

Final de Base de Datos

Ej 1	Ej 2	Ej 3.1	Ej 3.2	Ej 3.	Ej 4.	Ej 4.	Nota
/2	/2	/1	/1	/1	/2	/1	/10

Adicionales:

Ej 5.1	Ej 5.2	Ej 5.3	Ej 5.4
/1	/1	/1	/1

- **Los ejercicios se suben a Campus en archivos separados en formato apropiado, .png, .jpg pero en una única entrega (numerar los archivos como 3.1.jpg, 3.2.jpg, etc). Pueden directamente embeberse en el doc o bien zippearlos.**
- **La última entrega realizada dentro del horario que dura la actividad será corregida.**
- **Ser sucintos en las explicaciones solicitadas. Es decir, explicar exactamente lo que se les pregunta. Con 3 o 4 renglones, si no se dispersan, es más que suficiente.**
- **El examen dura 1 h y 20 min.**

Si eligen reemplazar alguno de estos ítems (son independientes entre sí): 1, 2, 3.1, 3.2, 3.3, 4.1, 4.2 entonces, dejar claro cuál de los adicionales los reemplaza.

Ejemplo:

**reemplazo el ejercicio 2 (que vale 2 puntos) por los ejercicios 5.1 y 5.3
reemplazo el ejercicio 3.2 (vale 1 punto) por el ejercicio 5.4**

Reemplazo los ejercicios 3.1, 3.2 y 3.3 (1 punto cada uno) por los ejercicios adicionales 5.1, 5.2 y 5.3 (1 punto cada uno).

**Ejercicio 1**

Dado el esquema de relación $R = (A, B, C, D, E, F)$ con
 $\text{Dep} = \{A \rightarrow B, BCD \rightarrow E, E \rightarrow F\}$

Para cualquier instancia r de R , se realizan las siguientes consultas:

```
SELECT COUNT(distinct A)
FROM r
```

Y se obtiene como mucho el valor 2

```
SELECT COUNT(distinct C)
FROM r
```

Y se obtiene como mucho el valor 2

```
SELECT COUNT(distinct D)
FROM r
```

Y se obtiene como mucho el valor 2

Si se ejecutara

```
SELECT COUNT(distinct F)
FROM r
```

Se pide indicar cuál es el máximo valor que podría obtenerse. Justificar claramente el razonamiento. No se considera válida una respuesta sin justificación o mal justificada.

Rta 1

Hay como mucho 2 valores de A, 2 valores de C y 2 valores de D.
 Cabe destacar que si tenemos $X \rightarrow Y$, como es una referencia lineal, considero que Y puede tener como máximo cantidad igual de valores que X .
 Si quisiéramos saber cuántos valores distintos de F puede haber, primero deberíamos averiguar cuántas E puede haber, dado que $E \rightarrow F$ y es la única dependencia en la cual se encuentra presente F, por lo cual F depende directamente de E. Para esto nos fijamos en $BCD \rightarrow E$, y las posibles combinaciones entre B, C y D. A su vez, sabemos que $A \rightarrow B$, por lo que es posible que aparezcan como máximo dos apariciones de B distintas, ligadas a las de A. Esto nos lleva a que hay como máximo 2 valores de B, C y D. Finalmente vemos que como $BCD \rightarrow E$, y cada uno de los atributos de la izquierda puede tomar dos valores, nos queda como un ejercicio de conteo que hay $2 \times 2 \times 2$ posibilidades dado el conjunto que se presenta. A partir de esto nos damos cuenta que E puede tomar 8 valores. Como $E \rightarrow F$, entonces F puede tomar 8 valores.



Ejercicio 2

Se tiene la siguiente tabla tabla

```
create table figura (
    tipo text NOT NULL check ( tipo IN ('circle', 'rectangle')),
    radius int check(radius > 0),
    width int check(width > 0),
    height int check(height > 0)
)
```

Agregar la/s setencia/s SQL que permite validar que si una tupla de figura:

- representa un **círculo**, entonces su width y height están en null, pero su radius no está en null.
- representa un **rectángulo**, entonces radius está en null, pero su width and height no están en null.

