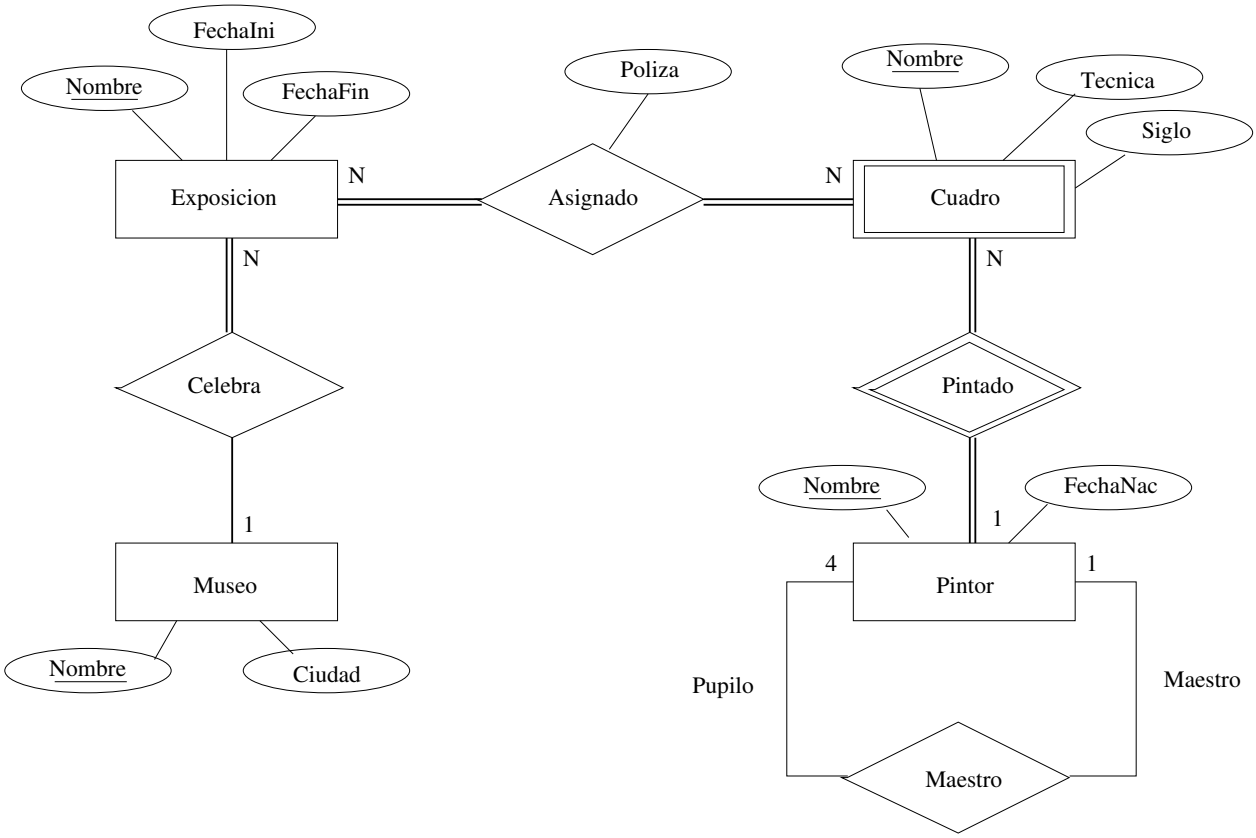
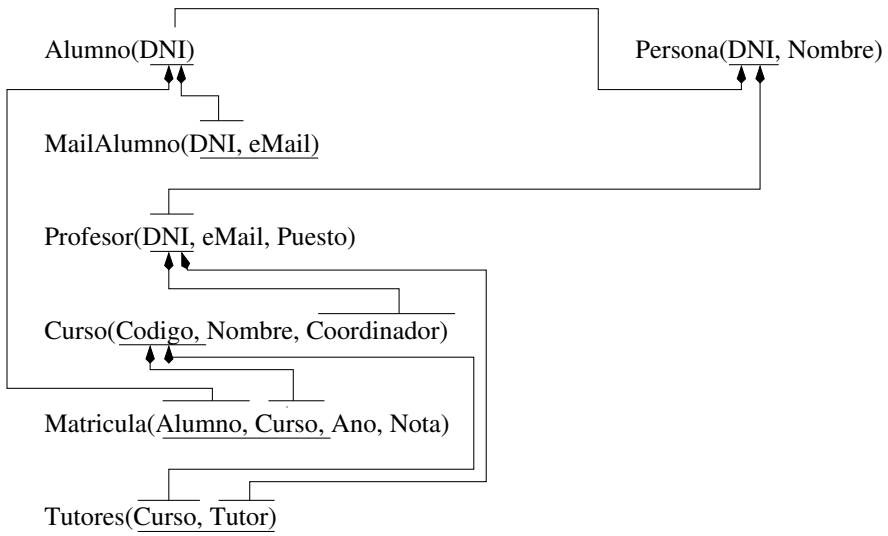
Facultad de Informática. Ingeniería en Informática / del Software / de Computadores.
Bases de datos. Curso 2016-2017. Grupos A y B. Solución examen de febrero. 26/01/2017.

