BD-GEI - Examen final **15.1.2021**

Temps: 2 hores

Notes 27 gener Revisió: 28 gener **Cada pregunta s'ha de lliurar en un full separat**

1. **(1 punt)** Donades les tres taules següents:

```
professors(dni, nomProf, telefon, sou)
despatxos(modul, numero, superficie)
assignacions(dni, modul, numero, instantInici, instantFi)
      {dni} referencia professors
      {modul,numero} referencia despatxos
```

Considereu el següent fragment de codi Java que implementa, utilitzant JDBC, l'assignació de tots els professors que tenen un sou superior al valor de la variable sou al despatx identificat per les variables modul i numero, amb instancInici = 100 i instanFi = null. En cas que algun dels profesors ja estigui assignat al despatx s'acaba el programa mostrant un missatge d'error.

```
try {
 /* Tenim ja una connexió c establerta amb la base de dades */
 int sou = 1200;
 String modul = "omega";
 String numero = "118";

 Statement st1 = c.createStatement();
 ResultSet rs1 = st1.executeQuery("select dni from professors where sou < " + sou);


 while (rs1.next()) {
   String dniProf = rs1.getString(1);

   // Consultem si el professor ja ha estat assignat
   Statement st2 = c.createStatement();
   ResultSet rs2 = st2.executeQuery("select * from assignacions"
                     + " where dni='" + dniProf + "'"
                     + " and modul='" + modul + "'"
                     + " and numero='" + numero + "'");

   // Si rs2 retorna resultat, aleshores el professor ja està assignat
   if (rs2.next()) {
         System.out.println("Error: Un dels professors ja està assignat");
         c.rollback();
         break;    /* surt del bucle  */
   }
   // En cas contrari, procedim a crear l'assignació
   else {
         Statement st3 = c.createStatement();
         st3.executeUpdate("insert into assignacions values ("
                          + "'" + dniProf + "',"
                          + "'" + modul + "',"
                          + "'" + numero + "',"
                          + "100, null)");
         st3.close();
    }
    rs2.close(); st2.close();
    }
 st1.close(); rs1.close();c.commit(); c.close();
 }
catch (ClassNotFoundException ce) {
         System.out.println ("Error al carregar el driver");}
catch (SQLException se) {
         System.out.println ("Excepcio: "+ se.getMessage());}
```

Donat aquest fragment de codi, identifiqueu quins criteris de qualitat no es compleixen. Per cadascun d'ells, expliqueu perquè no es compleix, i expliqueu amb detall quins canvis serien necessaris en la codificació per tal de satisfer aquests criteris.

SOLUCIÓ:

No es compleixen els següents criteris de qualitat:

- 1 - Execució de sentència SQL innecessària. La comprovació feta per st2 sobre l'existència de l'assignació és innecessària, ja que aquesta comprovació es pot fer simplement mitjançant l'execució de l'operació insert, delegant la responsabilitat al propi gestor de bases de dades, i capturant/gestionant la SQLException com correspongui.
- 2 - Utilització inadequada de Statement. Tant st2 com st3 utilitzen Statement, però aquestes operacions s'executen per a cada valor dniProf obtingut a la consulta st1. Per tant, el més adequat seria utilitzar PreparedStatement establint com a variable de les operacions el valor dniProf. En el cas de st2 , i en relació al criteri de qualitat 1, aquest criteri es solucionaria també simplement suprimint aquesta comprovació.
- 3 – Execució de sentència SQL innecessària. Es podria també implementar el programa amb una única sentència SQL feta com a un Statement que faci un insert amb una subconsulta que determina els professors que s'ha d'inserir. En cas que algun dels professors ja està assignat, es genera el missatge quan es captura l'excepció afegida segons el criteri de qualitat 1.

2. **(2 punts)** Considereu una base de dades amb les taules següents:

```
cantants (nom, pais, anyRetir)
albums (titolAlbum, nomCantant, anyPublicacio, preu)
         {nomCantant} referencia cantants
critiques (titolAlbum, nomCantant, font, puntuacio)
         {titolAlbum, nomCantant} referencia albums
estadístiques (nomCantant, millorPuntuació)
         {nomCantant} referencia cantants
```

Indiqueu quins són els triggers necessaris (tenint en compte els criteris de qualitat establerts a l'assignatura) a definir per tal que:

- **RI1.** Es generi una excepció en cas que un esdeveniment faci que no es compleixi la següent restricció: Un cantant no pot publicar nous àlbums en els anys posteriors a l'any de retir.

- **RI2.** Es mantingui el valor de l'atribut *millorPuntuació*. Aquest atribut ha de tenir com a valor, la puntuació més alta que ha rebut un cantant en els àlbums que ha publicat.
- **RI3.** Es generi una excepció en cas que un esdeveniment faci que no es compleixi la següent restricció: No es poden afegir crítiques si la base de dades conté menys de 10 cantants.

Per cada trigger cal indicar:

- Taula per a la que es defineix
- Esdeveniment que l'activa (INSERT, DELETE, UPDATE - en cas d'UPDATE indicar atribut/s rellevants)
- BEFORE/AFTER (justificar la resposta)
- ROW/STATEMENT (justificar la resposta)

SOLUCIÓ

	Trigger RI1	Trigger RI1	Trigger RI2	Trigger RI3
Taula	cantants	albums	critiques	critiques
Esdeveniments	UPDATE any_retir	INSERT UPDATE anyPublicacio	INSERT, DELETE UPDATE puntuacio	INSERT
before/after	BEFORE (1)	BEFORE (1)	AFTER (3)	BEFORE (1)
row/statement	ROW (2)	ROW (2)	ROW (4)	STATEMENT (5)

(1) Si fossin AFTER, en cas de violar-se la restricció caldria desfer l'esdeveniment sobre la taula cantants/albums/critiques
(2) Al ser ROW poden implementar-se de manera incremental, fent les comprovacions de la restricció només per a les files inserides/modificades
(3) Si fossin BEFORE, en cas que els esdeveniments no s'acabessin executant caldria desfer els canvis de l'atribut *millorPuntuació* fets.
(4) Al ser ROW pot implementar-se de manera incremental, només aplicant canvis a l'atribut millorPuntuació dels cantants als que se'ls ha fet canvis en les crítiques.
(5) La restricció no té a veure amb quines files s'insereixen ni amb quantes files s'insereixen, per tant amb un STATEMENT n'hi ha prou.

3. **(2 punts)** Suposeu la base de dades següent, que s'ha creat en un SGBD sense cap mecanisme de control de concurrència. Considereu que el gránul és la fila.

```
autors(   codiAutor,  nom,        telefon,   poblacio)
          ca1         Autor1      111        Pobl1
          ca2         Autor2      222        Pobl2

obres(    idObra,     codiAutor,  titol,     anyEdicio)
          ob1         ca1         Bon dia    2020
          ob2         ca2         Hola       2020
          ob3         ca1         Bona nit   2020
```

Suposeu també les sentències SQL següents:

s1	UPDATE obres SET codiAutor = 'ca1' WHERE idObra= 'ob2';
s2	DELETE FROM obres WHERE idObra='ob1';
s3	INSERT INTO obres VALUES('ob4','ca1','Adeu',2020);
s4	SELECT * FROM obres WHERE codiAutor='ca1';
sX és un codi que us pot permetre fer referència a les sentències	

3.1 Doneu un horari on intervinguin dues transaccions, que poden executar una o més de les sentències SQL donades, i que presenti una interferència d'ACTULITZACIÓ PERDUDA. Una sentència es pot usar únicament una vegada en l'horari, i s'ha d'incloure a l'horari el mínim número de sentències. En cas que no sigui possible formar un horari amb els requisits indicats, justifiqueu perquè no.

