Examen de Bases de datos
26 de Enero, 2017

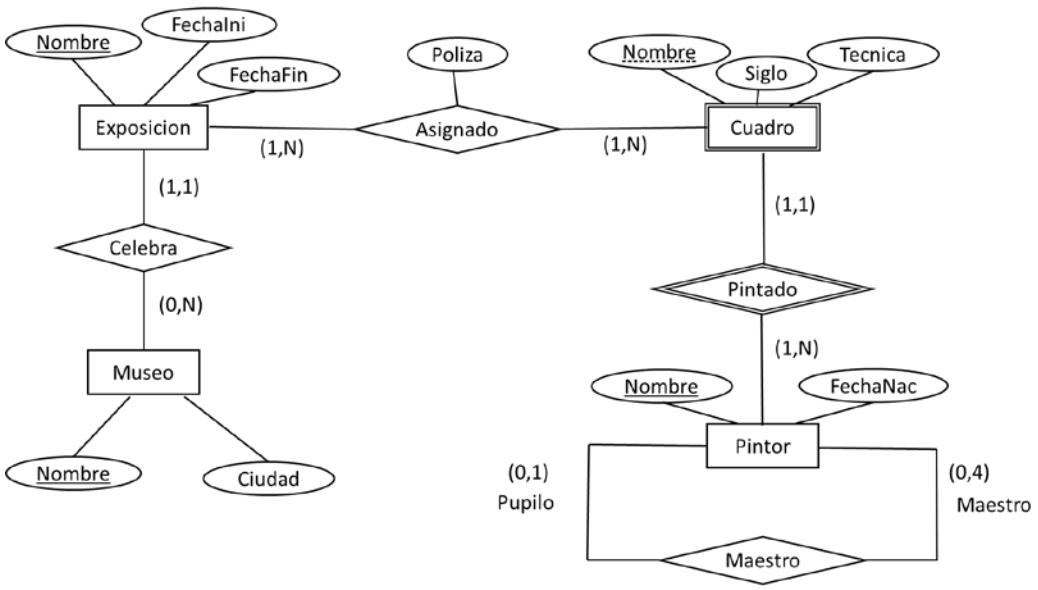
APELLIDOS, NOMBRE:

Las soluciones deben entregarse en las hojas que se os han entregado y dentro del espacio designado para ellas. Podéis utilizar papel de sucio, pero las respuestas definitivas deben indicarse en las hojas de los enunciados. NO se puede usar lápiz ni boli rojo.

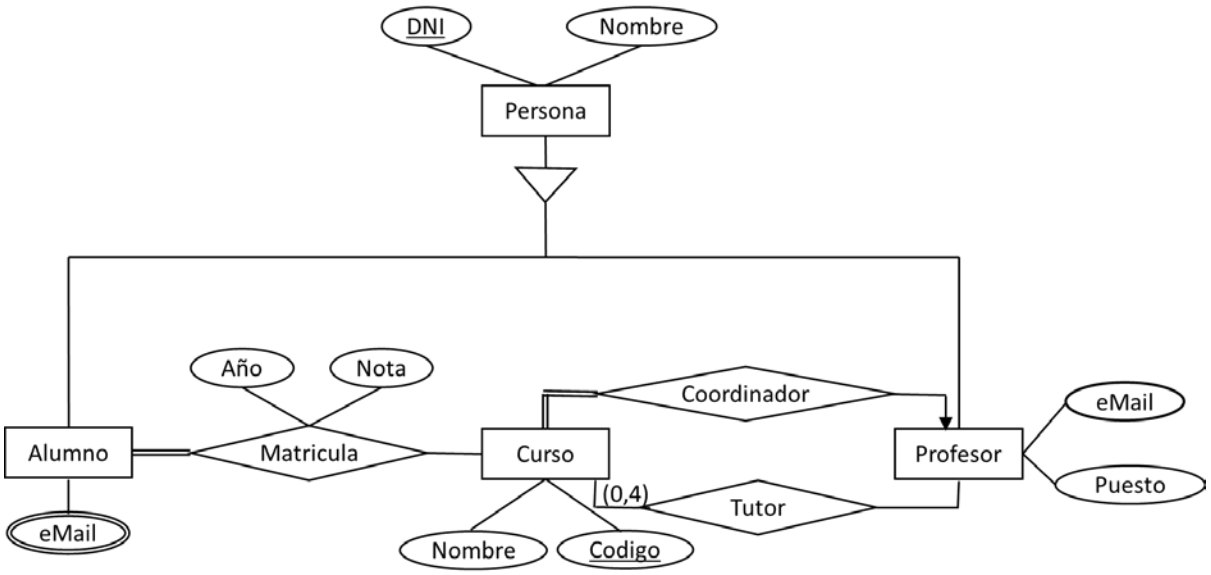
1. (1,5 puntos) Se desea diseñar la base de datos para la gestión de las exposiciones temporales que se realizan en todos los museos de España. Los requisitos de los que disponemos para diseñar dicha base de datos son los siguientes:

- De cada cuadro se registrará el nombre, el siglo en el que fue pintado y la técnica utilizada (óleo, acuarela, acrílico o carboncillo). Habrá que tener en cuenta que hay títulos de cuadros muy ampliamente utilizados (por ejemplo, muchos pintores tienen un autorretrato entre sus obras). Sin embargo, los títulos no se repiten entre las obras de un mismo pintor.
- Todos los cuadros son pintados por un pintor. De cada pintor guardaremos: el nombre (único) y su fecha de nacimiento.
- Un pintor puede tener (o no) a otro como maestro y un pintor puede ser maestro como máximo de 4 pintores. Solamente almacenaremos pintores que tengan cuadros en nuestra base de datos.
- De cada exposición se desea almacenar el nombre (único) y las fechas de inicio y fin, así como el museo en el que se realizan. Todas las exposiciones se celebran en algún museo de los registrados en la base de datos y no son itinerantes, es decir, solo se celebran en un museo.
- De cada museo se registrará el nombre (único para cada museo) y la ciudad en la que se encuentra. En la base de datos pueden aparecer museos que no tengan aún ninguna exposición temporal.
- Cada exposición temporal tiene asignados los cuadros que en ella se exhiben (al menos uno). Todos los cuadros que aparecen en la base de datos están asignados, al menos, a una de las exposiciones almacenadas. Cada vez que un cuadro se asigna a una exposición temporal es necesario suscribir una póliza de seguro cuyo número necesitamos almacenar.

Crea el esquema Entidad-Relación de la base de datos descrita. Debes de incluir atributos, claves y restricciones de cardinalidad y participación.



2. (1,5 puntos) Se desea diseñar una base para la gestión de una academia. A partir de los requisitos proporcionados por el director de la academia se ha creado el siguiente diagrama de Entidad-Relación:



Se pide:

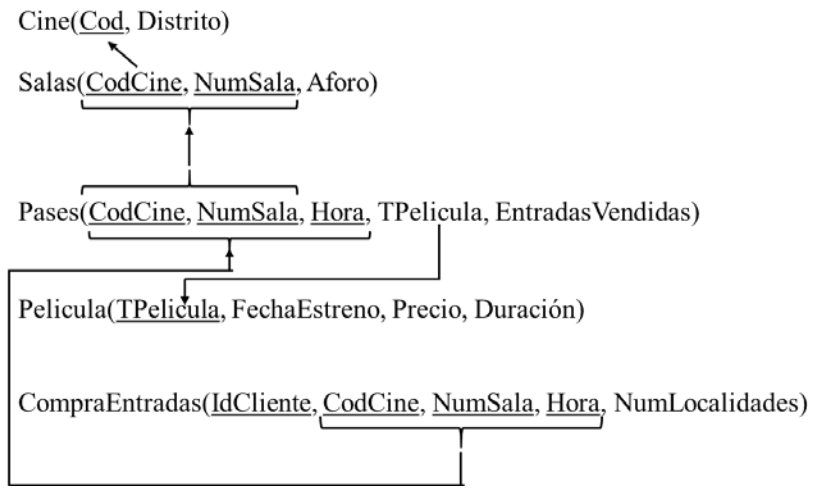
a. (1 punto) Pasa el modelo Entidad-Relación presentado a un modelo relacional haciendo uso de las transformaciones apropiadas e indicando las restricciones de integridad referencial resultantes.

