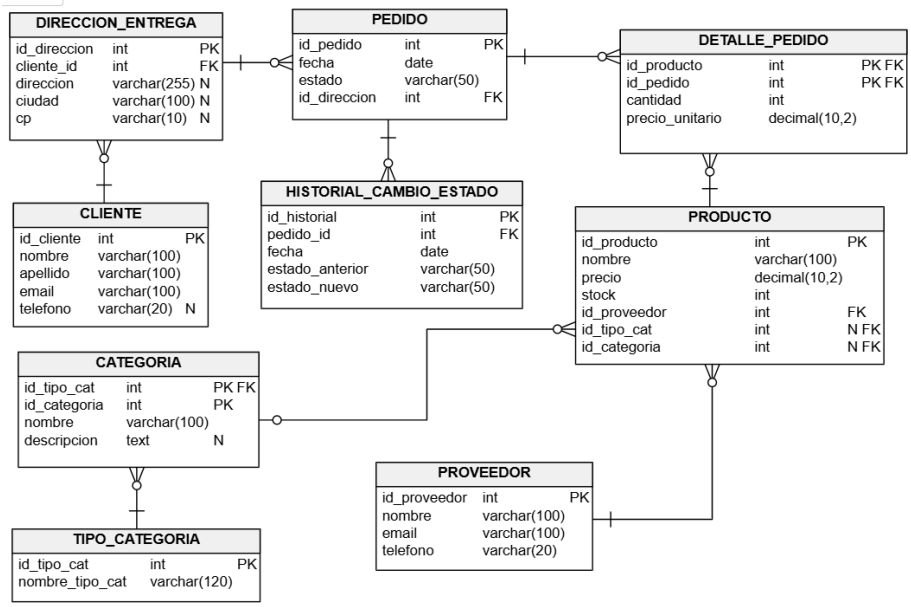
Comenzado el	jueves, 14 de agosto de 2025, 09:40
Estado	Finalizado
Finalizado en	jueves, 14 de agosto de 2025, 12:36
Tiempo empleado	2 horas 56 minutos

Información

Dado el siguiente esquema de bases de datos cuyo script de creación se encuentra en el siguiente link resuelva los ejercicios planteados a continuación:



Pregunta 1

Finalizado

Se puntúa como 0 sobre 1,00

1.a) En el esquema dado se requiere incorporar la siguiente restricción según **SQL estándar** utilizando el **recurso declarativo más restrictivo posible** (a nivel de atributo, de tupla, de tabla o general) y utilizando sólo las tablas/atributos necesarios.

- Verificar que cada pedido sólo incluya un producto sin categoría.

Seleccione la opción que considera correcta, de acuerdo a lo solicitado y justifique claramente porqué la considera correcta (debajo de la pregunta 1.b):

a.
```
CREATE ASSERTION check_control_pedido
CHECK ( NOT EXISTS
        ( SELECT 1
          FROM pedido pe JOIN detalle_pedido d using (id_pedido)
           JOIN producto p  USING (id_producto)
          WHERE id_tipo_cat IS NOT NULL
              AND id_categoria IS NOT NULL
          GROUP BY d.id_pedido
           HAVING COUNT(*) > 1) );
```

b. Ninguna de las opciones es la correcta

c.
```
CREATE ASSERTION check_control_pedido
CHECK ( EXISTS
        ( SELECT 1
          FROM pedido pe JOIN detalle_pedido d using (id_pedido)
           JOIN producto p  USING (id_producto)
          WHERE id_tipo_cat IS NOT NULL
              AND id_categoria IS NOT NULL
          GROUP BY d.id_pedido
           HAVING COUNT(*) > 1) );
```

d.
```
ALTER TABLE detalle_pedido ADD CONSTRAINT  check_control_pedido
CHECK (NOT EXISTS
        ( SELECT 1
          FROM detalle_pedido d JOIN producto p
          USING (id_producto)
           WHERE id_tipo_cat IS NULL
              AND id_categoria IS NULL
          GROUP BY d.id_pedido
           HAVING COUNT(*) > 1));
```

e.
```
CREATE ASSERTION check_control_pedido
CHECK ( NOT EXISTS
        ( SELECT 1
```

```
                    FROM pedido pe JOIN detalle_pedido d using (id_pedido)
                     JOIN producto p  USING (id_producto)
                    WHERE id_tipo_cat IS NULL
                        AND id_categoria IS NULL
                    GROUP BY d.id_pedido
                     HAVING COUNT(*) > 1) );
```

f.

```
CREATE ASSERTION check_control_pedido
CHECK ( NOT EXISTS
                ( SELECT 1
                  FROM detalle_pedido d JOIN producto p
                  USING (id_producto)
                   WHERE id_tipo_cat IS NULL
                       AND id_categoria IS NULL
                   GROUP BY d.id_pedido
                    HAVING COUNT(*) > 1) );
```

