IIC2413

AYUDANTÍA 11

REPASO 12

Vistas

- Una consulta almacenada que se comporta como tabla virtual.
- No almacena datos físicamente.
- Se reescribe y ejecuta sobre las tablas base.
- Uso: reducir complejidad y evitar repetición de consultas.

```
CREATE VIEW <nombre_view> AS (
    <sentencias_SQL>
);
```

Triggers

- Procedimientos que se ejecutan automáticamente cada vez que ocurre un evento específico en la base de datos (Ej: BEFORE/AFTER INSERT, UPDATE, DELETE).
- Objetivo: Forzar restricciones complejas y mantener la consistencia de la BDD de forma automática.

```
CREATE TRIGGER <nombre_trigger>
[BEFORE | AFTER | INSTEAD OF] [ DELETE | UPDATE | INSERT] ON <tabla>
[FOR EACH ROW | FOR EACH STATEMENT]
BEGIN
    UPDATE <tabla_a_actualizar>
    SET <atributo> = <valor>
    WHERE <condición>
END;
```

Stored Procedures

- Funciones definidas en SQL guardadas en el DBMS.
- Permiten ejecutar lógica compleja y repetitiva directamente en el DBMS.
- Ventajas: Velocidad, seguridad, rendimiento, reutilizabilidad.
- Desventaja principal: Dificultad o imposibilidad de migrar a otra base de datos.

```
CREATE OR REPLACE FUNCTION <nombre_función> (<argumentos>)
RETURNS [VOID | TABLE(...) ] AS $$
DECLARE
    <declaración_variables>
BEGIN
    <sentencias_SQL>
END; $$ language plpgsql
```

Ejercicio: I2 2024-2 Pregunta 1

- Producto (id prod SERIAL PRIMARY KEY, sku varchar(50), nombre prod VARCHAR(100), descripcion prod VARCHAR(500), id cate INT, activo BOOLEAN, FOREIGN KEY (id cate) REFERENCES categoria(id cate))
- Categoria (id cate SERIAL PRIMARY KEY, nombre cate VARCHAR(100), descripcion cate VARCHAR(100), activo BOOLEAN)
- Lista Precios (id lista SERIAL PRIMARY KEY, inicio vigencia DATE, fin vigencia DATE, tipo INT)
- Producto Lista (id prod INT, id lista INT, precio INT, PRIMARY KEY (id prod, id lista))

Con respecto al esquema anterior, es necesario tener en cuenta las siguientes consideraciones:

- La tabla de Producto contiene la información de todos los productos que han existido en la empresa DCCEcommerce, donde los productos inactivos tienen el campo activo en FALSE.
- En la tabla Categoría se almacenan todas las categorías a las cuales se puede asociar un producto (1 producto solo puede pertenecer a 1 categoría), teniendo el campo activo en F ALSE para las categorías que ya no son válidas.
- En la tabla Lista P recios están todas las listas de precios que existen en DCCEcommerce y se usan los campos de fechas para determinar si una lista de precios se encuentra vigente o no.
- Finalmente, la tabla Producto Lista permite realizar la asociación entre Productos y Lista Precios. Un producto puede pertenecer a 2 listas de precios. Una lista de precios base a la cual pertenecen todos los productos y una lista con la campaña actual. El tipo con valor 0 es la lista base y valor 1 es la oferta actual.