3.2 Doneu un horari on intervinguin dues transaccions, que poden executar una o més de les sentències SQL donades, i que presenti una interferència de FANTASMA. S'ha d'incloure a l'horari el mínim número de sentències, però una sentència es pot usar més d'una vegada. En cas que no sigui possible formar un horari amb els requisits indicats, justifiqueu perquè no.

SOLUCIO

3.1) Si volem actualització perduda, les dues transaccions han de fer una actualització (RU,W) sobre el mateix gränul. Les tres operacions d'actualització proposades treballen sobre gränuls diferents així que NO és possible construir un horari amb aquesta interferència.

3.2) Si volem que es produeixi una interferència de tipus FANTASMA, una de les dues transaccions ha de 'canviar' el contingut de la BD. Per exemple, una transacció ha de fer l'UPDATE o l'INSERT (amb una instrucció n'hi ha prou, hem de fer servir el mínim nombre d'instruccions) i l'altra transacció ha de fer dues lectures semblants però forçant que donin resultats diferents. Per tant, ficarem l'UPDATE/INSERT en mig de les dues lectures:

T1	T2
SELECT * FROM obres WHERE codiAutor='ca1';	
	INSERT INTO obres values('ob4','ca1','Adeu',2020,'g1'); -- o bé UPDATE obres SET codiAutor = 'ca1' WHERE idObra= 'ob2';
SELECT * FROM obres WHERE codiAutor='ca1';	
	commit
commit	

4. **(2 punts)** Considereu l'horari següent:

	T1	T2
1	R(A)	
2		RU(A)
3		W(A)
4		RU(B)
5		W(B)
6	RU(B)	
7	W(B)	
8	COMMIT	
9		COMMIT

Suposeu que s'incorporen tècniques de control de concurrència amb reserves.

4.1 En cas que les transaccions treballin en nivell d'aïllament READ UNCOMMITTED.

a) Doneu l'horari aplicant reserves corresponents al nivell d'aïllament (*)

b) Doneu el graf de precedències de l'horari resultant d'aplicar reserves

c) En cas que hi hagi alguna interferència en l'horari resultant digueu quina/es són, i el/s gràNul/s implicat/s.

4.2 En cas que les transaccions treballin en nivell d'aïllament REPETEABLE READ.

a) Doneu l'horari aplicant reserves corresponents al nivell d'aïllament (*)

b) Doneu el graf de precedències de l'horari resultant d'aplicar reserves

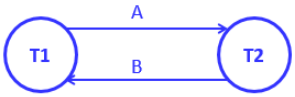
c) En cas que hi hagi alguna interferència en l'horari resultant digueu quina/es són, i el/s gràNul/s implicat/s.

(*) En els horaris heu d'incloure, a més de les operacions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves (LOCK, UNLOCK), i l'ordre d'execució de totes aquestes operacions. Quan més d'una transacció espera per un mateix gràNul, considereu que la política de la cua és FIFO (First In, First Out).

SOLUCIÓ

4.1)

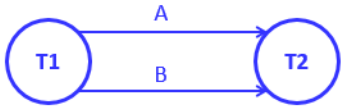
	T1	T2
1	R(A)	
2		LOCK(A,X)
3		RU(A)
4		W(A)
5		LOCK(B,X)
6		RU(B)
7		W(B)
8	LOCK(B,X)	
9	...	COMMIT+UNLOCKS (A,B)
10	RU(B)	
11	W(B)	
12	COMMIT+UNLOCK (B)	



Hi ha una interferència d'anàlisi inconsistent amb gràNuls A,B

4.2)

	T1	T2
1	LOCK(A,S)	
2	R(A)	
3		LOCK(A,X)
4	LOCK(B,X)	...
5	RU(B)	...
6	W(B)	...
7	COMMIT+UNLOCKS (A,B)	...
8		RU(A)
9		W(A)
10		LOCK(B,X)
11		RU(B)
12		W(B)
13		COMMIT+UNLOCKS (A,B)



No hi ha cap interferència

5. **(2 punts)** Suposeu la taula *albums*:

```
albums (titolAlbum, nomCantant, anyPublicacio, preu)
```

Aquesta taula té 2.000.000 files, i el factor de bloqueig, de les pàgines de dades on s'emmagatzemen aquestes files, és de 40 files per pàgina.

Sabem que el cantant `'Juli Catedrals'` ha fet 500 àlbums. També sabem que la condició `titolAlbum > 'xxx'` la compleixen 8.000 àlbums que han fet uns 1.000 cantants diferents.

Considereu la consulta següent.

```
SELECT *
FROM albums
WHERE nomcantant = 'Juli Catedrals'
     AND titolAlbum > 'xxx';
```

Doneu el cost en número de pàgines d'índex i número de pàgines de dades a les que cal accedir per resoldre la consulta:

5.1 Usant un índex agrupat definit sobre l'atribut nomCantant, amb ordre d=24, ple al 80%.

5.2 Usant un índex no agrupat definit sobre l'atribut titolAlbum, amb ordre d=16, ple al 70%

SOLUCIÓ:

5.1) Número pàgines d'índex + Número pàgines de dades = 4 + 13 = 17 pàgines

Número de pàgines d'índex = nivells de l'arbre = $\lceil \log_{40} 2.000.000 \rceil = 4$

número de valors per node = $\lceil 2d*0,8 \rceil = \lceil 2*24*0,8 \rceil = 39$ valors / node

número d'apuntadors per node = 39 + 1 = 40 apuntadors / node

Número de pàgines de dades = $\lceil$ (albums(nomcantant = 'Juli Catedrals') / factor de bloqueig $\rceil$

$= \lceil 500 / 40 \rceil = 13$

albums(nomcantant = 'Juli Catedrals') = 500

factor de bloqueig = 40

5.2) Número pàgines d'índex + Número pàgines de dades = 352 + 8.000 = 8.352 pàgines

Número de pàgines d'índex = nivells de l'arbre

$+ \lceil$ (albums(titolAlbum > 'xxx') / num valors/node $\rceil$ -1

$= \lceil \log_{24} 2.000.000 \rceil + \lceil 8.000 / 23 \rceil - 1 = 5 + 347 = 352$ pàgines

número de valors per node = $\lceil 2d*0,7 \rceil = \lceil 2*16*0,7 \rceil = 23$ valors / node

número d'apuntadors per node = 23 + 1 = 24 apuntadors / node

Número de pàgines de dades = albums(titolAlbum > 'xxx') = 8.000

6. **(1 punt)** Considereu un índex arbre B+ amb els següents paràmetres: el valor pel que es crea l'índex té V bytes, la mida dels nodes és B bytes, els RID tenen T bytes, i els apuntadors a nodes de l'índex tenen A bytes.

6.1 Sigui nf la quantitat màxima de valors que un node del nivell fulla pot contenir. Expresseu nf en funció dels paràmetres V, B, T, A

6.2 Suposa ara que 2d = 100.

a) Quin és el número màxim de files que el arbre B+ pot apuntar si té 3 nivells?

b) Quants nodes es necessiten per emmagatzemar l'índex en aquest cas?