Ejemplos:

está tupla **no debería estar** en la tabla: ('circle', null, null, null)

está tupla **no debería estar** en la tabla: ('circle', 1, 2, null)

está tupla **no debería estar** en la tabla: ('circle', 3, null, 5)

está tupla **no debería estar** en la tabla: ('circle', 3, 1, 9)

está tupla **podría estar en la tabla**: ('circle', 3, null, null)

está tupla **no debería estar** en la tabla: ('rectangle', null, null, null)

está tupla **no debería estar** en la tabla: ('rectangle', null, null, 9)

está tupla **no debería estar** en la tabla: ('rectangle', null, 5, null)

está tupla **no debería estar** en la tabla: ('rectangle', 3, 5, null)

está tupla **no debería estar** en la tabla: ('rectangle', 3, null, 5)

está tupla **no debería estar** en la tabla: ('rectangle', 3, 5, 5)

está tupla **podría estar en la tabla**: ('rectangle', null, 5, 5)

Aclaración: se puede directamente “agregar código en el create table anterior” o código adicional que asegure que las tuplas son correctas y nunca dejan a la tabla en un estado inconsistente respecto a las reglas explicadas.

Rta 2

```
create table figura (
    tipo text NOT NULL check ( tipo IN ('circle', 'rectangle')),
    radius int check(radius > 0),
    width int check(width > 0),
    height int check(height > 0)
    CHECK (
        ((tipo = 'circle') AND (radius IS NOT NULL) AND (width IS NULL) AND (height IS NULL))
        OR
        ((tipo = 'rectangulo') AND (radius IS NULL) AND (width IS NOT NULL) AND (height IS NOT NULL)))
);
```

Ejercicio 3

Dado el siguiente esquema (todas las columnas son not null). Se indicaron los PKs. La relación R refiere a B y C en los atributos b1+b2 y c1, respectivamente

B		
<u>b1</u>	<u>b2</u>	b3
5	50	500
6	60	600
7	70	800
8	50	900

C	
<u>c1</u>	c2
10	100
20	200
30	300

R				
<u>b1</u>	<u>b2</u>	<u>c1</u>	x	y
5	50	10	2018	A
5	50	10	2009	B
5	50	20	2018	B
6	60	10	2020	A
6	60	20	2020	A
6	60	30	2020	A
8	50	10	2019	A
8	50	30	2019	B

3.1) Cuántas y cuáles tuplas se obtienen al ejecutar la siguiente consulta SQL (mostrar nombres de las columnas también).

```
SELECT B.b1, B.b2
FROM B
WHERE NOT EXISTS (SELECT *
                   FROM C
                   WHERE C.c1 <= 20 AND NOT EXISTS (SELECT *
                                                   FROM R
                                                   WHERE R.b1 = B.b1 AND R.b2=B.b2 AND
                                                       C.c1 = R.c1)))
```

Rta 3.1
REEMPLAZO POR EL 5.1

3.2) Escribir una consulta del **Algebra Relacional equivalente a 3.1, es decir, que permita obtener lo mismo que devuelve la consulta 3.1 para CUALQUIER INSTANCIA (no solo la del ejemplo).**

Rta 3.2
REEMPLAZO POR EL 5.2



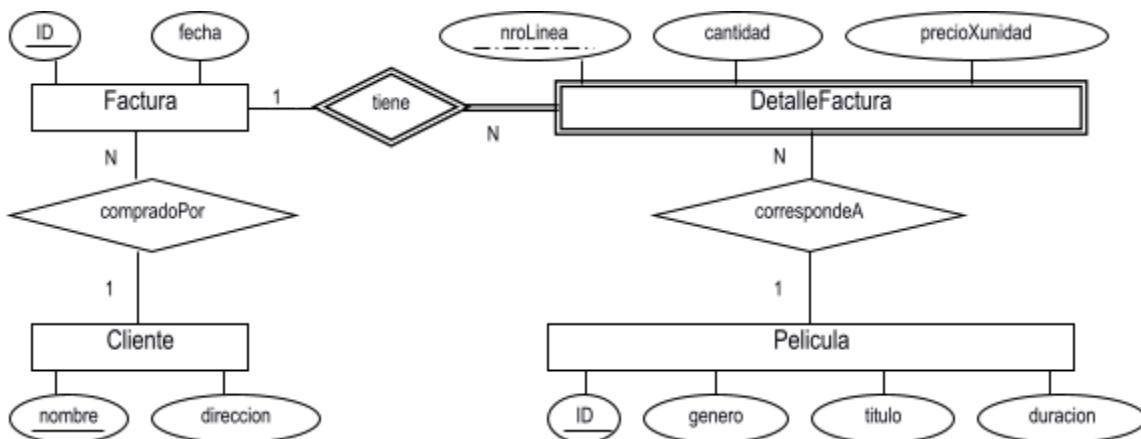
3.3) Escribir una consulta **SQL equivalente a 3.1**, es decir, que permita obtener lo mismo que devuelve la consulta 3.1 para CUALQUIER INSTANCIA (no solo la del ejemplo), pero que no utilice ninguna de estas 2 cláusulas: "NOT EXISTS", "EXISTS"