Ejercicio: I2 2024-2 Pregunta 1

a) Cree un Procedimiento Almacenado STORED PROCEDURE que reciba como parámetros una fecha y categoría; y retorne el precio vigente de todos los productos en la fecha entregada como parámetro, para la categoría consultada. En caso de que el producto pertenezca a una lista de oferta, debe seleccionar ese precio y en caso contrario, el precio base.

```
CREATE or REPLACE Function precio_vigente(fecha date, id_cate INT)
RETURNS TABLE (id_prod INT, nombre_prod VARCHAR(100), precio INT) AS $$
BEGIN
RETURN QUERY EXECUTE '
 SELECT prod.id_prod, prod.nombre_prod, prod_list.precio
 FROM Producto as prod NNER JOIN
 Categoria as categ ON prod.id_cate = categ.id_cate
 INNER JOIN Producto_Lista as prod_list ON prod.id_prod = prod_list.id_prod
 INNER JOIN Lista_Precios as list_prec ON prod_list.id_lista = lis_prec.id_lista
 WHERE categ.id_cate = $1
 AND list_prec.inicio_vigencia >= $2
 AND list_prec.fin_vigencia <= $2
 AND list_prec.tipo = (
 SELECT max(tipo)
 FROM Lista_Precios as lp INNER JOIN Producto_Lista as pl ON lp.id_lista = pl.id_lista
 WHERE lp.id_lista = list_prec.id_lista
 AND pl.id_prod= prod_list.id_prod)
 )'
 USING id_cate, fecha;
 RETURN;
END
$$ language plpgsql
```

b) Cree un TRIGGER, que al momento de insertar un registro en la tabla Lista Precios, cree una tabla con el nombre Productos Sin Precio Y Y Y Y MMDD e inserte todos los productos activos, que no pertenezcan a una lista de ofertas (tipo = 1). Ud decida que campos debe tener la tabla.

```
CREATE TRIGGER prod_sin_precios
AFTER CREATE ON Lista_Precios
FOR EACH ROW
EXECUTE FUNCTION crear_tabla_productos_sin_precio();
CREATE OR REPLACE FUNCTION crear_tabla_productos_sin_precio() RETURNS TRIGG
DECLARE
 fecha_actual TEXT;
 nombre_tabla TEXT;
BEGIN
 fecha_actual := TO_CHAR(NOW(), 'YYYYMMDD');
 nombre_tabla := CONCAT('Producto_Sin_Precio_',fecha_actual);
 EXECUTE 'CREATE TABLE IF NOT EXISTS ' || nombre_tabla
 || ' AS
 SELECT p.id_prod, p.nombre_prod
 FROM Producto p
 WHERE NOT EXISTS (
 SELECT 1
 FROM Lista_precios lp INNER JOIN Producto_Lista pl ON lp.id_lista = pl.id_lista
 WHERE lp.inicio_vigencia <= NOW()
 AND lp.fin_vigencia >= NOW()
 AND pl.id_prod = p.id_prod
 )';
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

c) Cree una vista VIEW, que permita calcular el precio promedio de los productos por categoría al día de hoy. Debe usar el precio de la lista de ofertas y en caso de que el producto no tenga, usar la lista base.