SOLUCIÓ:

6.1) nf = $\lfloor (B-2A) / (V + T) \rfloor$

6.2.a) files = 101*101*100 = 1,020,100

6.2.b) nodes = 1+ 101 + 101*101 =10303

Temps: 2,5 hores

Notes 22 juny Revisió: 28 juny **Cada pregunta s'ha de lliurar en un full separat**

1. **(1 punt)** Donades les base de dades següent:

```
create table professors
(dni char(50),
nomProf char(50) unique,
telefon char(15),
primary key (dni));

create table despatxos
(modul char(5),
numero char(5),
superficie integer not null check(superficie <=25),
primary key (modul,numero));

create table assignacions
(dni char(50),
modul char(5),
numero char(5),
instantInici integer,
instantFi integer,
primary key (dni, modul, numero, instantInici),
foreign key (dni) references professors,
foreign key (modul,numero) references despatxos,
check (instantInici < instantFi));
```

Considereu el següent fragment de codi Java/JDBC. Implementeu el cos del **while** per tal que s'incrementi en 5 unitats la superfície dels despatxos de cadascun dels mòduls obtinguts en la variable varModul. Definiu també els Statements o PreparedStatements necessaris on creieu convenient. Cal que el programa tregui un missatge d'excepció si la superfície d'algun despatx passa a ser superior a 25. En cas que no hi hagi cap excepció el programa indicarà el número de despatxos modificats de cada mòdul.

```
try {
 /* Tenim ja una connexió c establerta amb la base de dades */
 /* Tenim un variable in que permet llegir el que escriu l'usuari */

 System.out.print("Escriu el nom d'un mòdul: ");
 String varModul = in.nextLine();

 while (!varModul.equals("acabar")) {

     //codi a implementar

     System.out.print("Escriu el nom d'un mòdul: ");
     varModul = in.nextLine();
     }
 c.commit();
 }
catch (ClassNotFoundException ce) {
        System.out.println ("Error al carregar el driver");}
catch (SQLException se) {
        System.out.println ("Excepcio: "+ se.getMessage());}
```

Recordeu que podeu usar els mètodes/codis d'excepció de JDBC estudiats a classe:

Statements

- Statement createStatement();
- ResultSet executeQuery(String sql);
- int executeUpdate(String sql);

PreparedStatement

- PreparedStatement prepareStatement(String sql);
- ResultSet executeQuery();
- int executeUpdate();
- void setXXX(int posicioParametre, XXX valor);

ResultSet

- boolean next();
- XXX getXXX(String nomColumna)

SQLException

- // 23502 -not_null_violation, 23503 -foreign_key_violation
- // 23505 -unique_violation, 23514 -check_violation
- String getSQLState();

SOLUCIÓ

```
try {
    /* Tenim ja una connexió c establerta amb la base de dades */

    System.out.println("Escriu el nom d'un mòdul: ");
    String varModul = System.console().readLine();

    PreparedStatement ps = c.prepareStatement("update despatxos d "+
                                    "set superficie = superficie + 5 "+
                                    "where d.modul = ?");


    while (!varModul.equals("acabar")) {

        //codi a implementar
        ps.setString(1,varModul);
        int numFilesActualitzades = ps.executeUpdate();
        System.out.println (varModul+"  "+numFilesActualitzades);
        System.out.println("Escriu el nom d'un mòdul: ");
        varModul = System.console().readLine();
        }
    c.commit();
    }
   catch (ClassNotFoundException ce) {
             System.out.println ("Error al carregar el driver");}
   catch (SQLException se) {
            if(se.getSQLState().equals("23514"))
                  System.out.println("La superficie no pot ser més gran que 25");
            else
                  System.out.println ("Excepcio: "+ se.getMessage());}
```

2. (1 punt) Suposeu la base de dades següent:

```
empleats1 (numEmp1, nomEmp1, ciutatEmp1);
empleats2 (numEmp2, nomEmp2, ciutatEmp2);
```

Suposeu que es vol implementar la restricció següent mitjançant disparadors:

Els valors de l'atribut `ciutatEmp1` de la taula `empleats1` han d'estar inclosos en els valors de l'atribut `ciutatEmp2` de la taula `empleats2`

La idea és llançar una excepció en cas que s'intenti executar una sentència que violi la restricció.

2.1 Digueu quins són els esdeveniments rellevants (taula/es i esdeveniment/s).

empleats1	Insert	
empleats1	Update	ciutatEmp1
empleats2	Delete	
empleats2	Update	ciutatEmpl2

2.2 Digueu i justifiqueu de quin tipus han de ser el disparadors (ROW / STATEMENT, BEFORE/AFTER).

Tots els disparadors han de ser before, for each row.

Before perquè com que es tracta de que salti una excepció quan es violi una restricció d'integritat, com més aviat es descarti que la restricció es viola millor, per tal d'evitar fer feina innecessària.

For each row perquè és millor una solució incremental. Suposant que a cada taula hi ha una gran quantitat d'empleats, i que els esdeveniments afecten a poques files, una solució incremental sempre comportarà menys accessos a la base de dades, perquè només requereix fer comprovacions per a les tuples que s'han inserit/esborrat/modificat.

2.3 Implementeu un dels disparadors que hagueu identificat per a la taula empleats1. En cas de que no n'hi hagi cap justifiqueu perquè. Podeu fer servir la plantilla següent.

```
CREATE FUNCTION comprovarCiutat() RETURNS trigger AS $$
    BEGIN
     //implementació
    END;
$$LANGUAGE plpgsql;

CREATE TRIGGER ex3_3
    BEFORE/AFTER <esdeveniment>
    FOR EACH ROW/STATEMENT EXECUTE PROCEDURE comprovarCiutat();
```

```
CREATE FUNCTION comprovarCiutEmpl1() RETURNS trigger AS $$
    BEGIN
     IF  not exists(select* from empleats2
         where  ciutatEmp2=new. ciutatEmp1) THEN
         RAISE EXCEPTION 'Error en inserir o modificar empleats1';
     END IF;
     RETURN NEW;
    END;
$$LANGUAGE plpgsql;

CREATE TRIGGER ex3_3
    BEFORE INSERT OR UPDATE of ciutatEmp1 ON empleats1
    FOR EACH ROW EXECUTE PROCEDURE comprovarCiutEmpl1();
```

3. **(2 punt)** Considereu la base de dades de l'exercici 1 amb el contingut següent:

Professors	dni	nomProf	telefon	sou
	111	Joana	97743121	10000
	222	Martí	93818903	3000
	333	Lourdes	97261201	50000

Despatxos	modul	numero	superficie
	C6	101	13
	Omega	300	8
	Omega	301	8

Assignacions	dni	modul	numero	instantIni	instantFi
	333	C6	101	5	9
	111	C6	101	3	NULL
	222	Omega	300	10	NULL