Rta 3.3

REEMPLAZO POR EL 5.3

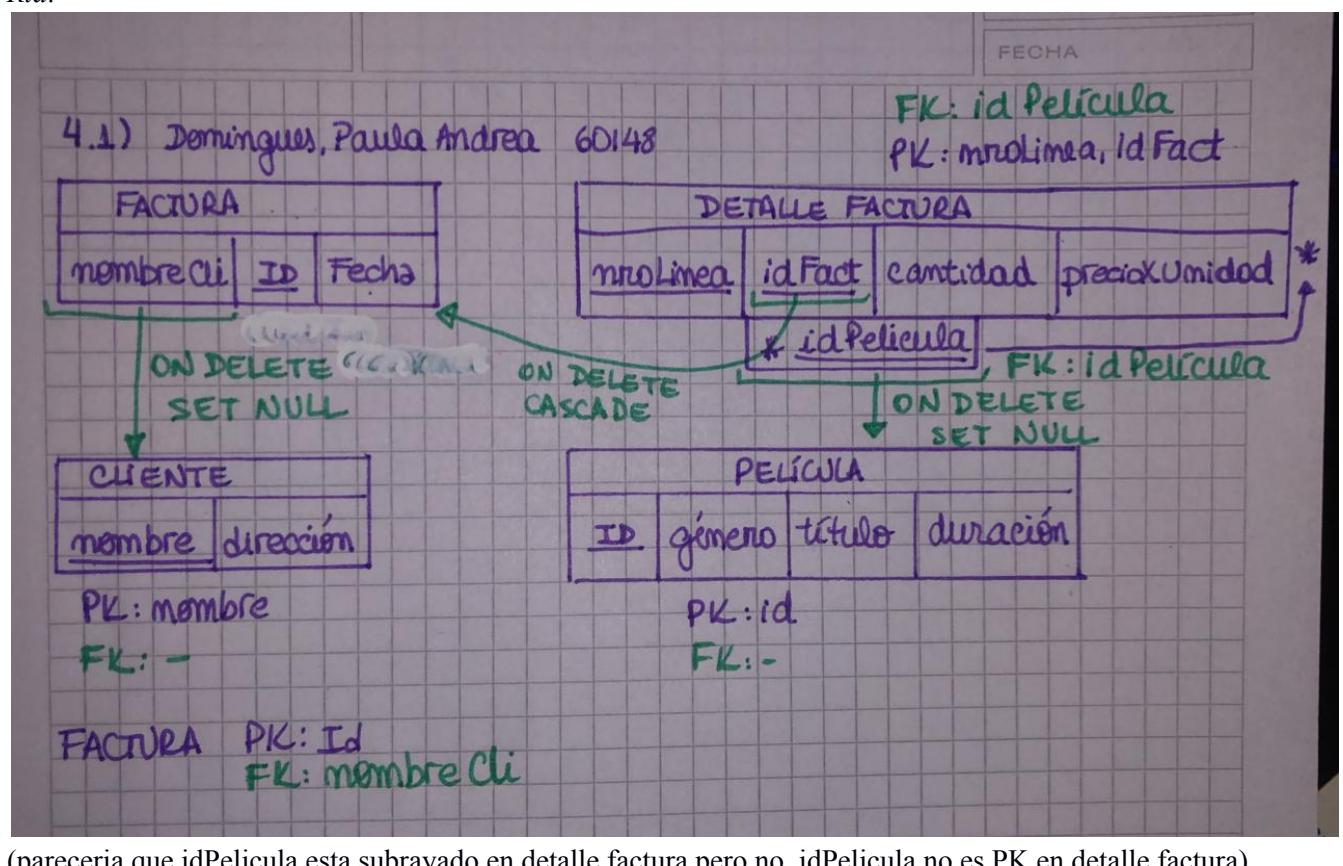
Ejercicio 4

El siguiente diagrama de E/R representa las películas de la *Warner Bros* y su comercialización en el mercado.



- 4.1)** Siguiendo el mapeo propuesto por Teorey, a partir de este EER, **mostrar gráficamente** la estructura de las tablas que se generan. **No asumir nada que no aparezca en el diagrama.** Representar PK, FK, NOT NULL, restricciones de FK, etc.

Rta:



- 4.2)** Indicar para cada una de las afirmaciones siguientes **Verdadero o Falso y justificar** (explicar claramente y exemplificar con instancias de esquemas, etc.):

- 4.2.1)** No puede aparecer en una factura el registro de dos clientes distintos.

Rta 4.2.1

Verdadero, pues como cada factura está relacionada con un único cliente pero un cliente puede aparecer en muchas facturas distintas, no es posible que aparezca en una factura el registro de dos clientes distintos.



- 4.2.2) Los números de líneas no pueden repetirse para facturas distintas.

Rta 4.2.2

Falso, pues detalle factura representa una clave compuesta por idFact y número de línea, a su vez, numero de linea es un atributo parcial. Sabemos que idFact si es unico por lo que el número de línea puede estar repetido, lo importante es que la combinación que representa la clave no lo este y esto no ocurrira nunca dado que idFact es unico.

**Adicionales**

- 5.1)** Enunciar por lo menos 2 características por las cuales la sentencia **PreparedStatement** es preferible a **CreateStatement**.

Rta 5.1)

La principal característica por la que preparedStatement es preferible sobre createStatement es que en este es posible hacer querys parametrizadas mediante el uso del carácter ? para luego asignar diversos valores.