```
CREATE VIEW Precio_Promedio AS
SELECT categ.id_cate, cate.nombre_cate, AVG(prod_list.precio)
 FROM Producto as prod NNER JOIN
 Categoria as categ ON prod.id_cate = categ.id_cate
 INNER JOIN Producto_Lista as prod_list ON prod.id_prod =
prod_list.id_prod
 INNER JOIN Lista_Precios as list_prec ON prod_list.id_lista =
lis_prec.id_lista
 WHERE list_prec.inicio_vigencia >= NOW()
 AND list_prec.fin_vigencia <= NOW()
 AND list_prec.tipo = (
 SELECT max(tipo)
 FROM Lista_Precios as lp INNER JOIN Producto_Lista as pl ON
lp.id_lista = pl.id_lista
 WHERE lp.id_lista = list_prec.id_lista
 AND pl.id_prod= prod_list.id_prod)
 )
 GROUP BY categ.id_cate, categ.nombre_cate
```

Transacciones

- Secuencia de una o más operaciones que modifican o consultan la BD.
- Ejemplo: transferencia de dinero, inscripción de curso, compra online.
- Objetivo: garantizar integridad y consistencia, incluso ante fallas.

Propiedades ACID

- Atomicity: se ejecuta completa o no.
- Consistency: preserva la consistencia de la BD.
- Isolation: debe comportarse como si fuera la única ejecutándose.
- Durability: los cambios persisten tras fallas.

Conflictos con transacciones

- Lecturas sucias (Write - Read): se lee un dato que otra transacción cambió pero aún no confirmó.
- Lecturas no repetibles (Read - Write): se lee dos veces el mismo dato y cambia entre lecturas porque otra transacción lo modificó.
- Actualización perdida o reescritura de datos temporales (Write - Write): dos transacciones escriben el mismo dato y una sobrescribe el cambio de la otra.

Schedules

Secuencia de operaciones realizadas por múltiples transacciones de forma concurrente.

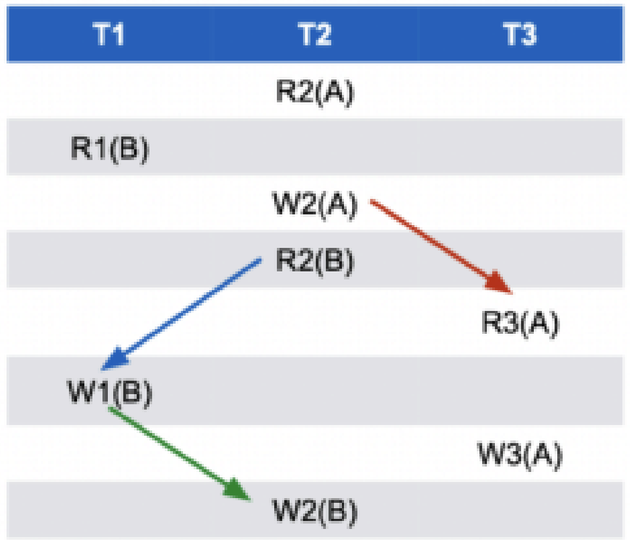
- Está compuesto por operaciones de múltiples transacciones que pueden ejecutarse de manera concurrente.
- Define el orden en el cual las operaciones de las transacciones son ejecutadas.
- Puede ser Serializable o No Serializable: depende de si la ejecución produce el mismo resultado que alguna secuencia serial (no concurrente) equivalente de las mismas transacciones.

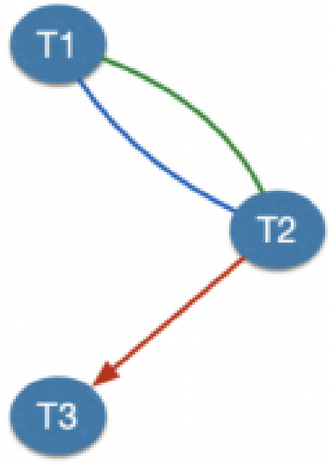
Schedules

- Schedule Serial: no hay intercalación entre las acciones.
- Schedule Serializable: si existe algún schedule S' serial con las mismas transacciones, tal que el resultado de S y S' es el mismo.
- Schedule No Serializable: produce un resultado diferente que cualquier secuencia serial equivalente de las mismas transacciones.
- Schedule Conflict Serializable: un schedule es conflict serializable si puedo transformarlo a uno serial usando permutaciones.

Grafos de procedencia