Considereu també el procediment emmagatzemat següent:

```
CREATE TYPE despatx AS (d_modul char(5), d_numero char(5));

CREATE FUNCTION assignaProfessor(dni_p char(50), modul_d char(5),
                                 numero_d char(5), inici_a integer)
            RETURNS SETOF despatx AS $$
DECLARE desp despatx;
BEGIN
  IF (NOT EXISTS(SELECT * FROM Professors WHERE dni=dni_p)
       OR NOT EXISTS(SELECT * FROM Despatxos
                     WHERE modul=modul_d AND numero=numero_d)) THEN
       RAISE EXCEPTION 'El professor o el despatx no existeix';
     ELSIF (EXISTS (SELECT * FROM Assignacions a
                    WHERE a.dni=dni_p AND a.instantFi IS NULL)) THEN
          UPDATE Assignacions SET instantFi = inici_a - 1
          WHERE dni=dni_p AND instantFi IS NULL;
   END IF;
   INSERT INTO Assignacions VALUES (dni_p,modul_d,numero_d,inici_a,NULL);
   FOR desp IN SELECT d.modul,d.numero
                  FROM Professors p NATURAL JOIN Assignacions a
                                    NATURAL JOIN Despatxos d
               WHERE a.instantFi IS NULL
               GROUP BY d.modul, d.numero
               HAVING COUNT(*) >= 2
    LOOP
      RETURN NEXT desp;
    END LOOP;
   EXCEPTION
       WHEN raise_exception THEN raise exception '%',sqlerrm;
 END;
 $$LANGUAGE plpgsql;
```

3.1 Indiqueu i justifiqueu quin és l'estat de la base de dades i el valor de retorn després de l'execució de la consulta: `SELECT * FROM assignaProfessor('111','Omega','300',15);`

Les taules Professors i Despatxos no es modifiquen, el contingut de la taula Assignacions és:

Dni	modul	numero	instantIni	instantFi
333	C6	101	5	9
111	C6	101	3	14
222	Omega	300	10	NULL
111	Omega	300	15	NULL

El procediment ha finalitzat totes les assignacions actives del professor amb dni_p (posant com a instant fi el d'inici menys 1), i n'ha creat una de nova per al despatx donat com a paràmetre. L'execució de la consulta retorna:

d_modul	d_numero
Omega	300

els quals corresponen als despatxos compartits (més d'un professor assignat) amb assignacions actives (instantFi amb valor NULL).

3.2 Digueu quins són els criteris de qualitat que no es compleixen en la implementació del procediment donat i com caldria canviar el codi per tal que es complíssin.

- Accessos innecessari a la BD.
  - El primer IF amb els seus SELECTs no cal. Per tant els SELECTs són innecessaris perquè l'excepció es pot capturar mitjançant la violació de la condició de foreign key que saltarà en fer el INSERT de l'assignació nova.
  - El ELEIF amb els seu SELECT no calen. El UPDATE ja només modifica les assignacions amb instantFi NULL si és que n'hi ha alguna.
- TAULES+JOINs innecessàries
  - En el SELECT que serveix per buscar el resultat que ha de donar el procediment, les taules Professors i Desspatxos no són necessàries.

4. **(3 punts)** Transaccions:

4.1 Suposeu ara l'horari següent.

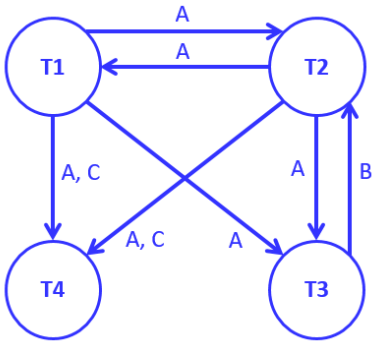
a) Doneu el graf de precedències per aquest horari.

b) En cas que hi hagi alguna interferència en l'horari resultant digueu quina/es són, les transaccions i el/s gràNul/s implicat/s.

c) És serialitzable? En cas afirmatiu doneu el horari serial equivalent. En cas negatiu perquè no.

	T1	T2	T3	T4
10			R(B)	
20		RU(A)		
30	RU(A)			
40		W(A)		
50		R(C)		
60		R(F)		
70	W(A)			
80				R(A)
90		RU(B)		
100	R(C)			
110				RU(C)
120		W(B)		
130			R(A)	
140				W(C)
150	commit			
160		commit		
170			commit	
180				commit

a) Graf de precedències.



b) Hi ha una actualització perduda entre T1 i T2 i un anàlisi inconsistent entre T2 i T3
c) No és serialitzable perquè hi ha interferències.

4.2 En cas que les transaccions treballin en nivell d'aïllament READ COMMITTED. Doneu l'horari un cop aplicades les reserves corresponents al nivell d'aïllament. En l'horari heu d'incloure, a més de les operacions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves (LOCK, UNLOCK), i l'ordre d'execució de totes aquestes operacions. Quan més d'una transacció espera per un mateix gràNul, considereu que la política de la cua és FIFO (First In, First Out).

T1	T2	T3	T4
		L(B,S)	
		R(B)	
		U(B)	
	L(A,X)		
	RU(A)		
L(A,X)			
	W(A)		
	L(C,S)		
	R(C)		
	U(C)		
	L(F,S)		
	R(F)		
	U(F)		
			L(A,S)
	L(B,X)		
	RU(B)		
	W(B)		
		L(A,S)	
	C+U(A,B)		
RU(A)			
W(A)			
L(C,S)			
R(C)			
U(C)			
C+U(A)			
			R(A)
			U(A)
			L(C,X)
			RU(C)
		R(A)	
		U(A)	
			W(C)
		C	
			C+U(C)

4.3 Considereu ara un SGBD sense cap mecanisme de control de concurrència, on el gràNul és la fila. I considereu l'horari següent que té una interferència d'anàlisi inconsistent. Expliqueu la interferència d'anàlisi inconsistent en base a l'horari i al contingut de la base de dades donat en l'exercici 3.

	T1	T2
1	select * from despatxos where modul='Omega' and numero=300;	
2		update despatxos set superficie = superficie + 8 where modul='Omega' and numero=300;
3		select * from despatxos where modul='Omega' and numero=301;
4	update despatxos set superficie = superficie + 5 where modul='Omega' and numero=301;	
5	Commit	
6		Commit

La interferència d'anàlisi inconsistent té a veure amb la visió que dues transaccionstenen d'un conjunt de dades. En aquest horari el conjunt de dades són les files 'Omega', 300 i 'Omega', 301.

En aquest horari:

- T1 llegeix les files amb valors 'Omega', 300, 8 i 'Omega', 301, 8.
- T2 llegeix les files amb valors 'Omega', 300, 8 i 'Omega', 301, 8

L'horari no és correcte perquè no dóna el mateix resultat que cap dels dos horaris serial.

En cas de l'horari seria T1;T2:

- T1 llegeix les files amb valors 'Omega', 300, 8 i 'Omega', 301, 8
- T2 llegeix les files amb valors 'Omega', 300, 8 i 'Omega', 301, 13

En cas de l'horari seria T2;T1:

- T2 llegeix les files amb valors 'Omega', 300, 8 i 'Omega', 301, 8
- T1 llegeix les files amb valors 'Omega', 300, 16 i 'Omega', 301, 8

5. **(3 punt)** Considereu la taula R(a,b,c,d). Aquesta taula té 100.000 files, i el factor de bloqueig, de les pàgines de dades on s'emmagatzemen aquestes files, és de 10 files per pàgina.

Considereu la consulta següent.

```
SELECT *
FROM R
WHERE b=5 and c >=50
```

Se sap que hi ha 70 files que compleixen la condició b=5, 100 files la condició c>=50 i només 1 fila les dues condicions alhora.

IMPORTANT: En tots els apartats cal detallar tots els càlculs que feu per calcular els costos.