Respuesta correcta

La respuesta correcta es:

```
CREATE ASSERTION check_control_pedido
CHECK ( NOT EXISTS
            ( SELECT 1
              FROM detalle_pedido d JOIN producto p
              USING (id_producto)
               WHERE id_tipo_cat IS NULL
                   AND id_categoria IS NULL
               GROUP BY d.id_pedido
                HAVING COUNT(*) > 1) );
```

Pregunta 2

Finalizado

Se puntúa como 0 sobre 1,00

1.b) En el esquema dado se requiere incorporar la siguiente restricción según SQL estándar utilizando el recurso declarativo más restrictivo posible (a nivel de atributo, de tupla, de tabla o general) y utilizando sólo las tablas/atributos necesarios.

- *La fecha en el historial de cambios debe ser posterior o igual a la del pedido*

Resuelva según lo solicitado y justifique el tipo de chequeo utilizado.

Nota: No se olvide de justificar la pregunta 1.a)

```
1. La opción correcta es la "f" ya que este utiliza un ASSERTION
   que controla que no exista mas de un registro asociado a un pedido
   con mas de un producto con id_tipo_cat NULL y id_categoria NULL.
   Hace justamente lo que pide el enunciado.


1b. "La fecha en el historial de cambios
    debe ser posterior o igual a la del pedido"

La restriccion es un ASSERTION ya que la consulta involucra datos de mas
de una tabla.

CREATE ASSERTION ej1b CHECK (
       NOT EXISTS ( SELECT 1
                    FROM historial_cambio_estado h
                    JOIN pedido p ON p.id_pedido = h.pedido_id
                    WHERE h.fecha < p.fecha
                   )
);
```

Comentario:

1.a) Bien

1.b) Bien

Pregunta 3

Finalizado

Se puntúa como 0 sobre 1,00

2.a) Sobre el esquema dado se requiere definir la siguiente vista, de manera que resulte automáticamente actualizable en PostgreSQL, siempre que sea posible:

- V1: *que contenga todos los datos de los pedidos cuyo monto total sea mayor a $100.000*

Considerando la siguiente definición para V1, seleccione la/s afirmación/es que considere correcta/s respecto de esta vista (Nota: tenga en cuenta que las opciones incorrectamente seleccionadas pueden restar puntaje) y justifíquela/s claramente (debajo de la pregunta 2.b).

```
CREATE VIEW V1 AS
SELECT *
FROM pedido p
WHERE EXISTS (SELECT 1
                FROM detalle_pedido d
                WHERE p.id_pedido = d.id_pedido
                 GROUP BY id_pedido
                 HAVING SUM (precio_unitario * cantidad) > 100000);
```

- [ ] a. No está correctamente correlacionada la consulta con la subconsulta
- [ ] b. No es posible reformular la consulta para que cumpla con lo requerido (y sea automáticamente actualizable)
- [x] c. Contiene todos los datos de los pedidos cuyo monto total sea mayor a $100.000
- [ ] d. Para cumplir lo requerido hay que reformular la consulta, sólo cambiando > por <=
- [ ] e. No resulta automáticamente actualizable en PostgreSQL
- [x] f. Filtra correctamente los pedidos cuyo monto total es mayor a $100.000
- [x] g. Es automáticamente actualizable en PostgreSQL
- [ ] h. Para cumplir lo requerido hay que reformular la consulta, el EXISTS por NOT EXISTS y el > (mayor) por <= (menor igual)
- [ ] i. Ninguna de las opciones
- [x] j. Está correctamente correlacionada la consulta con la subconsulta

Respuesta correcta

Las respuestas correctas son:

Contiene todos los datos de los pedidos cuyo monto total sea mayor a $100.000,

Está correctamente correlacionada la consulta con la subconsulta,

Filtra correctamente los pedidos cuyo monto total es mayor a $100.000,

Es automáticamente actualizable en PostgreSQL

Pregunta 4

Finalizado

Se puntúa como 0 sobre 1,00

2.b) Sobre el esquema dado se requiere definir la siguiente vista, de manera que resulte automáticamente actualizable en PostgreSQL, siempre que sea posible, y que se verifique que no haya migración de tuplas de la vista. Resuelva según lo solicitado y justifique su solución.

- V2: Que liste todos los productos indicando el nombre, precio, stock, el nombre de su categoría y el nombre de su tipo categoría. Debe incluir los productos sin categoría.