Representación gráfica que muestra las dependencias de orden entre las operaciones de diferentes transacciones.





*** Un schedule es conflict serializable si solo si el grafo es acíclico (en este caso NO lo es)

Ejercicio: I2 2025-1 Pregunta 2

a) Defina los siguientes conceptos:

1. Schedule Serializable.
2. Schedule Serial.
3. Schedule Conflict Serializable.

a) Defina los siguientes conceptos:

i. Schedule Serializable. (2 pts)

Solución: Es un orden de ejecución de transacciones concurrentes que produce el mismo resultado que la ejecución de un schedule serial.

ii. Schedule Serial. (2 pts)

Solución: Es aquel en que las transacciones se ejecutan una después de otra, sin intercalación entre ellas.

iii. Schedule Conflict Serializable. (2 pts)

Solución: Un schedule conflict serializable es un tipo de schedule serializable donde es posible reordenar las operaciones (sin cambiar el orden relativo de operaciones conflictivas) para obtener un schedule serial equivalente.

b) Para cada una de las siguientes ejecuciones, especifique si corresponde a Lecturas Sucias, Lecturas No repetibles o Actualización Perdida. Justifique su respuesta.

T1	T2	T3
R(a)		
	W(a)	
R(b)		
	W(c)	
W(b)		
		R(a)
R(a)		
		R(c)

Schedule 1

T1	T2	T3
R(a)		
	R(a)	
R(b)		
		W(c)
W(b)		
	W(a)	
R(c)		
		Abort

Schedule 2

T1	T2	T3
R(a)		
	R(a)	
W(b)		
		W(c)
W(a)		
	W(a)	
R(c)		
	W(c)	

Schedule 3

T1	T2	T3
R(a)		
	W(a)	
R(b)		
	W(c)	
W(b)		
		R(a)
R(a)		
		R(c)

Schedule 1

T1	T2	T3
R(a)		
	R(a)	
R(b)		
		W(c)
W(b)		
	W(a)	
R(c)		
		Abort

Schedule 2

T1	T2	T3
R(a)		
	R(a)	
W(b)		
		W(c)
W(a)		
	W(a)	
R(c)		
	W(c)	

Schedule 3

Solución: El Schedule 1, es Lectura no repetible. T1 lee a, después T2 escribe en a y cuando T1 nuevamente lee a, el valor fue actualizado por T2.

En Schedule 2, es Lectura Sucia. T3 escribe en c, T1 lee c y después T3 aborta. T1 leyó un valor de c que no se confirmó.

En Schedule 3, es Actualización Perdida. T1 lee a, después lee T2, posteriormente T1 escribe en a y finalmente T2 lo sobreescribe, Por tanto, se pierde la escritura de T1.

c) Aplicando la técnica de permutación, demuestre si la siguiente ejecución es conflict serializable.

T1	T2	T3
R(b)		
	R(a)	
W(b)		
		W(c)
W(b)		
		W(a)
R(c)		
	W(c)	

Cuadro 1: Conflict Serializable

T1	T2	T3
R(b)		
	R(a)	
W(b)		
		W(c)
W(b)		
		W(a)
R(c)		
	W(c)	

Cuadro 1: Conflict Serializable

Solución: Hay dependencia entre la T2 y la T3 por las operaciones sobre a, por lo tanto no es posible permutar, También hay conflicto entre T3 y T2 por las operaciones sobre c, por lo tanto tampoco se pueden permutar. Esto produce que no es posible dejar T2 antes que T3 o T3 antes que T2.