Alumno (DNI, Nombre)
MailAlumno (DNI, eMail)
Profesor (DNI, Nombre, eMail, Puesto)
Curso (Codigo, Nombre, Coordinador)
Matricula (Alumno, Curso, Año, Nota*)
Tutores (Curso, Tutor)

$\prod_{DNI}(MailAlumno) \subseteq \prod_{DNI}(Alumno)$
$\prod_{Coordinador}(Curso) \subseteq \prod_{DNI}(Profesor)$
$\prod_{Alumno}(Matricula) \subseteq \prod_{DNI}(Alumno)$
$\prod_{Curso}(Matricula) \subseteq \prod_{Codigo}(Curso)$
$\prod_{Curso}(Tutores) \subseteq \prod_{Codigo}(Curso)$
$\prod_{Tutor}(Tutores) \subseteq \prod_{DNI}(Profesor)$

b. (0,5 puntos) Indica qué restricciones de integridad se han perdido en la transformación.
   - Un curso puede tener como máximo 4 tutores.
   - Todos los alumnos que aparecen en la base de datos están matriculados de, al menos, un curso

3. (6 puntos) Dado el siguiente modelo relacional de una base de datos que almacena la gestión de una cadena de cines:



a. (0,5 puntos) Escribe una consulta en lenguaje SQL que muestre el título de las películas de más de 90 minutos de duración que se proyectan en algún cine del distrito 24321.

```
SELECT DISTINCT TPelicula
FROM (PELICULA JOIN PASES USING (TPELICULA))
      JOIN CINE ON CINE.COD=PASES.CODCINE
WHERE DISTRITO=24321 AND DURACION > 90;
```

b. (0.5 puntos) Escribe una consulta en lenguaje SQL que muestre el título de las películas de más de 90 minutos para las que se ofrecen en el mismo distrito más de 300 butacas de aforo.

```
SELECT distinct P.TPelicula
FROM ((Cine C JOIN Pases ON C.COD=Pases.CodCine) JOIN Salas S
       USING (CodCine, NumSala)) JOIN Pelicula P USING (TPelicula)
WHERE P.DURACION > 90
GROUP BY P.TPelicula, C.Distrito
HAVING SUM(Aforo) > 300;
```

c. (0.5 puntos) Escribe una consulta en lenguaje SQL que muestre para todos los cines el número de pases que no tienen ninguna entrada vendida. Si algún cine tiene entradas vendidas para todos sus pases se debe de mostrar un 0.

```
SELECT C.Cod, NVL(NumSalas,0)
FROM Cine C LEFT OUTER JOIN (SELECT CodCine, COUNT(*) as NumSalas
                             FROM Pases
                             WHERE EntradasVendidas = 0
                             GROUP BY CodCine) P ON C.Cod = P.CodCine;
```

d. (0.75 puntos) Escribe una consulta en lenguaje SQL que muestre el código de los cines en los que solamente se proyecten películas estrenadas en el 2016.

```
SELECT Cod FROM CINE C
WHERE NOT EXISTS (SELECT *
                  FROM PASES JOIN PELICULA USING (TPELICULA)
```

```
                WHERE CodCine = C.Cod
                   AND EXTRACT (YEAR FROM FECHAESTRENO) <> 2016);

SELECT DISTINCT CodCine
FROM SALA JOIN PELICULA ON SALA. TPELICULA = PELICULA.TPELICULA
WHERE EXTRACT (YEAR FROM FECHAESTRENO) = 2016)
 AND CodCine NOT IN (SELECT DISTINCT CodCine
                     FROM SALA JOIN PELICULA ON SALA.
                     TPELICULA=PELICULA.TPELICULA
                     WHERE EXTRACT (YEAR FROM FECHAESTRENO) <> 2016));
```

