 punto). Averigua qué índices crea el SGBD cuando se ejecutan las sentencias CREATE TABLE que aparecen en el boletín de la práctica 1. Explica cómo lo has averiguado y explica también el motivo por el cual se han creado estos índices de manera automática. Por último, indica sobre qué columna sería conveniente crear un nuevo índice, justifica su conveniencia y escribe la sentencia CREATE INDEX correspondiente.

Mirando la documentación de la versión de PosgreSQL 9.3 (la que he utilizado en mis comprobaciones) explica que PosgreSQL crea automáticamente índices para los valores únicos y claves primarias de una tabla. Por esa razón en teoría para la práctica 1 debería de crear 3 índices. Dos para las claves primarias de cada tabla (Tipos y Modelos) y otro para el campo único de Modelo (nombre).
http://www.postgresql.org/docs/9.3/static/indexes-unique.html

Después de crear las tablas, efectivamente los 3 índices aparecieron.

```
practica1=# select * from pg_indexes where tablename = 'tipos';
 schemaname | tablename | indexname | tablespace |                          indexdef
------------+-----------+-----------+------------+-------------------------------------------------------------
 public     | tipos     | cp_tipos  |            | CREATE UNIQUE INDEX cp_tipos ON tipos USING btree (idtipo)
(1 row)

practica1=# select * from pg_indexes where tablename = 'modelos';
 schemaname | tablename |      indexname      | tablespace |                                indexdef
------------+-----------+---------------------+------------+-------------------------------------------------------
----------------------
 public     | modelos   | cp_modelos          |            | CREATE UNIQUE INDEX cp_modelos ON modelos USING
btree (idmod)
 public     | modelos   | calt_modelos_nombre |            | CREATE UNIQUE INDEX calt_modelos_nombre ON
modelos USING btree (nombre)
(2 rows)
```

Sería conveniente crear un índice sobre "tarifa" de la tabla "modelos" porque es un valor que se consulta a menudo, con un grado de selectividad bastante alto. Este valor se cambiará pero cada cierto tiempo, no diariamente por lo que no supondrá un alto coste en realizar las modificaciones.

```
CREATE UNIQUE INDEX tarifaModelo ON modelos (tarifa);
```

Ejercicio 5. (0,5 puntos) Según se indica en el manual de PostgreSQL 8.3, este SGBD dispone de un módulo que se encarga de recoger estadísticas que podemos utilizar después para analizar el rendimiento: número de accesos a tablas y a índices, además de las estadísticas que ya conocemos en cuanto al número de las de cada tabla, los bloques
de disco que ocupa cada una y los que ocupan sus índices, etc. Puesto que el recoger estas estadísticas puede suponer una sobrecarga al ejecutar las consultas, el administrador puede deshabilitar su recogida mediante dos parámetros de configuración del fichero *postgresql.conf*. Estos parámetros también se pueden cambiar dentro de una sesión individual mediante la orden SET, aunque para evitar que los usuarios intenten ocultar su actividad a los ojos del administrador, solo los superusuarios pueden cambiarlos ¿Cual es el nombre de los dos parámetros de configuración del recolector de estadísticas?

El nombre de los dos parámetros son "Track Counts" y "Track Activities".
LINK: http://www.postgresql.org/docs/8.3/static/monitoring-stats.html

Ejercicio 6. (2 puntos). Tenemos dos transacciones T1 y T2. A partir de un estado concreto de la base de datos, la ejecución serie de ambas transacciones en este orden: T1, T2, hace que T1 obtenga los resultados: 300 y 200. Ejecutando las transacciones en orden inverso y partiendo del mismo estado que en la situación anterior, los resultados que obtiene T1 son: 240 y 180. Ejecutando el plan no serie que se muestra a continuación, los resultados que obtiene T1 son 300 y 180.

T1: SELECT MAX(a) FROM t WHERE tipo = 1;
T2: UPDATE t SET a = 0 WHERE a = 300;
T1: SELECT MAX(a) FROM t WHERE tipo = 2;
T2: UPDATE t SET a = 0 WHERE a = 200;
T1: COMMIT;
T2: COMMIT;

Explica en qué nivel de aislamiento nos encontramos. Justifíca la respuesta

index	a	tipo
1	300	1
2	200	2
3	240	1
4	180	2

Estamos en el nivel de aislamiento “READ UNCOMMITTED” porque T1 recupera el dato 180 (en lugar de 200) que fue modificado por la transacción T2. Esto significa que se ha realizado una lectura sucia porque se realizó antes de que se haga el commit y el único nivel de asilamiento que lo permite es “READ UNCOMMITTED”.

TEORÍA:
READ UNCOMMITTED puede recuperar datos modificados pero no confirmados por otras transacciones (**lecturas sucias - dirty reads**). En este nivel se pueden producir todos los efectos secundarios de simultaneidad (lecturas sucias, lecturas no repetibles y lecturas fantasma), pero no hay bloqueos ni versiones de lectura, por lo que se minimiza la sobrecarga. **Una operación de lectura (SELECT) no establecerá bloqueos compartidos (shared locks)** sobre los datos que está leyendo, por lo que no será bloqueada por otra transacción que tenga establecido un bloqueo exclusivo por motivo de una operación de escritura.

Ejercicio 7. (0,20 puntos) Explica (sí, con tus propias palabras), lo que has aprendido en la asignatura y cómo puedes utilizarlo en los distintos papeles desde los que te puedes enfrentar a un SGBD: diseñando una base de datos, programando aplicaciones, etc. No utilices más de una página para responder a esta pregunta. Recuerda: usa tus propias palabras y exprésate en primera persona.