Undo Loggin

- Regla 1: Si T modifica X, todos log <T, X, t> deben ser escritos antes que el valor X sea escrito en disco.
- Regla 2: Si T hace COMMIT, el log <COMMIT T> debe ser escrito justo después de que todos los datos modificados por T estén almacenado en disco.

1. T cambia el valor del X (t es el valor antiguo)
2. Generar el log <T, X, t>
3. Escribir los logs <T, X, t> al disco
4. Escribir valor nuevo de X a disco.
5. Escribir <COMMIT T>

Undo Loggin: Recovery

¿Qué hacer cuando hay falla?

- Si leo <COMMIT T> , marco T como realizada
- Si leo <ABORT T>, marco T como realizada
- Si leo <T, X, t>, debo restituir X := t en disco, si no fue realizada.
- Si leo <START T>, lo ignoro

- Esto solo se realiza dentro de la sección donde hay falla, no en toda la ejecución!

Recovery con checkpoints

- Si encontramos un <END CKPT> , hacemos undo de todo lo que haya empezadodespués del inicio del checkpoint <START CKPT>.
- Si encontramos un <START CKPT (T1...TN)> sin su <END CKPT>, debemos analizar el log desde el inicio de la transacción más antigua entre T1 ,...,TN.

Redo Loggin

Regla 1:
Antes de modificar cualquier elemento X en el disco, es necesario que todos los logs esten almacenados en disco:

1. Escribir el log <T, X, v>
2. Escribir <COMMIT T>
3. Escribir los datos en disco.
4. Escribir<T, End> en log.

Recovery con checkpoints

Procesar el log desde el final al inicio:

- Si hay un <END CKPT>, retroceder hasta el respectivo <START CKPT> y comenzar a hacer redo desde la transacción más antigua sin END.
- No hacer redo de las transacciones con COMMIT antes del <START CKPT>.

Ejercicio: I2 2024-2 Pregunta 2

a) Escriba el log que se genera usando UNDO LOGGING

b) Escriba el log que se genera usando REDO LOGGING

T1
T1 START
R(A)
A = A + 10
W(A)
W(X)
R(B)
B = B - 5
W(B)
T1 COMMIT

Cuadro 7: Ejecución Recuperación

T1
T1 START
R(A)
A = A + 10
W(A)
W(X)
R(B)
B = B - 5
W(B)
T1 COMMIT

Cuadro 7: Ejecución Recuperación

a)

Suponemos los siguientes valores iniciales A = 0, B = 5, X = 10

<START T1>

<T1, A, 0>

<T1, X, 10>

<T1, B, 5>

<COMMIT T1>

b)

Suponemos los siguientes valores iniciales A = 0, B = 5, X = 10

<START T1>

<T1, A, 10>

<T1, X, 10>

<T1, B, 0>

<COMMIT T1>

<T1 END>

c) Describa las acciones que se realizarían usando UNDO LOGGING si ocurre una falla antes de ejecutar W(B)

d) Describa las acciones que se realizarían usando REDO LOGGING si ocurre una falla antes de ejecutar W(B)

T1
T1 START
R(A)
A = A + 10
W(A)
W(X)
R(B)
B = B - 5
W(B)
T1 COMMIT

Cuadro 7: Ejecución Recuperación

c) Describa las acciones que se realizarían usando UNDO LOGGING si ocurre una falla antes de ejecutar W(B)