Luego, es mejor utilizar preparedStatement al momento de realizar varios INSERT dado que con estos se invoca una única vez el INSERT y sólo se envían cada vez los argumentos correspondientes (setXXX). De esta manera la sentencia optimizada en el DBMS se puede reutilizar. En cambio, si utilizamos createStatement para varios insert, cada vez que se realice una de estas operaciones la sentencia viaja por la red, el DBSM la compila y optimiza, por lo que son operaciones que se terminan repitiendo múltiples veces.

- 5.2)** En el ejercicio 3 se tiene una entidad B cuya clave es compuesta b1+b2 (ambos enteros). Explicar:

- a) en qué escenario sería deseable usar un atributo auto-numérico (por ejemplo, serial)
- b) cuál sería su ventaja de uso
- c) cuál es la estrategia para poblar la información de la tabla B y las relacionadas con ella.

Rta 5.2)

- a) en qué escenario sería deseable usar un atributo auto-numérico (por ejemplo, serial)

Sería deseable utilizar un atributo auto-numérico en un escenario en el cual las claves ocupen mucho espacio y a su vez haya otras tablas que referencian a una clave.

- b) cuál sería su ventaja de uso

Lo que permite utilizar un atributo auto-numérico es que aquellas tablas que deban referenciar a la misma ocupen menos espacio dado que almacenan la clave ficticia y no el atributo de gran tamaño original que ocupaba mucho mas.

- c) cuál es la estrategia para poblar la información de la tabla B y las relacionadas con ella.

Una estrategia vista en clase es, al momento de definir la tabla, indicar que habrá una secuencia (con un CREATE SEQUENCE, aquí se pueden agregar restricciones a la misma). Luego se utiliza un NEXT VAL (nombreSecuencia) para que se genere el proximo número. Por último se debe dropear la secuencia.

