Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2413 - Bases de Datos

Interrogación 2

Fecha: 2 de junio de 2025

1° semestre 2025 - Profesores: Eduardo Bustos - Christian Álvarez

---

INSTRUCCIONES

- Al finalizar la prueba debe **digitalizarla y subirla a Canvas** correctamente. No seguir esta instrucción los expone a un descuento de hasta 5 décimas.
- Cada hoja de respuesta debe contener su **nombre completo** y **número de lista** en la parte superior.
- Responda cada pregunta en una hoja diferente.
- Debe firmar la lista de asistencia.
- Duración 2 horas.

Pregunta 1 - Lógica en la BD

Usted como experto de BD, ha sido seleccionado para realizar una asesoría de Base de Datos al banco DCCBank. El script de creación de su base de datos es el siguiente:

```
-- Tabla de Ejecutivos
CREATE TABLE Ejecutivos (
    id_ejecutivo INT PRIMARY KEY,
    nombre VARCHAR(100),
    correo VARCHAR(100),
    telefono VARCHAR(20)
);

-- Tabla de Clientes
CREATE TABLE Clientes (
    id_cliente INT PRIMARY KEY,
    nombre VARCHAR(100),
    rut VARCHAR(12) UNIQUE,
    direccion VARCHAR(200),
    telefono VARCHAR(20),
    correo VARCHAR(100),
    id_ejecutivo INT,
    FOREIGN KEY (id_ejecutivo) REFERENCES Ejecutivos(id_ejecutivo)
);

-- Tabla de Cuentas Corrientes
CREATE TABLE CuentasCorrientes (
    id_cuenta INT PRIMARY KEY,
    numero_cuenta VARCHAR(20) UNIQUE,
    saldo DECIMAL(15,2),
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);

-- Tabla de Tarjetas de Crédito
CREATE TABLE TarjetasCredito (
    id_tarjeta INT PRIMARY KEY,
    numero_tarjeta VARCHAR(20) UNIQUE,
    limite_credito DECIMAL(15,2),
    saldo_actual DECIMAL(15,2),
    fecha_vencimiento DATE,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
```

```
);

-- Tabla de Líneas de Crédito
CREATE TABLE LineasCredito (
    id_linea INT PRIMARY KEY,
    monto_aprobado DECIMAL(15,2),
    saldo_utilizado DECIMAL(15,2),
    tasa_interes DECIMAL(5,2),
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);

-- Movimientos de Cuenta Corriente
CREATE TABLE MovimientosCuenta (
    id_movimiento INT PRIMARY KEY,
    id_cuenta INT,
    fecha TIMESTAMP,
    tipo_movimiento VARCHAR(10), -- 'abono' o 'cargo'
    (abono=depósito desde línea de cédito, es positivo; cargo=compra, es negativo)
    monto DECIMAL(15,2),
    descripcion VARCHAR(255),
    FOREIGN KEY (id_cuenta) REFERENCES CuentasCorrientes(id_cuenta)
);

-- Movimientos de Tarjeta de Crédito
CREATE TABLE MovimientosTarjeta (
    id_movimiento INT PRIMARY KEY,
    id_tarjeta INT,
    fecha TIMESTAMP,
    tipo_movimiento VARCHAR(10), -- 'pago', 'compra'
    (pago es positivo; compra es negativo)
    monto DECIMAL(15,2),
    descripcion VARCHAR(255),
    FOREIGN KEY (id_tarjeta) REFERENCES TarjetasCredito(id_tarjeta)
);

-- Movimientos de Línea de Crédito
CREATE TABLE MovimientosLineaCredito (
    id_movimiento INT PRIMARY KEY,
    id_linea INT,
    fecha TIMESTAMP,
    tipo_movimiento VARCHAR(10), -- 'pago', 'uso'
```

```
    (pago es positivo; uso=traspaso a cuenta corriente, es negativo)
    monto DECIMAL(15,2),
    descripcion VARCHAR(255),
    FOREIGN KEY (id_linea) REFERENCES LineasCredito(id_linea)
);
```

Basándose en lo anterior, responda las siguientes preguntas:

a) Cree un procedimiento almacenado (**STORED PROCEDURE**) que reciba como parámetros un rut de un cliente, fecha de inicio y fecha de fin, y con ello devuelva todos los movimientos de cuenta corriente, línea de crédito y tarjeta de crédito desde la fecha de inicio a la fecha de fin inclusive.