El log se lee desde abajo hacia arriba. Cómo el error es antes de W(B) y no se ha ejecutado COMMIT, se realizan las siguientes acciones:

- Escribir 10 en X
- Escribir 0 en A

d) Describa las acciones que se realizarían usando REDO LOGGING si ocurre una falla antes de ejecutar W(B)

El log se lee desde el principio hasta el final. Cómo el error se produce antes de hacer COMMIT, ninguna acción se realiza.

T1
T1 START
R(A)
A = A + 10
W(A)
W(X)
R(B)
B = B - 5
W(B)
T1 COMMIT

Cuadro 7: Ejecución Recuperación

Modelo de costos

Se mide por el número de accesos (lectura/escritura) a la Memoria Secundaria.

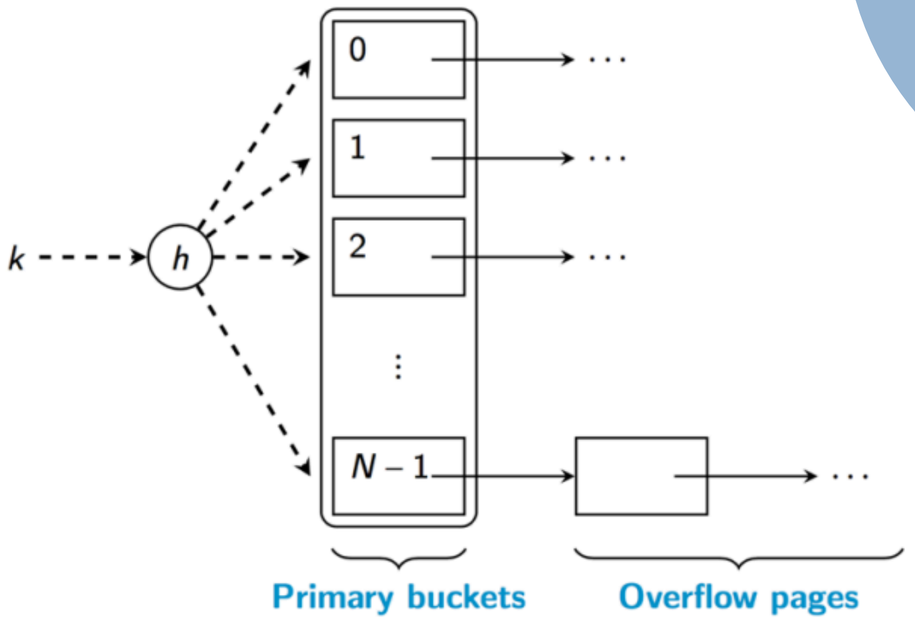
- B: Tamaño de una página en tuplas.
- |R|: Número total de tuplas en la relación.
- I/O: La unidad de costo (lectura de una página).

SELECT * FROM T WHERE T.a = 4
SELECT * FROM T WHERE T.a >= 4

- Costo sin índice: O(⌈ |R| / B ⌉

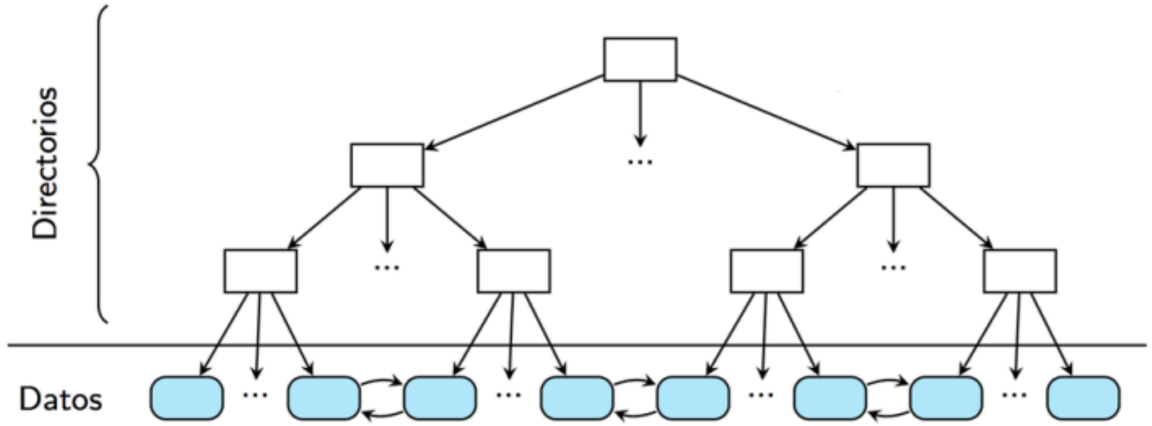
Método Hash Index

- Usa una función de hash que mapea valores a buckets (páginas).
- Ideal para consultas de igualdad, ineficiente para rangos.
- Costo O(1).
- En el peor caso el costo es O(⌈ |R| / B | ⌉ .



Método B+ Tree Index

- Árbol balanceado de altura h
- Soporta igualdad y rango eficientemente.
- Cada nodo y hoja ocupa una página.
- El costo de búsqueda es logarítmico log(|R|)
- Si B es el tamaño de una página en tuplas, el costo de búsqueda es: O(log⌈ |R| / B | ⌉) * log en base B .



Clustered v/s unclustered

Clustered

- Tupla completa en el bucket
- Acceso directo a la página de datos

Unclustered

- Puntero a la tupla
- Lee índice y luego página de datos

Resumen #I/O

Método	# I/O	Aclaración
Hash index clustered	1	El bucket es la página con mi tupla
Hash index unclustered	2	
B+ tree clustered	h + Bm	h: altura árbol Bm: # bloques con tuplas que calzan búsqueda
B+ tree unclustered	h + ⌈ n/P⌉ + n	n : número de tuplas que calzan con búsquedas P : Cantidad de punteros por bloque

Ejercicio: I2 2024-2 Pregunta 3

Sea la relación S(a, b, c, d, e) cuyo tamaño es de 10 millones de tuplas, en que cada página de disco contiene P tuplas.

Las tuplas de S están distribuidas de manera aleatoria.

El atributo c es además una llave candidata, cuyos valores únicos van del 0 al 9.999.999 (distribuidos uniformemente).

Para cada una de las consultas a continuación, indique el número de operaciones de I/O que se realizarán para cada una de las siguientes consultas:

a) Si no existe índice: Encontrar todas las tuplas de S tal que $50 < c < 150$.

a) Si no existe índice: Encontrar todas las tuplas de S tal que $50 < c < 150$.