e. (0.75 puntos) Escribe una consulta en lenguaje SQL que muestre los distritos en los que se proyectan el mayor número de películas distintas.

```
SELECT C.Distrito
FROM Pases P JOIN Cine C ON P.CodCine = C.Cod
GROUP BY C.Distrito
HAVING COUNT(DISTINCT P.TPelicula) >= ALL (SELECT COUNT(DISTINCT P.TPelicula)
                                           FROM Pases P JOIN Cine C
                                                ON P.CodCine = C.Cod
                                           GROUP BY distrito);

SELECT distrito
FROM Pases P JOIN Cine ON P.CodCine = C.Cod
GROUP BY distrito
HAVING COUNT(DISTINCT TPelicula)= (SELECT MAX(COUNT(DISTINCT TPelicula))
                                  FROM Pases P JOIN Cine C ON P.CodCine = C.Cod
                                  GROUP BY distrito);
```

f. (1,5 puntos) Crea un procedimiento almacenado que reciba por argumento el código de un cine y escriba por consola el código del cine, el número de salas y su aforo total (suma del aforo de todas sus salas). A continuación se deben listar los pases de dicho cine en orden cronológico incluyendo la hora, la sala, la película y el número de localidades libres. Si el cine recibido por parámetro no se encuentra en la base de datos el procedimiento debe escribir únicamente el siguiente mensaje: 'El cine xxx no existe'.

```
CREATE OR REPLACE PROCEDURE pasesCine(Cine Salas.CodCine%TYPE) AS

DATOS_CINE VARCHAR(300);

CURSOR CPases IS
    SELECT P.Hora,P.NumSala,P.TPelicula,S.Aforo-P.EntradasVendidas LocLibres
    FROM Salas S JOIN Pases P ON (S. CodCine = P. CodCine
                                  AND S.NumSala = P.NumSala)
    WHERE S.CodCine = Cine
    ORDER BY P.Hora;

BEGIN
  DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
  SELECT 'Cine: '|| TRIM(Cine)|| ', Num Salas: ' || COUNT(*)
    || ', Aforo Total: '|| SUM(Aforo)
  INTO DATOS_CINE
  FROM Salas
  WHERE CodCine = Cine;
```

```
  DBMS_OUTPUT.PUT_LINE(DATOS_CINE);

  DBMS_OUTPUT.PUT_LINE('-----------------------------------------');

  FOR PASE IN CURSORPASES
  LOOP
    DBMS_OUTPUT.PUT_LINE('Hora: ' || PASE.HORA ||', Sala: ' || PASE.NUMSALA ||
', Pelicula: ' || PASE.TPELICULA || ', Localidades Libres: ' ||
PASE.LOCLIBRES);
  END LOOP;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('El cine ' || Cine || ' no existe');
END;
```

g. (1.5 puntos) Escribe un disparador que mantenga la columna EntradasVendidas de la tabla Pases actualizado cuando se vendan, cancelen o se cambie el número de localidades de una compra de entradas en la tabla CompraEntradas.

```
CREATE OR REPLACE TRIGGER EntradasVendidas
AFTER INSERT OR DELETE OR UPDATE OF NumLocalidades ON CompraEntradas
FOR EACH ROW
BEGIN
      IF DELETING THEN
            UPDATE Pases
            SET EntradasVendidas = EntradasVendidas - :OLD.NumLocalidades
            WHERE CodCine = :OLD.CodCine
              AND NumSala= :OLD.NumSala
               AND Hora = :OLD.Hora;
      ELSIF INSERTING THEN
            UPDATE Pases
            SET EntradasVendidas = EntradasVendidas + :NEW.NumLocalidades
            WHERE CodCine = :NEW.CodCine
              AND NumSala= :NEW.NumSala
               AND Hora = :NEW.Hora;
      ELSE
            UPDATE Pases
            SET EntradasVendidas = EntradasVendidas
                                   -  :OLD.NumLocalidades
                                   + :NEW.NumLocalidades
            WHERE CodCine = :OLD.CodCine
              AND NumSala= :OLD.NumSala
               AND Hora = :OLD.Hora;
      END IF;
END;
```