Los movimientos deben identificar el producto al que pertenecen, id del producto, fecha, tipo de movimiento, monto y descripción. Los movimientos deben ir ordenados desde el más antiguo al mas nuevo. (5 pts)

Solución:

```
CREATE OR REPLACE FUNCTION obtener_movimientos(
    rut_cliente VARCHAR(12),
    fecha_inicio TIMESTAMP,
    fecha_fin TIMESTAMP
    )
RETURNS TABLE (producto TEXT, id_producto INT, fecha TIMESTAMP,
tipo_movimiento VARCHAR, monto DECIMAL, descripcion VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT 'Cuenta Corriente' AS producto, mc.id_cuenta, mc.fecha,
        mc.tipo_movimiento, mc.monto, mc.descripcion
    FROM MovimientosCuenta mc
    JOIN CuentasCorrientes cc ON mc.id_cuenta = cc.id_cuenta
    JOIN Clientes c ON cc.id_cliente = c.id_cliente
    WHERE c.rut = rut_cliente AND mc.fecha BETWEEN fecha_inicio AND fecha_fin

    UNION ALL

    SELECT 'Tarjeta de Crédito', mt.id_tarjeta, mt.fecha,
        mt.tipo_movimiento, mt.monto, mt.descripcion
    FROM MovimientosTarjeta mt
    JOIN TarjetasCredito tc ON mt.id_tarjeta = tc.id_tarjeta
    JOIN Clientes c ON tc.id_cliente = c.id_cliente
    WHERE c.rut = rut_cliente AND mt.fecha BETWEEN fecha_inicio AND fecha_fin
```

```
    UNION ALL

    SELECT 'Línea de Crédito', ml.id_linea, ml.fecha, ml.tipo_movimiento,
        ml.monto, ml.descripcion
    FROM MovimientosLineaCredito ml
    JOIN LineasCredito lc ON ml.id_linea = lc.id_linea
    JOIN Clientes c ON lc.id_cliente = c.id_cliente
    WHERE c.rut = rut_cliente AND ml.fecha BETWEEN fecha_inicio AND fecha_fin

    ORDER BY fecha ASC;
END; $$ LANGUAGE plpgsql;
```

b) Cree un **TRIGGER**, que al momento de insertar un registro nuevo en la tabla `MovimientosCuenta`, actualice el saldo en la tabla de `CuentasCorrientes`. (5 pts)

Solución:

```
CREATE OR REPLACE FUNCTION actualizar_saldo_cuenta()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.tipo_movimiento = 'abono' THEN
        UPDATE CuentasCorrientes
        SET saldo = saldo + NEW.monto
        WHERE id_cuenta = NEW.id_cuenta;
    ELSIF NEW.tipo_movimiento = 'cargo' THEN
        UPDATE CuentasCorrientes
        SET saldo = saldo - NEW.monto
        WHERE id_cuenta = NEW.id_cuenta;
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_actualizar_saldo
AFTER INSERT ON MovimientosCuenta
FOR EACH ROW
EXECUTE FUNCTION actualizar_saldo_cuenta();
```

c) Cree una vista (**VIEW**), que muestre el estado financiero consolidado de los clientes para cada uno de sus productos (cuenta corriente, tarjeta de crédito, y línea de crédito). Además del saldo consolidado de cada producto, debe entregar el rut, id y nombre del cliente, así como el id y nombre del ejecutivo. (5 pts)

Solución:

```
CREATE VIEW EstadoFinancieroClientes AS (
    SELECT c.rut, c.id_cliente, c.nombre AS nombre_cliente,
        e.id_ejecutivo, e.nombre AS nombre_ejecutivo,
        COALESCE(cc.saldo, 0) AS saldo_cuenta_corriente,
        COALESCE(tc.saldo_actual, 0) AS saldo_tarjeta_credito,
        COALESCE(lc.saldo_utilizado, 0) AS saldo_linea_credito
    FROM Clientes c
    LEFT JOIN Ejecutivos e ON c.id_ejecutivo = e.id_ejecutivo
    LEFT JOIN CuentasCorrientes cc ON c.id_cliente = cc.id_cliente
    LEFT JOIN TarjetasCredito tc ON c.id_cliente = tc.id_cliente
    LEFT JOIN LineasCredito lc ON c.id_cliente = lc.id_cliente;
    );
```

d) [**Bonus**] Cree un procedimiento almacenado (**STORED PROCEDURE**) que devuelva el ejecutivo que tiene la mayor cantidad de clientes con saldo negativo en su cuenta corriente. (3 pts)