- 5.3)** Explicar por qué hay 2 tipos de validaciones de tipo CHECK (a nivel atributo y a nivel tupla). Ejemplificar claramente qué permite una que la otra no.

Rta 5.3)

Las validaciones a nivel tupla permiten hacer validaciones que involucren no solo a un atributo sino a además valores de otros atributos de la tupla, permiten expresar condiciones más avanzadas, comparando atributos de una misma tabla como distintas , mientras que las validaciones a nivel atributo solo permiten validar restricciones en referencia al atributo en particular, sobre el cual estamos modificando.

- 5.4)** JDBC (basado en ODBC) fue creado con el objetivo de permitir que una clase Java ya compilada (class) pueda ejecutar sus sentencias SQL conectándose



a bases de datos que soporten JDBC y a las cuales se las pueda conectar dinámicamente (en tiempo de ejecución, sin recompilar).

Obviamente cada driver de JDBC tiene un grupo de clases que se corresponden con la especificación, pero Oracle, Postgresql, DB2, etc. llaman a sus clases de diferente forma (aunque implementan la misma interface).

Para poder hacer uso de ese binding dinámico con **clases cuyo nombre no se conoce en tiempo de compilación** no se debe usar "import" en el código Java. Explicar qué estrategia hay que seguir para evitar el "import".

Rta 5.4)

Ejercicio 1

- (1) z valores de A
- (2) z valores de C
- (3) z valores de D

De (1) y de $A \rightarrow B$, podemos deducir que B toma como maximo z valores distintos. Luego, como B, C y D toman z valores distintos cada uno y $BCD \rightarrow E$, entonces la cantidad de valores que puede tomar E equivale a las distintas combinaciones posibles entre B, C y D $\Rightarrow z * z * z = 8$ combinaciones posibles. Finalmente, como E toma como maximo 8 valores y $E \rightarrow F$, entonces F tambien toma como maximo 8 valores.

Conclusion: Con esa sentencia se obtiene como mucho el valor 8.

Ejercicio 2

CREATE TABLE Figura (

```
tipo text NOT NULL CHECK (tipo IN ('circle', 'rectangle')),  
radius int CHECK (radius > 0),  
width int CHECK (width > 0),  
height int CHECK (height > 0),  
CHECK ((tipo IN 'circle' AND width IS NULL AND height IS NULL AND radius IS NOT NULL) OR  
(tipo IN 'rectangle' AND radius IS NULL AND width IS NOT NULL AND height IS NOT NULL))  
)
```

Ejercicio 3

3.1 Obtengo la siguiente tabla :

b_1	b_2
5	50
6	60

⇒ busco los valores b_1 y b_2 tales que no existan tuplas en R donde $R.b_1 = B.b_1$, $R.b_2 = B.b_2$ y $R.c_1 \leq 20$

3.2 $\Pi_{b_1, b_2} (\Pi_{b_1, b_2, c_1} (R) \% \Pi_{c_1} (\sigma_{c_1 \leq 20} (C)))$