En la asignatura he realizado un buen repaso de cómo funcionaba un Sistema Gestor de Bases de Datos, además de haber ayudado a entender mejor como funciona internamente y ver el alcance que puede llegar a tener.

He aprendido que los manuales me pueden ayudar mucho a la hora de entender como funciona un SGDB en concreto y puede facilitar la respuesta a algún problema encontrado o entender algunas de las anomalías encontradas.

He aprendido como funciona PosgreSQL, Oracle y visto por encima como funciona MySQL, junto con las ventajas, inconvenientes, parecidos y diferencias. Estudié como optimizar las consultas o mejorar el rendimiento de la base de datos con ayuda de los índices en algunos casos.

También he aprendido utilizar los triggers o disparadores para gestionar las consultas y facilitar la administración de la base de datos, evitar errores o incluso permitir insertar o actualizar información en las vistas si es necesario.

Aunque la recomendación que recibí es que deje la base de datos de la forma más sencilla posible y crear la lógica en la aplicación.

Creación Base de datos (Posgresql 9.35):

```
CREATE TABLE Empleados (
        cod_empleado        SERIAL,
        nombre              VARCHAR(20),
        email               VARCHAR(20),
        edad                INTEGER,
        CONSTRAINT          cp_Empleados PRIMARY KEY ( cod_empleado )
);

CREATE TABLE Conductores (
        cod_empleado        INTEGER,
        horas_Act           INTEGER,
        horas_max           INTEGER,
        CONSTRAINT          cp_Conductores PRIMARY KEY (cod_empleado),
        CONSTRAINT          ca_Conductores FOREIGN KEY (cod_empleado)
                            REFERENCES Empleados
                            ON DELETE RESTRICT ON UPDATE RESTRICT
);

CREATE TABLE Revisores (
        cod_empleado        INTEGER,
        turno               VARCHAR(10),
        cargo               VARCHAR(20),
        CONSTRAINT          cp_Revisores PRIMARY KEY (cod_empleado),
        CONSTRAINT          ca_Revisores FOREIGN KEY (cod_empleado)
                            REFERENCES Empleados
                            ON DELETE RESTRICT ON UPDATE RESTRICT
);
CREATE TABLE Trenes (
        cod_tren            SERIAL,
        nombre              VARCHAR(20),
        origen              VARCHAR(20),
        destino             VARCHAR(20),
        duracion            VARCHAR(20),
        cod_empleado        INTEGER,
        horario             VARCHAR(200),
        CONSTRAINT          cp_Trenes PRIMARY KEY ( cod_tren ),
        CONSTRAINT          ca_Trenes FOREIGN KEY (cod_empleado)
                            REFERENCES Conductores
                            ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

```
CREATE TABLE Controla (
        cod_tren            INTEGER,
        cod_empleado        INTEGER,
        CONSTRAINT          cp_Controla PRIMARY KEY ( cod_tren, cod_empleado),
        CONSTRAINT          ca_ControlaTren FOREIGN KEY (cod_tren)
                            REFERENCES Trenes
                            ON DELETE RESTRICT ON UPDATE RESTRICT,
        CONSTRAINT          ca_ControlaRevisores FOREIGN KEY (cod_empleado)
                            REFERENCES Revisores
                            ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

Datos Insertados:

```
--Empleados
INSERT INTO Empleados (nombre, email, edad) VALUES ('Jose Garcia', 'jgarcia@gmail.com', 35);
INSERT INTO Empleados (nombre, email, edad) VALUES ('Antonio Jose', 'ajose@gmail.com', 42);
INSERT INTO Empleados (nombre, email, edad) VALUES ('Jesus Gonzalez', 'jgonzalez@gmail.com', 38);
INSERT INTO Empleados (nombre, email, edad) VALUES ('Carlos Remon', 'cremon@gmail.com', 47);
INSERT INTO Empleados (nombre, email, edad) VALUES ('Juan Jose', 'jjose@gmail.com', 29);
INSERT INTO Empleados (nombre, email, edad) VALUES ('Alex Beltran', 'abeltran@gmail.com', 22);
--Conductores
INSERT INTO Conductores VALUES (1, 8, 12);
INSERT INTO Conductores VALUES (2, 3, 12);
INSERT INTO Conductores VALUES (5, 4, 12);
--Revisores
INSERT INTO Revisores VALUES (3, 'dia', 'Responsable');
INSERT INTO Revisores VALUES (4, 'noche', 'Revisor');
INSERT INTO Revisores VALUES (6, 'dia', 'Revisor');
--Trenes
INSERT INTO Trenes (nombre, origen, destino, duracion, cod_empleado, horario) VALUES ('ALVIA
04111','Castellon','Madrid','3 h. 3 min.',1,'Tren Fuera de horario');
INSERT INTO Trenes (nombre, origen, destino, duracion, cod_empleado, horario) VALUES ('EUROMED
01091','Castellon','Valencia','49 min.',1,'Tren Fuera de horario');
INSERT INTO Trenes (nombre, origen, destino, duracion, cod_empleado, horario) VALUES ('EUROMED
01472','Castellon','Barcelona','2 h. 17 min.',2,'Tren Fuera de horario');
INSERT INTO Trenes (nombre, origen, destino, duracion, cod_empleado, horario) VALUES ('ALVIA
04111','Madrid','Castellon','3 h. 10 min.',5,'Tren Fuera de horario');
--Controla
INSERT INTO Controla VALUES (1, 3);
INSERT INTO Controla VALUES (2, 4);
INSERT INTO Controla VALUES (4, 6);
```