Solución:

```
CREATE OR REPLACE FUNCTION ejecutivo_mas_clientes_en_rojo()
RETURNS TABLE (
    id_ejecutivo INT,
    nombre_ejecutivo VARCHAR,
    cantidad_clientes INT
    ) AS $$
BEGIN
    RETURN QUERY
    SELECT e.id_ejecutivo, e.nombre,
        COUNT(DISTINCT c.id_cliente) AS cantidad_clientes
    FROM Ejecutivos e
    JOIN Clientes c ON e.id_ejecutivo = c.id_ejecutivo
    JOIN CuentasCorrientes cc ON c.id_cliente = cc.id_cliente
    WHERE cc.saldo < 0
    GROUP BY e.id_ejecutivo, e.nombre
    ORDER BY cantidad_clientes DESC
    LIMIT 1;
END; $$ LANGUAGE plpgsql;
```

Pregunta 2 - ACID, Transacciones y Locks

Responda cada una de las siguientes consultas

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. (2 pts c/u)

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: En 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. (3 pts)

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.

Pregunta 3 - Arquitectura de Sistemas y Recuperación de fallas

a) Para cada una de los siguientes tipos de fallas en transacciones, indique cuál es el mecanismo de manejo y/o solución, y explique cómo se usa.

i. Datos erróneos. (1 pt)

Solución: Restricciones de integridad o limpieza de datos.

ii. Falla física en el almacenamiento. (1 pt)

Solución: RAID.

iii. Catástrofes ambientales (por ej., tornado). (1 pt)

Solución: Respaldos (backups), replicación.

iv. Fallas del sistema (1 pt).

Solución: Log recovery.

b) Para la siguiente transacción, usando Undo Logging, responda las siguientes preguntas. Considere que los valores iniciales son `A=15`, `B=23`, `C=37`.

T1
R(A)
A=A+7
W(A)
R(C)
C=C+13
R(B)
B = B + C
W(C)
W(B)
COMMIT

Cuadro 2: Undo Logging

i. Con la sintaxis vista en clase, registre las operaciones que se escriben en el log. (3 pts)

Solución:

```
<START T1>
<T1, A, 15>
<T1, C, 37>
<T1, B, 23>
```

`<COMMIT T1>`

II. Usando Undo Logging, ¿qué acciones se deben realizar si sucede un error después de la ejecución de `B=B+C`?. Especifique las acciones para las variables `A`, `B` y `C`, y los valores finales después de la recuperación. (3 pts)

Solución: Como el error sucede antes del `COMMIT`, se deben deshacer todas las operaciones de escritura. La única escritura es sobre A, por lo tanto se debe deshacer la escritura sobre la variable A. Las variables B y C no es necesario realizar undo sobre ella. Los valores finales son A=15, B=23 y C=37.

c) Dada la siguiente ejecución, usando Redo Logging, se realiza un nonquiescent checkpoint después de la ejecución de `T1:COMMIT`. Aplicando la sintaxis vista en clase, escriba las operaciones que se registran en el log. Considere los valores iniciales `A=20`, `B=13`, `C=27` (5 pts)

Redo Logging
T1: START
T1: R(A)
T1: R(C)
T1: C=C+A
T2: START
T2: R(B)
T1: W(C)
T2: B = B + 3
T1: COMMIT
T3: START
T3: R(D)
T1: END
T2: W(B)
T3: COMMIT
T2: COMMIT
T3: END
T2: END

Cuadro 3: Redo Logging

Solución:

`<START T1>`
`<START T2>`
`<T1, C, 47>`

```
<COMMIT T1>
<START CKPT (T2, T3)>
<START T3>
<END T1>
<END CKPT>
<T2, B, 16>
<COMMIT T3>
<COMMIT T2>
<END T3>
<END T2>
```