Nota: Recuerde justificar la pregunta 2.a)

```
2b. CREATE VIEW V2 AS (
    SELECT p.nombre, p.precio, p.stock, c.nombre, t.nombre_tipo_cat
    FROM producto p
    LEFT JOIN categoria c ON p.id_categoria = c.id_categoria AND p.id_tipo_cat = c.id_tipo_ca
    JOIN tipo_categoria t ON c.id_tipo_cat = t.id_tipo_cat
    );
```

Comentario:
Debe usar LEFT JOIN tipo_categoria para incluir todos los productos no solo los que tienen categoría

Pregunta 5

Finalizado

Se puntúa como 0 sobre 1,00

3) Para el esquema dado, se ha creado la tabla productos_cliente donde se requiere registrar la siguiente información para todos los clientes que están registrados en la base:

id_cliente, nombre, apellido, email, cantidad_productos, fecha_ultimo_pedido

donde, para cada cliente:

- cantidad_productos corresponde a la cantidad de productos pedidos.

- fecha_ultimo_pedido es la fecha correspondiente al último pedido.

Nota: en caso que un cliente no registre pedidos, se deberá indicar apropiadamente.

a) Implemente el método más adecuado en PostgreSQL que permita completar dicha tabla con la información de todos los clientes a partir de los datos existentes en la base. Explique su solución e incluya la sentencia que debería utilizar un usuario para la ejecución del mismo.

Nota: no puede utilizar sentencias de bucle (for, loop, etc.) para resolverlo.

```
Entiendo que tengo que implementar un UPDATE sobre la nueva
tabla para poder cargar todos los datos correspondientes
en la misma.


CREATE VIEW v_datos AS (
SELECT c.*, SUM(dp.cantidad)cantidad_productos,(SELECT MAX(p.fecha) FROM pedido p2 JOIN dir
FROM cliente c
JOIN direccion_entrega d ON ...
JOIN pedido p ON ...
JOIN detalle_pedido dp ON ...
GROUP BY c.id_cliente
)
```

Comentario:
Mal

No agrupa bien

Pregunta 6

Finalizado

Se puntúa como 0 sobre 1,00

3.b) Indique y justifique todos los eventos críticos necesarios para mantener los datos actualizados en la tabla productos_cliente cuando se produzcan actualizaciones en la base. Incluya la declaración de los triggers correspondientes en PostgreSQL y escriba la implementación de la/s función/es requerida/s para operaciones de insert.

```
No llegue con el tiempo, pero entiendo el concepto que
hay que implementar los triggers con sus funciones
para mantener la nueva tabla actualizada.
```

Comentario:
No resuelto

Pregunta 7

Finalizado

Se puntúa como 0 sobre 1,00

4) El dueño del esquema dado ha ejecutado las siguientes sentencias

CREATE USER user_admin IDENTIFIED BY 'admin123';
CREATE USER user_vendedor IDENTIFIED BY 'vend123';
CREATE USER user_cliente IDENTIFIED BY 'cli123';
CREATE USER user_auditor IDENTIFIED BY 'audit123';

GRANT ALL PRIVILEGES ON PRODUCTO TO user_admin WITH GRANT OPTION;
GRANT INSERT ON PEDIDO TO user_vendedor;
GRANT INSERT ON DETALLE_PEDIDO TO user_vendedor;
GRANT SELECT ON CLIENTE TO user_cliente;
GRANT SELECT ON DIRECCION_ENTREGA TO user_cliente;
GRANT SELECT ON HISTORIAL_CAMBIO_ESTADO TO user_auditor WITH GRANT OPTION;
GRANT UPDATE (stock) ON PRODUCTO TO user_vendedor;

Luego de algunos meses el dueño del esquema ejecuta

REVOKE INSERT ON PEDIDO FROM user_vendedor;
REVOKE SELECT ON DIRECCION_ENTREGA FROM user_cliente;
REVOKE GRANT OPTION FOR ALL PRIVILEGES ON PRODUCTO FROM user_admin;

...y user_admin ejecuta

REVOKE GRANT OPTION FOR SELECT ON HISTORIAL_CAMBIO_ESTADO FROM user_auditor;
REVOKE ALL PRIVILEGES ON DETALLE_PEDIDO FROM user_vendedor;

¿Qué permisos tendrá cada usuario después de estas modificaciones?

```
ADMIN: SELECT, INSERT, UPDATE, DELETE, REFERECES (TODOS SIN GRANT OPTION)

VENDEDOR: INSERT (SOBRE DETALLE_PEDIDO), UPDATE (SOBRE STOCK DE PRODUCTO)

CLIENTE: INSERT (SOBRE CLIENTE)

AUDITOR : SELECT (SOBRE HISTORIAL_CAMBIO ESTADO) CON GRANT OPTION
```

user_admin conserva ALL PRIVILEGES ON PRODUCTO sin el WGO
user_vendedor INSERT ON DETALLE_PEDIDO y UPDATE (stock) ON PRODUCTO
user_cliente SELECT ON CLIENTE
user_auditor SELECT ON HISTORIAL_CAMBIO_ESTADO con WGO

Comentario:

Bien

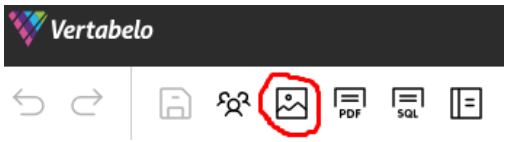
Pregunta 8

Finalizado

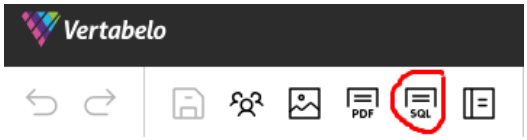
Se puntúa como 0 sobre 1,00

5) Dado el script de tablas (link) complete y/o corrija (en caso de ser necesario) en Vertabelo el esquema de tablas y restricciones de integridad (de clave, referencial y de nulidad) para el siguiente diagrama de entidades y relaciones. Incluya los siguientes archivos:

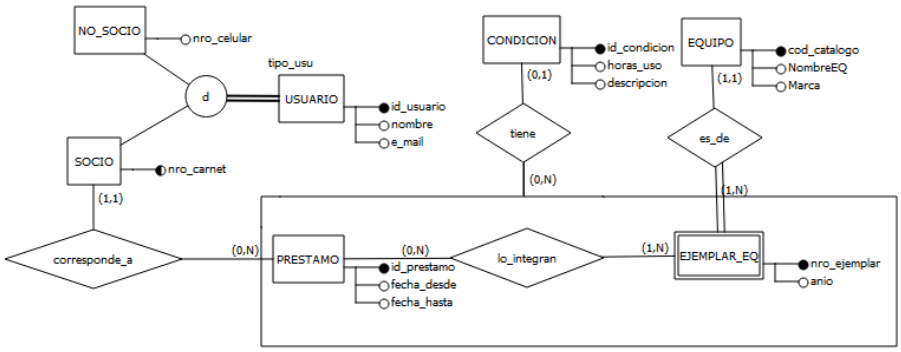
1. **Imagen del esquema en PNG: nombrado como DERE_IMG_XXX.png** (donde XXX son los ultimos 3 nros de su documento): El tipo de imagen seleccionado debe ser



2. **Script SQL generado por Vertabelo nombrado como DERE_ESQ_XXX.sql** (donde XXX son los ultimos 3 nros de su documento)



Nota: nombre adecuadamente los constraint de PK, FK. AK, como prefijos



completa

DERE_ESQ_729.sql

DERE_IMG_729.png

Comentario:
Bin, la cardinalidad entre lo_integra y ejemplar_eq es (1,N)

Pregunta 9

Finalizado

Se puntúa como 0 sobre 1,00

6) En el esquema dado se detectó un problema:
Algunos clientes logran registrar múltiples direcciones residenciales idénticas, generando redundancia innecesaria en la tabla DIRECCION_ENTREGA.

Por otro lado, se quiere evitar que un cliente tenga más de una dirección registrada por ciudad, ya que esto genera conflictos logísticos en los envíos.

La política que se desea imponer es:

- Un cliente no puede registrar más de una dirección en la misma ciudad, y no puede registrar direcciones idénticas (mismo texto en dirección) más de una vez.

a) Agregar una restricción UNIQUE(cliente_id, direccion) en la tabla DIRECCION_ENTREGA.

b) Agregar una restricción UNIQUE(cliente_id, ciudad) en la tabla DIRECCION_ENTREGA.

c) Combinar las restricciones UNIQUE(usuario_id, direccion) y UNIQUE(usuario_id, ciudad) en la tabla, aunque esto puede producir errores si se insertan direcciones distintas dentro de la misma ciudad.

d) Crear un trigger BEFORE INSERT OR UPDATE que consulte si ya existe una fila para el mismo cliente_id con la misma ciudad o la misma dirección exacta, y si es así, rechace la operación.

e) Rediseñar la tabla DIRECCION_ENTREGA creando una entidad separada para CIUDAD, asociándola por clave foránea y asegurando unicidad en una tabla intermedia.

f) Sólo se puede resolver a nivel de aplicación, ya que las bases de datos relacionales no permiten imponer múltiples restricciones condicionales sobre columnas diferentes.

g) Ninguna opción es correcta

```
Estoy entre la solucion D y la solucion E. Pero me quedo con la D ya que esta contempla
la unicidad de direcciones por usuario y por cuidad, al utilizar trigger
before insert or update checkea todo el tiempo que la regla no se rompa.
En cambio la solucion E solo resuelve la unicidad por cuidad.


"d) Crear un trigger BEFORE INSERT OR UPDATE que consulte si ya existe una
    fila para el mismo cliente_id con la misma ciudad o la misma dirección
    exacta, y si es así, rechace la operación."
```

Comentario:
No son correctas esas opiciones

Actividad previa

◄ Avisos

Ir a...