5.1 Suposant que <u>només podem definir un únic índex, arbre B+, per un únic atribut</u>, i que aquest índex tindrà una ocupació del 80% i un ordre d=50, Quin tipus d´índex (agrupat o no agrupat) escolliríeu i per quin atribut hauria d'estar definit per tal de fer la consulta el més eficient possible? Justifiqueu les respostes en base els costos de totes les opcions possibles.

SOLUCIÓ:

Les opcions possibles són definir un únic índex agrupat o no agrupat pels atributs b o c, i aplicar la resta de condicions a mida que es van obtenint les files de les pàgines de dades. Qualsevol altre índex no augmentaria l'eficiència de la consulta.

Índex agrupat R.b: Cost = h +D = 3 +7 = 10

Valors per node = $\lceil 2d*0,8 \rceil = \lceil 2*50*0,8 \rceil$ = 80 valors / node

h = $\lceil \log_{81} 100000 \rceil$ = 3

D= $\lceil$ card(R.b=5) / 10 $\rceil = \lceil 70 / 10 \rceil = 7$

Índex no agrupat R.b: Cost = h + F + card(R.b=5) = 3 +0 + 70 = 73

h = $\lceil \log_{81} 100000 \rceil$ = 3

F= $\lceil 70 / 80 \rceil$ - 1= 0

card(R.b=5) = 70

Índex agrupat R.c: Cost = h +D = 3 +10 = 13

Valors per node = $\lceil 2d*0,8 \rceil = \lceil 2*50*0,8 \rceil$ = 80 valors / node

h = $\lceil \log_{81} 100000 \rceil$ = 3

D= $\lceil$ card(R.c>=50) / 10 $\rceil = \lceil 100 / 10 \rceil = 10$

Índex no agrupat R.c: Cost = h + F + card(R.c > 50) = 3 +1 + 100 = 104

h = $\lceil \log_{81} 100000 \rceil$ = 3

F= $\lceil 100/ 80 \rceil$ - 1= 1

card(R.c > =50) = 100

La millor opció serà definir un índex agrupat per l'atribut R.b, amb un cost de 10 accessos a disc.

5.2 Suposant ara que en lloc d'un únic índex per un únic atribut, en podem definir 2 també per un únic atribut i del mateix tipus que abans (una ocupació del 80% i un ordre d=50) quin seria el cost fent servir l'estratègia de intersecció de RIDs?

Cost índex per R.b = h + F = 3

Cost índex per R.c = h + F = 4

Cost per acedir a dades = Card(R.b=5 and R.c >50) = 1

El cost seria 8 accesos a disc

5.3 Suposant ara que podem definit un índex no agrupat multiatribut pels atributs b, c. Quin seria el cost fent servir aquest índex en cas de tenir ordre d=25 i ocupació 80%?

el cost seria: h +F +Card(R.b=5 and R.c >=50) = 4 + 0 +1 = 5

Valors per node = $\lceil 2d*0,8 \rceil = \lceil 2*25*0,8 \rceil$ = 40 valors / node

h = $\lceil \log_{41} 100000 \rceil$ = 4

F = F= $\lceil 1/ 80 \rceil$ - 1= 0

1. (1 punt) Donades les taules següents.

```
assignatures(idAssignatura,nomAssignatura,credits)

estudiants(idEstudiant,nomEstudiant,username)

assignaturaMatriculada(idEstudiant,idAssignatura,quadrimestre,nota)
   {idEstudiant} referencia estudiants
   {idAssignatura} referencia assignatures
```

Considereu el següent fragment de codi Java/JDBC. El programa té dos parts. En la primera part es mostra en quantes assignatures han estat matriculats cadascun dels estudiants indicats en la variable *ids*. En la segona part s'esborra de la base de dades una assignatura identificada en la variable *id* si no hi ha cap estudiant que s'hi hagi matriculat; si a l'assignatura s'hi ha matriculat algun estudiant, aleshores es dóna un missatge d'error.

```
01  /* Tenim ja una connexió c establerta amb la base de dades */
02
03  /* consultar assignatures */
04  ResultSet rs = null;
05  int[] ids = {1, 2, 3};
06  PreparedStatement ps = c.prepareStatement("select distinct idAssignatura "+
07                                            "from assignaturaMatriculada "+
08                                            "where idEstudiant = ?");
09  for(int i=0; i<ids.length;i++){
10     ps.setInt(1,ids[i]);
11     rs = ps.executeQuery();
12     int quantesAssignatures=0;
13     while(rs.next()){quantesAssignatures = quantesAssignatures + 1;}
14     System.out.println(ids[i] + " ha estat matriculat en "
15                               + quantesAssignatures + " assignatures!");
16     }
17
18  /* esborrar assignatura */
19  int id = 4;
20  Statement stCons = c.createStatement();
21  Statement stDel = c.createStatement();
22  rs = stCons.executeQuery("select count(*) as quants "+
23                           "from assignaturaMatriculada "+
24                           "where idAssignatura = "+id);
25  if (rs.next() && rs.getInt("quants")>0)
26      {System.out.println("Error: no es pot esborrar assignatura "+
27                          "perquè hi ha estudiants matriculats a"+id);}
28    else {stDel.executeUpdate("delete from assignatures where idAssignatura ="+id);}
29
30 /* Commit i desconnexio de la base de dades */
```

Donat aquest fragment de codi, identifiqueu quins dels criteris de qualitat estudiats a l'assignatura no es compleixen. Per cadascun d'ells:

- Expliqueu perquè no es compleix
- Expliqueu en detall quins canvis seria necessari fer en el codi donat per tal de que es complexi.

Solució

- Primera part - Part del codi és redundant
  - No és necessari el WHILE que calcula la quantitat d'assignatures matriculades d'un estudiant. Cal canviar el SELECT per buscar directament aquest total en el mateix SELECT.
  - El nou SELECT, que substitueix l'existent en línies 05..07:

```
select count(distinct idAssignatura) as total
from students_marks where student_id=?
```

  - S'eliminarà el WHILE de la línia 12, i la línia 11 canviarà de la manera indicada a continuació:

```
rs.next();
int quantesAssignatures = rs.getInt("total");
```

- Segona part – Select innecessari
  - Es pot saber si hi ha l'error simplement executant el DELETE, i recullint en cas que es produeixi l'error de clau forana. Per tant, no es necessari fer el SELECT, i s'ha de recollir l'excepció amb un CATCH.
  - S'eliminarà les línies de codi 19, 21-23 i 24
  - El DELETE de la línia 27 es farà just després de crear el Statement de la línia 20

```
stDel.executeUpdate("delete from assignatures where idAssignatura ="+id);
```

  - Les línies 18, 19 i la nova línia amb el DELETE estaràn en try, i s'afegirà un catch que contidrà les línies 25 i 25 dins de un IF.

```
catch (SQLException se)
        { if (se.getSQLState().equals("23503")
          {System.out.print("Error: ....  ");}
```

2. (2.5 punts) Considereu la base de dades següent.

```
tipusProductes(idTipus, nomTipus, preuMaxim)

productes(nomProducte, idTipus, preu)
    {idTipus} references tipusProductes

comandes(numComanda, import)

productesComanda(numComanda, nomProducte, quantitat)
    {numComanda} references comandes
    {nomProducte} references productes
```

2.1 Suposeu que les taules de la base de dades estan inicialment buides i que executem una crida al següent procediment emmagatzemat *afegir_producte()*. Quin és l'estat de la taula *productes* després de l'execució? Justifiqueu la resposta.

```
CREATE or replace FUNCTION afegir_producte() RETURNS SETOF char(20)
AS $$
    DECLARE nom char(20);
    BEGIN
        FOR nom IN SELECT nomProducte FROM productes
            LOOP RETURN NEXT nom;
            END LOOP;
        IF NOT FOUND THEN
            RAISE EXCEPTION 'Error inesperat';
        END IF;
        INSERT INTO tipusProductes VALUES (1, 'aperitius', 10);
        INSERT INTO productes VALUES ('patates', 1, 4);
    RETURN;
END; $$LANGUAGE plpgsql;
```

En no haver-hi files a productes, el programa no entra en el FOR i per tant FOUND val fals. S'entra a l'IF i salta l'excepció, aturant l'execució del procediment. No s'arriba a fer l'insert i per tant la taula segueix estant buida.