Solución Ejercicio 1.



Solución Ejercicio 2.



No se puede representar la restricción de cardinalidad 4 en la relación `Tutor` ni la restricción de participación total en la relación `Matrícula`.

Solución Ejercicio 3.

```
alter session set nls_date_format = "DD/MM/YYYY HH24:MI";
-- ------------------------------------------------------------------------
-- 3.a Escribe una consulta en lenguaje SQL que muestre el titulo de las
-- peliculas de mas de 90 minutos de duracion que se proyectan en algun cine
-- del distrito 24321.
-- ------------------------------------------------------------------------

select * from pelicula;
SELECT DISTINCT pe.TPelicula
FROM pelicula pe
JOIN pases pa ON pe.TPelicula = pa.TPelicula
JOIN cine ci ON ci.cod = pa.codCine
WHERE duracion > 90 AND distrito = 24321;

-- ------------------------------------------------------------------------
-- 3.b Escribe una consulta en lenguaje SQL que muestre el titulo de las
-- peliculas de mas de 90 minutos para las que se ofrecen en el mismo distrito
-- mas de 300 butacas de aforo.
-- ------------------------------------------------------------------------

-- En esta consulta se debe utilizar WHERE para seleccionar las peliculas segun
-- su duracion y HAVING para seleccionar aquellas que tienen un aforo total
-- mayor a 300.  Observa que para indicar que las peliculas se ofrecen en el
-- mismo distrito se agrupan las filas no solo por TPelicula, sino tambien por
-- distrito.

SELECT TPelicula
FROM pelicula pe
JOIN pases pa USING (TPelicula)
JOIN salas sa USING (codCine,numSala)
JOIN cine ci ON ci.cod = codCine
WHERE pe.duracion > 90
GROUP BY TPelicula, ci.distrito
HAVING SUM(aforo) > 300;

-- ------------------------------------------------------------------------
-- 3.c Escribe una consulta en lenguaje SQL que muestre para todos los cines el
-- numero de pases que no tienen ninguna entrada vendida. Si algun cine tiene
-- entradas vendidas para todos sus pases se debe de mostrar un 0.
-- ------------------------------------------------------------------------

-- (Solucion 1: utilizando UNION)
-- Obtenemos por separado todos los cines: por un lado, los cines con algun pase
-- sin ninguna entrada vendida; por otro lado, su complementario: los cines en
-- los que *todos* los pases tienen *alguna* entrada vendida.
-- Aunque UNION ALL no es necesario porque las condiciones de las dos consultas
-- son excluyentes, se utiliza en este caso para mejorar la eficiencia.

SELECT DISTINCT c.cod, c.distrito, count(*)
FROM cine c
JOIN pases p ON c.cod = p.codCine
WHERE p.entradasVendidas = 0
GROUP BY c.cod, c.distrito
UNION ALL
SELECT cod, distrito, 0
FROM cine
WHERE cod NOT IN (SELECT codCine FROM pases WHERE entradasVendidas = 0);
```

```
-- (Solucion 2: utilizando LEFT JOIN)
-- Observa que se debe utilizar una subconsulta en la clausula FROM.

SELECT DISTINCT c.cod, c.distrito, NVL(numPases,0)
FROM cine c
LEFT JOIN (
  SELECT p.codCine AS codCine, COUNT(*) AS numPases
  FROM pases p
  WHERE p.entradasVendidas = 0
  GROUP BY p.codCine)
ON c.cod = codCine;

-- ------------------------------------------------------------------------
-- 3.d Escribe una consulta en lenguaje SQL que muestre el codigo de los cines
-- en los que solamente se proyecten peliculas estrenadas en el 2016.
-- ------------------------------------------------------------------------

-- Observa que este apartado contiene una cuantificacion universal: la palabra
-- "solamente" indica que *todas* las peliculas proyectadas sean de 2016.
-- Por ello es necesario negar la condicion y seleccionar los que NO cumplen la
-- condicion negada.

SELECT codCine
FROM pases
JOIN pelicula USING (TPelicula)
WHERE EXTRACT(YEAR FROM fechaEstreno) = 2016
AND codCine NOT IN (
  SELECT codCine
  FROM pases
  JOIN pelicula USING (TPelicula)
  WHERE EXTRACT(YEAR FROM fechaEstreno) != 2016);

-- ------------------------------------------------------------------------
-- 3.e Escribe una consulta en lenguaje SQL que muestre los distritos en los
-- que se proyectan el mayor numero de peliculas distintas.
-- ------------------------------------------------------------------------

-- En esta consulta se debe utilizar DISTINCT dentro de la funcion de agregacion
-- COUNT para poder contar el numero de peliculas distintas de un mismo
-- distrito.  Ademas debe utilizarse HAVING para expresar una condicion sobre
-- el numero de peliculas del distrito.  Por ultimo, esta condicion debe
-- incluir una subconsulta, pues debe mostrar aquellos cines que proyecten el
-- maximo numero de peliculas distintas (pueden ser varios, como es el caso de
-- los datos de prueba incluidos en el fichero).

-- (Solucion 1: utilizando >= ALL)

SELECT c.distrito, COUNT(DISTINCT p.TPelicula)
FROM cine c
JOIN pases p ON c.cod = p.codCine
GROUP BY c.distrito
HAVING COUNT(DISTINCT p.TPelicula) >= ALL (
  SELECT COUNT(DISTINCT p.TPelicula)
  FROM cine c
  JOIN pases p ON c.cod = p.codCine
  GROUP BY c.distrito);
```

```
-- (Solucion 2: utilizando MAX(COUNT(...)) )
-- En lugar del operador >= ALL, tambien se puede utilizar la consulta
-- alternativa que devuelva el maximo numero de peliculas.

SELECT c.distrito, COUNT(DISTINCT p.TPelicula)
FROM cine c
JOIN pases p ON c.cod = p.codCine
GROUP BY c.distrito
HAVING COUNT(DISTINCT p.TPelicula) = (
  SELECT MAX(COUNT(DISTINCT p.TPelicula))
  FROM cine c
  JOIN pases p ON c.cod = p.codCine
  GROUP BY c.distrito);

-- ------------------------------------------------------------------------
-- 3.f  Crea un procedimiento almacenado que reciba por argumento el codigo de
-- un cine y escriba por consola el codigo del cine, el numero de salas y su
-- aforo total (suma del aforo de todas sus salas). A continuacion se deben
-- listar los pases de dicho cine en orden cronologico incluyendo la hora, la
-- sala, la pelicula yÂ el numero de localidades libres. Si el cine recibido por
-- parametro no se encuentra en la base de datos el procedimiento debe escribir
-- unicamente el siguiente mensaje: 'El cine xxx no existe'.
-- ------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE pasesCine(p_Cine Salas.CodCine%TYPE) AS
  v_datosCine 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 = p_Cine
    ORDER BY p.Hora, p.numSala;

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

  DBMS_OUTPUT.PUT_LINE(v_datosCine);
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
  DBMS_OUTPUT.PUT_LINE(' Hora   Sala  Pelicula                Loc.libres');
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');

  FOR rPase IN CPases
  LOOP
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(rPase.HORA,'HH24:MI') || '  '
      || TO_CHAR(rPase.numSala,'9999') || '  '
      || RPAD(rPase.TPelicula,25) || '   ' || TO_CHAR(rPase.locLibres,'9G999'));
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');

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