Solución: como no existe índice, es necesario recorrer todas las páginas para encontrar los registros que nos interesan.

Dado que son 10^7 tuplas y en cada página hay P tuplas, el costo es: 10^7 / P

b) Si existe un índice clustered tipo B+ Tree sobre el atributo c, el árbol es de altura h y cada página de las hojas está ocupada al 60 %.
Encontrar todas las tuplas de S tal que $c < 50$.

b) Si existe un índice clustered tipo B+ Tree sobre el atributo c, el árbol es de altura h y cada página de las hojas está ocupada al 60 %.
Encontrar todas las tuplas de S tal que $c < 50$.

Solución: como se necesitan las tuplas donde $c < 50$, las tuplas necesarias son 50 (el índice comienza en 0).
Como las hojas de las páginas están al 60 %, para guardar todas las tuplas se necesitan: $10^7 / (0{,}6 \times P)$
Como en este caso son 50 registros, para encontrar todas las tuplas se necesitan recorrer: $50 / (0{,}6 \times P)$, pero es necesario restar 1 porque es clustered.
Para llegar a las hojas, el costo es h, por lo tanto el valor es:
$h - 1 + 50 / (0{,}6 \times P)$

c) Si existe un índice unclustered de tipo Hash con 10 millones de buckets.
Cada página del índice contiene M punteros (M > P).
Encontrar todas las tuplas de S tal que c = 125.

c) Si existe un índice unclustered de tipo Hash con 10 millones de buckets.
Cada página del índice contiene M punteros (M > P).
Encontrar todas las tuplas de S tal que c = 125.

Solución: en este caso, como necesitamos un solo registro, se necesitan 2 operaciones: una operación para llegar al bucket del índice, donde se almacenan los índices por ser unclustered, y la otra para acceder a la página donde está la tupla.

d) Si existe un índice B+ Tree unclustered sobre el atributo c, el árbol es de altura h y cada página contiene M punteros (M > P).

Encontrar todas las tuplas de S tal que c < 100.

d) Si existe un índice B+ Tree unclustered sobre el atributo c, el árbol es de altura h y cada página contiene M punteros (M > P).

Encontrar todas las tuplas de S tal que c < 100.

Solución: como se necesitan las tuplas donde c < 100, las tuplas necesarias son 100 (el índice comienza en 0).

Como en cada página hoja se almacenan M punteros, se necesitan 10^7 / M páginas para todos los punteros.

Como en este caso son 100 registros, para todos los punteros se necesitan recorrer 100 / M páginas.

Para llegar a las hojas, el costo es h, pero como al recorrer se llega a la primera página con índices, es necesario restar 1.

Adicionalmente, hay que ir desde cada puntero a la página por cada tupla, por lo tanto se deben sumar 100.

El costo total es: h − 1 + (100 / M) + 100.

¡MUCHAS GRACIAS!