2.2 Suposeu que cadascuna de les regles que s'indiquen a continuació es vol implementar mitjançant triggers. Indiqueu i justifiqueu, per a cada regla, quins triggers caldria definir, tenint en compte els criteris de qualitat de l'assignatura. Per cada trigger cal indicar:

- Esdeveniment que l'activa (cal indicar esdeveniment, taula i atributs rellevants)
- *Before* o *After* (justifiqueu la resposta)
- *Row* o *Statement* (justifiqueu la resposta)

a) REGLA1: Hi ha d'haver una fila a la taula `logs(tipusEsdeveniment, usuari, instant)` per cada vegada que un usuari de la base de dades faci insercions o modificacions de comandes. Es vol mantenir amb triggers el contingut de la taula `logs`.

b) REGLA2: L'`import` d'una comanda ha de ser igual a la suma dels resultats de multiplicar el `preu` de cada producte comprat en la comanda per la `quantitat` d'aquest producte que s'ha comprat. Es vol mantenir correcte el valor de l'atribut `import` quan quedi afectat per esdeveniments sobre la taula `productes`.

c) REGLA3: No pot existir cap producte a la base dades amb un `preu` superior al `preuMaxim` del tipus del producte. Es vol generar una excepció en cas que no es compleixi aquesta regla.

Regla 1

esdeveniments:
- INSERT comandes
- UPDATE comandes

before/after: AFTER. Per no fer feina innecessària en cas que es violi alguna RI de la BD
row/statement: STATEMENT. només ens importa l'esdeveniment i no les files

Regla 2

esdeveniments:
- UPDATE of preu on productes

before/after: AFTER. Per no fer feina innecessària en cas que es violi alguna RI de la BD
row/statement: ROW. Es pot fer statement però implicaria recalcular tots els imports de totes les comandes de la base de dades -> millor incremental

Regla 3

esdeveniment:

- UPDATE of idTipus,preu on productes
- UPDATE preuMaxim on tipusProductes
- INSERT on productes

before/after: BEFORE. Ja que és millor comprovar el més aviat possible el que es compleixen les restriccions d'integritat.

row/statement: ROW. Es pot fer statement però implicaria verificar que es compleix la restricció per tots els productes de la base de dades -> millor incremental

2.3 Implementeu un trigger que mantingui correcte el valor de l'atribut `import` de la taula `comandes` quan s'executi una sentència INSERT sobre la taula `productesComanda`. Tingueu en compte la REGLA2 de l'apartat anterior que defineix com es calcula l'import d'una comanda.

Podeu fer servir la plantilla de triggers de PostgreSQL següent:

```
CREATE TRIGGER nomTrigger
[BEFORE/AFTER] [UPDATE (of column)/DELETE/INSERT]ON taula
[FOR EACH ROW/FOR EACH STATEMENT] execute procedure nomp();

CREATE FUNCTION nomp() RETURNS trigger AS $$
    DECLARE
        ...
    BEGIN
        ...
    END;
$$ LANGUAGE plpgsql;
```

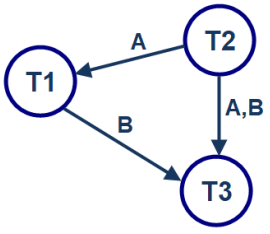
```
create function actualitza_import() returns trigger as $$
     declare
          preu integer;
     begin
          select p.preu into preu from productes p
          where nomProducte = new.nomProducte;

          update comandes
          set import = (import + (preu * new.quantitat))
          where numComanda = new.numComanda;

          return null;
        end;
$$ language plpgsql;

create trigger mante_import after insert on productesComanda
for each row execute procedure actualitza_import();
```

3. (2 punt) Donat el graf de precedències següent:



3.1 Digueu per cadascuna de les interferències que es produeixen: entre quines transaccions es produeixen, i els grànuls implicats.

No es produeix cap interferència donat que no hi ha cap cicle.

3.2 Doneu un horari que inclogui les mínimes operacions (R, RU, W, COMMIT) possibles i que es correspongui amb el graf de precedències.

Una possible solució és:

	T1	T2	T3
10		RU(A)	
20		W(A)	
30	R(A)		
40			R(A)
50	R(B)		
60			RU(B)
70		R(B)	
80			W(B)
90		commit	
100	commit		
110			commit

3.3 Digueu si l'horari que heu donat en l'apartat anterior és o no recuperable. Justifiqueu la resposta.

L'horari donat en la solució anterior és recuperable. Veure transparència.

3.4 Modifiqueu l'horari que heu donat, afegint una única operació (R,RU,W), per tal que s'afegeixi una única interferència de lectura no repetible entre T2 i T3. Si no és possible, justifiqueu la resposta.

Una possible solució és:

	T1	T2	T3
			R(A)
10		RU(A)	
20		W(A)	
30	R(A)		
40			R(A)
50	R(B)		
60			RU(B)
70		R(B)	
80			W(B)
90		commit	
100	commit		
110			commit

4. (2 punt) Donat un SGBD sense cap mecanisme de control de concurrència, suposem que es produeix l'horari següent:

Temps	T1	T2	T3
10		RU(B)	
20		W(B)	
40	RU(A)		
50	W(A)		
60	R(D)		
70	R(B)		
80			R(E)
100			RU(A)
110			W(A)
120			RU(F)
130		R(F)	
140		RU(D)	
150		W(D)	
160			W(F)
170	RU(F)		
180	W(F)		
190		COMMIT	
200			COMMIT
210	COMMIT		

4.1 Suposeu que s'incorpora un mecanisme per al control de la concurrència basat en reserves S, X, on les transaccions T1 i T3 treballa amb SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED, i T2 amb SET TRANSACTION ISOLATION LEVEL READ COMMITTED. Doneu l'horari aplicant reserves que ha d'incloure, a més de les operacions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves (LOCK, UNLOCK), i l'ordre d'execució de totes aquestes operacions. Quan més d'una transacció esperi per un mateix gràNul, considereu que la política de la cua és FIFO (First In, First Out). En cas que cap acció de l'horari pugui executar-se en un moment donat, no continueu i justifiqueu el motiu.