```
/
-- Bloque anonimo para probar el procedimiento.
SET SERVEROUTPUT ON;
BEGIN
  pasesCine(3);
END;
/

-- ------------------------------------------------------------------------
-- 3.g Escribe un disparador que mantenga la columna EntradasVendidas de la
-- tabla Pases actualizado cuando se vendan, cancelen o se cambie el numero de
-- localidades de una compra de entradas en la tabla CompraEntradas.
-- ------------------------------------------------------------------------

CREATE OR REPLACE TRIGGER EntradasVendidas
AFTER INSERT OR DELETE OR UPDATE 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
    -- En el enunciado no se indica como se debe hacer este caso.
    -- Si lo que se realiza es un cambio de numero de sala o de
    -- cine, las filas sobre las que se hacen las operaciones son
    -- distintas.  Se pueden cubrir estos casos con dos
    -- sentencias UPDATE.
    UPDATE Pases
    SET EntradasVendidas = EntradasVendidas - :OLD.NumLocalidades
    WHERE CodCine = :OLD.CodCine
      AND NumSala = :OLD.NumSala
      AND Hora    = :OLD.Hora;
    UPDATE Pases
    SET EntradasVendidas = EntradasVendidas + :NEW.NumLocalidades
    WHERE CodCine = :NEW.CodCine
      AND NumSala = :NEW.NumSala
      AND Hora    = :NEW.Hora;
  END IF;
END;
/
-- Para probarlo insertamos, eliminamos y modificamos CompraEntradas.
alter session set nls_date_format = 'HH24:MI:SS';
DELETE FROM CompraEntradas;
PROMPT Pruebas realizadas sobre el trigger:
PROMPT Antes de modificar CompraEntradas:
SELECT * FROM pases WHERE codCine = 1 AND numSala = 1 AND Hora = TO_DATE('18:00:00');