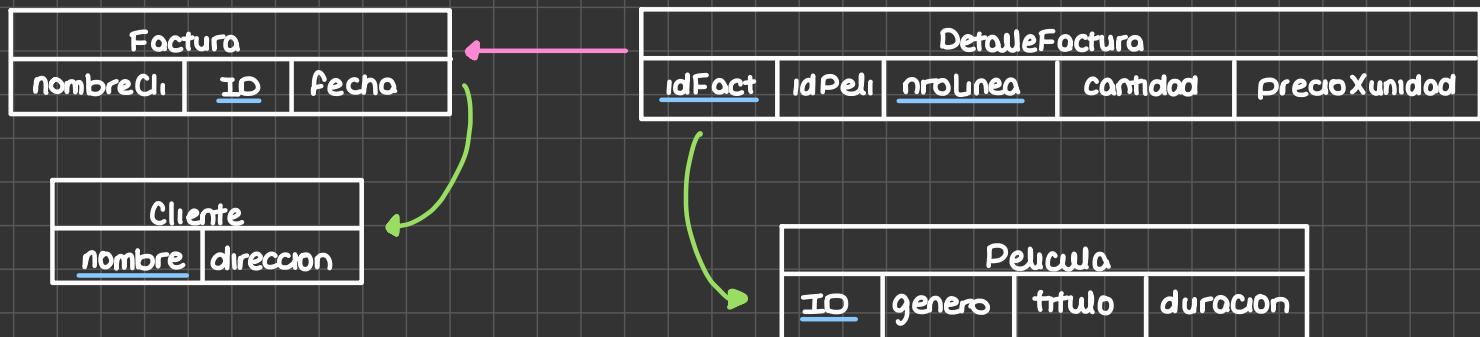
3.3 SELECT b_1, b_2 FROM B

WHERE (SELECT COUNT(*) FROM C WHERE $C.c_1 \leq 20$ AND

(SELECT COUNT(*) FROM R WHERE $R.b_1 = B.b_1$ AND $R.b_2 = B.b_2$ AND $R.c_1 = C.c_1$) = 0

Ejercicio 4

4.1



➡ ON DELETE SET NULL

➡ ON DELETE CASCADE

Cliente: nombre = PK

Factura: ID = PK y nombreCli = FK (puede ser NULL)

DetalleFactura: (idFact, nroLinea) = PK , idFact = FK (no puede ser NULL) y idPeli = FK (puede ser NULL)

Pelicula: ID = PK

4.2.1 VERDADERO

Una factura corresponde a un único cliente pues es una relación N:1. Sin embargo, un cliente puede tener múltiples facturas.

4.2.2 FALSO

Los números de líneas no pueden repetirse para una misma factura. Sin embargo, se pueden repetir para distintas facturas dado que la PK de DetalleFactura se compone del idFact y del nroLinea. Por lo tanto, se puede dar que dos facturas distintas (distintos IDs) repiten el mismo nroLinea.

Ejercicio 5

5.1 El PreparedStatement es preferible porque permite parametrizar una sentencia mediante wildcards, evitando tener que reescribir una misma sentencia muchas veces. Esto a su vez nos permite reusar la sentencia cambiando los datos. Por otro lado, usar PreparedStatement permite evitar el tráfico de datos pues la sentencia se compila una única vez y luego solo viajan los parámetros. Sin embargo, con el CreateStatement, el ODBC compila infinito veces una misma sentencia aunque solo difieran en los parámetros resultando en tráfico en la red innecesario.

5.2a Es deseable usar un atributo auto-numérico cuando el Pk de una tabla ocupa mucho espacio y hay otros tablas que referencien a dicha tabla (estanamos ocupando $(n+1) \times \text{tamaño(Pk)}$ si n tablas referencia a la tabla primaria).

5.2b La ventaja de usar un atributo auto-numérico es que las tablas que referencian a la original terminan ocupando menos espacio porque solo deben almacenar un número entero.

5.2c En PostgreSQL, la sentencia CREATE SEQUENCE permite crear un objeto con valores enteros únicos para ser utilizado como Pk. Luego, mediante un TRIGGER se puede incrementar el valor del objeto antes de cada inserción. Si bien este trigger puede fallar, no hay problema pues la secuencia puede tener huecos.

Obs: Existe una opción que permite omitir el trigger. Al momento de crear la tabla, se puede indicar la secuencia que se desea utilizar.

Otra alternativa en PostgreSQL:

Existen los datos de tipo SERIAL que, internamente, genera una secuencia i.e hace un CREATE SEQUENCE internamente.

5.3 Las validaciones a nivel atributo solo permiten hacer una verificación de dicho atributo. Sin embargo, las validaciones a nivel tupla/tabla permiten expresar condiciones más avanzadas, no modifican a un atributo en particular (Ej. comparar atributos de una misma tabla o de tablas distintas).

Obs: Las validaciones de tipo Check ya sea a nivel atributo o a nivel tupla solo se evalúan en cada inserción o modificación. Si una vez insertada la tupla, lo referenciado por ella desaparece o cambia, no se considera que viola la condición.

5.4 Para cargar un driver de manera dinámica se puede hacer uso de Class.forName en vez de import. De esta forma no hay invocación del constructor para registrar al driver con el DriverManager y no es necesario conocer en tiempo de compilación los drivers que se desean conectar.