4. (1 punto) Dada la tabla **VENTAS(TPelicula, EntradasVendidas)** vacía considera la ejecución de la siguiente lista de instrucciones SQLDeveloper asumiendo que **autocommit= off**

```
savepoint paso_uno;
INSERT INTO VENTAS  VALUES ('Blancanitos', 200);

-- paso 1 –

savepoint paso_dos;
update VENTAS
 set EntradasVendidas = EntradasVendidas + 100
where TPelicula = 'Blancanitos';

-- paso 2 –

rollback to savepoint paso_dos;

-- paso 3 –

update VENTAS
 set EntradasVendidas = EntradasVendidas + 200
where TPelicula = 'Blancanitos';
```

```
-- paso 4 –

   rollback;

-- paso 5 --

   INSERT INTO VENTAS  VALUES ('Blancanitos', 1000);
   update VENTAS
    set EntradasVendidas = EntradasVendidas + 300
   where TPelicula = 'Blancanitos';

-- paso 6  --

   savepoint paso_tres;
   commit;

-- paso 7 –

   create table superventas(Tpeli varchar(20), TotEntradas number(5));
   Insert into superventas values('Enanieves', 100);
   rollback to savepoint paso_tres;

-- paso 8 --

   select * from  SUPERVENTAS  where TPeli = 'Enanieves';
   rollback;

-- paso 9 --
```

a) Describe qué valor tiene EntradasVendidas para la fila 'Blancanitos' exactamente en el momento indicado por cada comentario `-- paso N --` (aunque todavía no sea un valor definitivo).
b) Indica con qué instrucción empieza cada una de las transacciones de la secuencia.
c) ¿Se produce algún error? Si lo hay, indica en qué instrucción y por qué.
d) ¿Qué tablas quedan al final de la ejecución?

Solución

```
savepoint paso_uno;  -- crea una transacción
INSERT INTO VENTAS  VALUES ('Blancanitos', 200);
-- paso 1 -- tengo 200 , empieza transacción primera

savepoint paso_dos;
update VENTAS
 set EntradasVendidas = EntradasVendidas + 100
where TPelicula = 'Blancanitos';
-- paso 2 -- tengo 300

rollback to savepoint paso_dos;
-- paso 3 -- tengo 200, sigo en transacción primera

update VENTAS
 set EntradasVendidas = EntradasVendidas + 200
where TPelicula = 'Blancanitos';
```

```
-- paso 4 -- tengo 400

rollback;
-- paso 5 – elimina insert (Blancanitos), cierra trans, no trans activa

INSERT INTO VENTAS  VALUES ('Blancanitos', 1000);
        --   empieza segunda transacción

update VENTAS
 set EntradasVendidas = EntradasVendidas + 300
where TPelicula = 'Blancanitos';
-- paso 6  -- tengo 1300

savepoint paso_tres;
commit;
-- paso 7 -- tengo 1300

create table superventas(Tpeli varchar(20), TotEntradas number(5));
   --  empieza la tercera transacción, pero termina porque tiene
   --  un commit implícito. Se queda sin transacción activa

Insert into superventas values('Enanieves', 100);
   --  empieza la cuarta transacción 7.5.3354
   --     (o tercera si no cuentas la create)

rollback to savepoint paso_tres;
-- paso 8   -- da error ORA-01086, no existe la transacción
            -- con ese save_point (y no cambia la trans)

select * from  SUPERVENTAS  where TPeli = 'Enanieves';

rollback;
-- paso 9  -- tengo 1300,
           -- elimina Enanieves (insert), termina trans
           --  no elimina la tabla superventas por ser DDL
```

Se pide:

a) Describe qué valor tiene EntradasVendidas exactamente en el momento indicado por cada comentario **-- paso N --** (aunque todavía no sea un valor definitivo).
SOLUCION en los comentarios del código

b) Indica con qué instrucción empieza cada una de las transacciones que hay.

```
-- solución:
    1ª - INSERT antes del paso 1 (savepoint paso_uno : también me vale)
    2ª - INSERT ventas (después del paso 5)
    3ª - Create table superventas (después de paso 7)
    4ª - INSERT después de paso 7
         (me vale si dicen solo una de la 3ª o la 4ª)
```

c) ¿se produce algún error? Si lo hay, indica en qué instrucción y porqué.
rollback to savepoint paso_tres; en paso 8 da error ORA-01086, no existe la transacción con ese save_point, porqué ya se confirmó

d) ¿Qué tablas quedan al final de la ejecución
Ventas y superventas, el rollback no elimina las instrucciones DDL