INSERT INTO CompraEntradas (idCliente, codCine, numSala, Hora, TPelicula, NumLocalidades)
  VALUES (100, 1, 1, to_date('18:00:00'), 'Lo que el viento se llevo', 3);
```

```
PROMPT Despues de que el cliente 100 compre 3 entradas:
SELECT * FROM pases WHERE codCine = 1 AND numSala = 1 AND Hora = TO_DATE('18:00:00');

INSERT INTO CompraEntradas (idCliente, codCine, numSala, Hora, TPelicula, NumLocalidades)
  VALUES (200, 1, 1, to_date('18:00:00'), 'Lo que el viento se llevo', 2);

PROMPT Despues de que el cliente 200 compre 2 entradas:
SELECT * FROM pases WHERE codCine = 1 AND numSala = 1 AND Hora = TO_DATE('18:00:00');

UPDATE CompraEntradas SET NumLocalidades = 5 WHERE idCliente = 100;

PROMPT Despues de que el cliente 100 cambie a 5 entradas:
SELECT * FROM pases WHERE codCine = 1 AND numSala = 1 AND Hora = TO_DATE('18:00:00');

DELETE FROM CompraEntradas WHERE idCliente = 200;

PROMPT Despues de cancelar las entradas compradas por el cliente 200:
SELECT * FROM pases WHERE codCine = 1 AND numSala = 1 AND Hora = TO_DATE('18:00:00');
```

Solución Ejercicio 4.

```
-- ------------------------------------------------------------------------
-- Dada la tabla VENTAS(TPelicula, EntradasVendidas) vacia considera
-- la ejecucion de la siguiente lista de instrucciones SQLDeveloper
-- asumiendo que autocommit= off.

-- a) Describe que valor tiene EntradasVendidas para la fila
-- 'Blancanitos' exactamente en el momento indicado por cada
-- comentario -- paso N -- (aunque todavia no sea un valor
-- definitivo).

-- b) Indica con que instruccion empieza cada una de las transacciones
-- de la secuencia.

-- c) Se produce algun error? Si lo hay, indica en que instruccion y
-- por que.

-- d) Que tablas quedan al final de la ejecucion?
-- ------------------------------------------------------------------------

savepoint paso_uno;
INSERT INTO VENTAS  VALUES ('Blancanitos', 200);  -- b) Inicio de 1a transaccion.

-- paso 1  -- a) fila 'Blancanitos' con valor 200.

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

-- paso 2  -- a) fila 'Blancanitos' con valor 300.

rollback to savepoint paso_dos;

-- paso 3  -- a) fila 'Blancanitos' con valor 200.

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

-- paso 4  -- a) fila 'Blancanitos' con valor 400.

rollback;

-- paso 5  -- a) Deshace todos los cambios. No existe fila 'Blancanitos'
           -- (Fin de transaccion.  No hay transaccion activa).

INSERT INTO VENTAS  VALUES ('Blancanitos', 1000); -- b) Inicio 2a transaccion.

update VENTAS
 set EntradasVendidas = EntradasVendidas + 300
where TPelicula = 'Blancanitos';

-- paso 6  -- a) fila 'Blancanitos' con valor 1300.

savepoint paso_tres;
commit;
           -- (Fin de transaccion.  No hay transaccion activa).
```

```
-- paso 7  -- a) fila 'Blancanitos' con valor 1300.

create table superventas(Tpeli varchar(20), TotEntradas number(5));
   --  b) Inicio 3a transaccion, pero termina porque tiene
   --  un commit implicito. Se queda sin transaccion activa.

Insert into superventas values('Enanieves', 100); --  b) Inicio 4a transaccion.

rollback to savepoint paso_tres;
-- paso 8  -- a) fila 'Blancanitos' con valor 1300.
           -- c) Se produce error ORA-01086, no existe la transaccion
           --    con ese save_point (y no cambia la transaccion).

select * from  SUPERVENTAS  where TPeli = 'Enanieves';

rollback;
-- paso 9  -- a) fila 'Blancanitos' con valor 1300.
           -- Elimina los cambios DML desde el paso 7.
           -- (Fin de transaccion.  No hay transaccion activa).
           -- d) Los rollback no eliminan ninguna tabla por ser DDL.
           --    Permanecen ambas tablas: VENTAS y SUPERVENTAS.
```