T1	T2	T3
	LOCK(B,X)	
	RU(B)	
	W(B)	
LOCK(A,X)		
RU(A)		
W(A)		
R(D)		
R(B)		
		R(E)
		LOCK(A,X)
	L(F,S)	
	R(F)	
	U(F)	
	LOCK(D,X)	
	RU(D)	
	W(D)	
LOCK(F,X)		
RU(F)		
W(F)		
	COMMIT+U(B,D)	
COMMIT+U(A,F)		
		RU(A)
		W(A)
		LOCK(F,X)
		RU(F)
		W(F)
		COMMIT+U(A,F)

4.2 Suposeu que s'incorpora un mecanisme per al control de la concurrència basat en reserves S, X, on les transaccions T1, T2 i T3 treballen amb SET TRANSACTION ISOLATION LEVEL SERIALIZABLE. Doneu l'horari aplicant reserves que ha d'incloure, a més de les operacions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves (LOCK, UNLOCK), i l'ordre d'execució de totes aquestes operacions. Quan més d'una transacció esperi per un mateix gränul, considereu que la política de la cua és FIFO (First In, First Out). En cas que cap acció de l'horari pugui executar-se en un moment donat, no continueu i justifiqueu el motiu.

T1	T2	T3
	LOCK(B,X)	
	RU(B)	
	W(B)	
LOCK(A,X)		
RU(A)		
W(A)		
LOCK(D,S)		
R(D)		
LOCK(B,S)		
		LOCK(E,S)
		R(E)
		LOCK(A,X)
	LOCK(F,S)	
	R(F)	
	LOCK(D,X)	

5. (2.5 punts) Considereu la taula T(a,b). La taula T té 40000 tuples i està emmagatzemada en un fitxer on a cada pàgina hi caben en promig 10 files.

5.1 En la taula no hi ha cap índex definit. Doneu el cost de la consulta següent i mostreu el detall de com es calcula.

```
SELECT * FROM T where a = valor
```

5.2 Es defineix ara un índex B+ no agrupat per l'atribut `a` de T. L'índex té d=50 i els nodes estan plens al 70% d'ocupació. Doneu el cost de la consulta de l'apartat anterior i mostreu el detall de com es calcula.

5.3 Es defineix ara un índex B+ no agrupat per l'atribut `b` de T. L'índex té d=50 i els nodes estan plens al 70% d'ocupació. Doneu el cost de la consulta següent i mostreu el detall de com es calcula.

```
SELECT SUM(b) FROM T
```

5.4 Suposeu ara que només hi ha un índex B+ agrupat per l'atribut `a` de T. L'índex té d=50 i els nodes estan plens al 70% d'ocupació. No es defineix cap índex sobre l'atribut `b`. Suposeu que els valors de l'atribut `a` estan distribuïts uniformement entre 1 i 40000. Doneu el cost de la consulta següent i mostreu el detall de com es calcula.

```
SELECT * FROM T WHERE a>=33000 AND b<=1000
```

Solució

5.1) Com que no hi ha cap índex definit a T i el fitxer no està ordenat, caldrà fer un recorregut seqüencial del fitxer de dades.

Card(T)=40000

f=factor de bloqueig del fitxer de dades=10

Cost = $\lceil Card(T) / f \rceil = \lceil 40000 / 10 \rceil$ = 4000 accessos a pàgines de dades.

5.2) Com que hi ha un índex definit per a l'atribut a, es baixarà per l'arbre buscant el valor (a=valor), i després un cop localitzat s'accedirà a la única fila (clau primària) que pot tenir aquest valor.

Al tenir una d=50, i ocupació del 70%, el número de valors per node és de $\lceil 2*50*0,7 \rceil$, és a dir, 70 valors per node, i 71 apuntadors en nodes interns.

Al baixar per l'arbre s'accedirà a tantes pàgines com nivells (h) té l'arbre.

$h=\lceil \log_{71} 40000 \rceil = 3$

El cost de localitzar la fila on es compleix a=valor, tenint en compte que "a" és clau primària, és:

Cost = h + 1 = 4

5.3) Caldrà accedir a totes les fulles de l'arbre per obtenir els valors de b i anar-los sumant. No és necessari accedir a les pàgines de dades, perquè els valors de b estan a l'índex. Per tant, el cost de resoldre la consulta és:

h és la mateixa que per l'índex sobre l'atribut a, ja que la d i la ocupació són les mateixes.

h = 3

F = nombre de fulles que cal llegir.

F = $\lceil 40000/70 \rceil$= 572

Cost = h + F - 1 = 3 +572 -1 =3 + 572 -1 =575, és resta 1 perquè hi ha una fulla que s'ha comptat dues vegades.

5.4) Com que només tenim un índex agrupat per l'atribut "a". Cal baixar per l'índex buscant el valor 33000. Tenint en compte que l'índex és agrupat, després es va a les pàgines de dades per buscar les files on es compleixi a>=33000. En accedir a les files dins de les pàgines de dades es podrà saber per cada fila si es compleix la condició b<=1000.

h és la mateixa que per l'índex sobre l'atribut a, ja que la d i la ocupació són les mateixes.

h = 3

D = nombre de pàgines de dades que cal llegir

D = $\lceil 7001 / 10 \rceil$ = 701

Cost = h + D = 3 + 701 = 704

Temps: 2,5 hores

Notes 20 juny - Revisió d'examen: 21 juny (presencial)

Cada pregunta s'ha de lliurar en un full separat

1. (1 punts) Donada la base de dades següent:

```
create table professors
(dni char(50),
nomProf char(50) unique,
sou integer,
primary key (dni));

create table despatxos
(modul char(5),
numero char(5),
superficie integer not null check(superficie <=25),
primary key (modul,numero));

create table assignacions
(dni char(50),
modul char(5),
numero char(5),
instantInici integer,
instantFi integer,
primary key (dni, modul, numero, instantInici),
foreign key (dni) references professors,
foreign key (modul,numero) references despatxos,
check (instantInici < instantFi));
/*instantFI té valor nul quan una assignació és vigent */
```

Considereu el següent fragment de codi Java/JDBC. Implementeu el cos del programa per tal que per cada professor de la base de dades que tingui alguna assignació vigent al despatx número 124 del mòdul 'Omega', es decrementi el sou del professor en 20 euros. Cal que el programa tregui un missatge d'error si no hi ha cap professor que compleixi la condició indicada.

```
try {
 /* Tenim ja una connexió c establerta amb la base de dades */

 //codi a implementar

 c.commit();
 }
catch (ClassNotFoundException ce) {
        System.out.println ("Error al carregar el driver");}
catch (SQLException se) {
        System.out.println ("Excepcio: "+ se.getMessage());}
```

Recordeu que podeu usar els mètodes/codis d'excepció de JDBC estudiats a classe:

Statements

- Statement createStatement();
- ResultSet executeQuery(String sql);
- int executeUpdate(String sql);

PreparedStatement

- `PreparedStatement prepareStatement(String sql);`
- `ResultSet executeQuery();`
- `int executeUpdate();`
- `void setXXX(int posicioParametre, XXX valor);`

ResultSet

- `boolean next();`
- `XXX getXXX(String nomColumna)`

SQLException

- `// 23502 -not_null_violation`
- `// 23503 -foreign_key_violation`
- `// 23505 -unique_violation`
- `// 23514 -check_violation`
- `String getSQLState();`

Solució

```
try {
 /* Tenim ja una connexió c establerta amb la base de dades */
  Statement stDel = c.createStatement();
  int numDcr = stDel.executeUpdate("update professors p "+
                 "set sou = sou - 20 "+
                 "where exists (select * from assignacions a "+
                               "where a.dni = p.dni and a.modul='Omega' "+
                               "and a.numero='124' "+
                                "and a.instantFi is null)");
  if (numDcr==0)
     {System.out.println("Error");}
     else {System.out.println("S'han decrementat ="+numD);}

 c.commit();}
 catch (ClassNotFoundException ce)
        {System.out.println     ("Error     al     carregar     el     driver");}
 catch (SQLException se)
        {System.out.println ("Excepcio: "+ se.getMessage());}
```