Pregunta 4 - EEDD, Almacenamiento e Índices

Dada la siguiente tabla de la base de datos de la universidad, responda las siguientes preguntas:

```
CREATE TABLE Estudiantes (
    n_alumno SERIAL PRIMARY KEY,       -- Número de alumno único por carrera
    nombre VARCHAR(100) NOT NULL,      -- Nombre del estudiante
    RUN VARCHAR(10) NOT NULL,          -- Rol Único Nacional (formato NNNNNNNN-A)
    carrera VARCHAR(50) NOT NULL,      -- Nombre de la carrera
    PPA DECIMAL(3,2) NOT NULL          -- Promedio ponderado acumulado
);
```

a) ¿Qué ventaja tendría para el negocio crear un índice sobre la columna `RUN`? Explique. (3 pts)

Solución: Es muy usual que los estudiantes se sepan su RUN y menos que se sepan el número de alumno, por lo que agregar este índice secundario da un segundo mecanismo de búsqueda eficiente, si no se agrega un índice a RUN la búsqueda necesita un full scan.

b) Interprete, desde el punto de vista del negocio, cuál es el efecto, de agregar UNIQUE al parámetro `RUN`. (4 pts)

Solución: Agregar UNIQUE significa que no puede haber estudiantes con más de un programa ya sea en paralelo o consecutivas.

c) ¿Para qué tipo de búsqueda, en esta tabla, se recomienda el método B+ y Hash? (4 pts)

Solución: El método B+ se recomienda para búsquedas de rango y el método Hash para búsquedas exactas.

d) ¿Qué impacto tiene en el rendimiento de las consultas el uso excesivo de índices en una tabla? (4 pts)

Solución: El uso excesivo de índices puede degradar el rendimiento de las operaciones de escritura (INSERT, UPDATE, DELETE), ya que cada índice debe actualizarse. También aumenta el uso de almacenamiento y puede hacer más compleja la administración de la base de datos.

e) [**Bonus**] Si los índices primarios de la tabla se implementan usando B+ Clustered y realizamos una búsqueda por el campo `n_alumno=12345678`, ¿cuál es el número de I/O de la búsqueda? (Hint: suponga árbol de altura h y que cada página contiene P tuplas). (3 pts)

Solución: En B+ Clustered los datos se encuentran en las hojas del árbol. Cómo en este caso se busca un valor específico, solo es necesario navegar a través del árbol hasta la hoja y en ese caso el costo de I/O es la altura del árbol h. Si el índice está cargado en memoria el costo de I/O es despreciable.

Torpedo

Sintaxis consulta SQL

```
SELECT [ALL | DISTINCT] * | [COUNT, MAX, MIN, SUM, AVG](col1), col2, ...
       [AS alias_column]
FROM table1
     [JOIN table2 ON table1.common_field = table2.common_field]
WHERE condition1 [AND | OR condition2]
      [BETWEEN value1 AND value2]
      [IN (value1, value2, ...)]
      [LIKE pattern]
      [= | <> | <= | >= | < | >] value
GROUP BY column1, column2, ...
HAVING aggregate_condition [COUNT, MAX, MIN, SUM, AVG]
ORDER BY column1 [ASC | DESC]
UNION | INTERSECT | EXCEPT
[SELECT column1 FROM another_table]
LIMIT row_count;
```

Sintaxis Stored Procedure

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

Sintaxis Trigger

```
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;
```

Sintaxis View

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

Sintaxis PL/pgSQL

```
BEGIN
    IF <condición> THEN
        <cuerpo_if>
    ELSE
        <cuerpo_else>
    END IF;
    INSERT INTO <tabla> VALUES (<tupla_valores>);
    FOR <variable> IN <iterable>
    LOOP
        <cuerpo_loop>
    END LOOP;
    RETURN QUERY <consulta_SQL>;
END;
```

Sintaxis Índices

```
CREATE [CLUSTERED | UNCLUSTERED] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ]
name ] ON [ ONLY ]     table_name [ USING method ]
( { column_name | ( expression ) } [ COLLATE collation ]
[ opclass [ ( opclass_parameter = value [, ... ] ) ] ]
[ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
[ INCLUDE ( column_name [, ...] ) ]
[ NULLS [ NOT ] DISTINCT ]
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ TABLESPACE tablespace_name ]
[ WHERE predicate ]
```