2. **(3.5 punts)** Donades les transaccions següents (per cada transacció s'indica les operacions que vol fer i l'ordre en que la transacció realitza aquestes operacions):

T1: RU(A), W(A), RU(A), W(A), Commit

T2: R(A), RU(B), W(B), Commit

T3: R(B), R(C), R(B), Commit

Es demana:

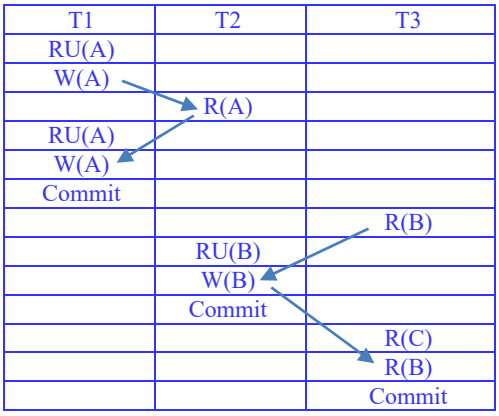
2.1 Proposeu un horari que inclogui les tres transaccions i que tingui dues interferències. Cal indicar el nom de les interferències, els grànuls implicats i les transaccions implicades.

2.2 Proposeu un horari, que inclogui encavalcaments entre les tres transaccions, i que tingui els dos horaris serials equivalents següents: T2;T1;T3 i T2; T3;T1.

2.3 Considereu ara només T1 i T2 i suposeu que tenim control de concurrència basat en reserves. Per cadascun dels quatre nivells d'aïllament, proposeu un horari amb aquestes dues transaccions de tal manera que es produeixi una interferència. En cas que no sigui possible produir la interferència, expliqueu breument perquè. L'horari ha d'incloure, a més de les peticions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves i l'ordre d'execució de totes aquestes peticions. També ha d'incloure, en cas que es produeixin, les esperes de les transaccions.

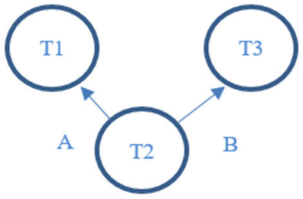
2.4 Considereu ara només T2 i suposeu que tenim control de concurrència basat en reserves. Per al nivell d'aïllament serialitzable proposeu un horari format per T2 i una altra transacció de tal manera que es produeixi una abraçada mortal. L'horari ha d'incloure, a més de les peticions que executen les transaccions (R, RU, W, COMMIT), les operacions de petició i alliberament de reserves i l'ordre d'execució de totes aquestes peticions. També ha d'incloure, en cas que es produeixin, les esperes de les transaccions. Doneu també el graf d'espera just després de que es produeixi l'abraçada mortal.

2.1.



Es produeix una lectura no confirmada entre T1 i T2 pel gränul A i una lectura no repetible entre T2 i T3 pel gränul B.

2.2. Per tenir els dos horaris serials equivalents demanats cal tenir el següent graf de precedències.



Per tant, cal generar un horari on les operacions no commutables entre T2 i T1 segueixin l'ordre següent: primer s'executen les de T2 i després les de T1. Entre T2 i T3 s'ha de produir un cas similar. Evidentment, l'horari ha d'estar lliure d'interferències.

T1	T2	T3
	R(A)	
RU(A)		
W(A)		
RU(A)		
W(A)		
Commit		
	RU(B)	
	W(B)	
		R(B)
		R(C)
	Commit	
		R(B)
		Commit

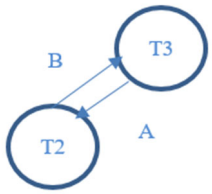
2.3. En el nivell Read Uncommitted no es fan reserves per lectura (modalitat S) i es fan reserves en modalitat X fins a l'acabament de la transacció. Per generar la interferència de lectura no confirmada hem d'assegurar-nos que la lectura del gränul A que fa T2 es faci entre les dues escriptures que fa T1.

T1	T2
Lock (A,X)	
RU(A)	
W(A)	
	R(A)
RU(A)	
W(A)	
Commit (+ unlock (A))	
	Lock (B,X)
	RU(B)
	W(B)
	Commit (+ unlock (B))

Aplicant les reserves del nivell Read Committed s'evita la lectura no confirmada. Per tan, no és possible produir un horari amb aquesta interferència.
Amb la resta de nivells d'aïllament també s'evita aquesta interferència. Per tant, no és possible generar un horari amb aquesta interferència.

2.4. A l'horari següent es pot veure que a partir de l'instant t=8 les dues transaccions estan suspeses sense possibilitat de continuar. T2 està esperant per reservar B (aquest gränul el té reservat T3) i T3 està esperant per reservar A (aquest gränul el té reservat T2).

Temps	T2	T3
1	Lock(A,S)	
2	R(A)	
3		
4		Lock (B,S)
5		R(B)
6	Lock(B,X)	
7		Lock (A,X)
8		



Graf d'espera en temps=8:

3. **(2 punts)** Considereu la base de dades següent.

```
competicions(idCompetició, nomCompetició)
equips(nomEquip, ciutat)
atletes(idAtleta, nomAtleta, dataNaixement, nomEquip)
    {nomEquip} referencia equips
arribadaAtletes(idCompetició, idAtleta, temps)
    {idCompetició} referencia competicions
    {idAtleta} referencia atletes
    -- el valor de l'atribut temps correspon al temps que
    ha trigat l'atleta a arribar a la meta en la competició
arribadaEquips(idCompetició, nomEquip, numAtletes)
    {idCompetició} referencia competicions
    {nomEquip} referencia equips
```

3.1 Es vol mantenir amb disparadors el contingut de la taula *arribadaEquips*.

- Hi ha d'haver una fila en aquesta taula per una competició i equip a partir de que es registra l'arribada a la meta del primer atleta de l'equip que participa en la competició.
- L'atribut *numAtletes* de la taula *arribadaEquips* ha de tenir per valor el número d'atletes d'un equip pels que s'ha registrat la seva arribada a la meta per una determinada competició.

Implementeu el(s) disparador(s) necessari(s) per al cas de l'esdeveniment d'inserció de files a la taula *arribadaAtletes*. Podeu fer servir la plantilla de triggers de PostgreSQL inclosa.

3.2 Suposeu ara que a la base de dades, en lloc d'haver-hi la taula *arribadaEquips*, hi ha la taula

```
ranquing(idCompetició, idAtleta, posicio, temps)
    {idCompetició} referencia competicions
    {idAtleta} referencia atletes
```

Es vol mantenir amb disparadors el contingut de la taula *ranquing*.

- Hi ha d'haver una fila en aquesta taula per cada atleta pel que s'ha registrat la seva arridada a la meta en una competició.
- El valor de l'atribut *temps* correspon al temps que ha trigat l'atleta a arribar a la meta en la competició.
- El valor de l'atribut *posicio* correspon al lloc on l'atleta ha quedat en la competició. La posició es calcula en funció del temps que han trigat els atletes a arribar a la meta. Dos atletes NO poden trigar exactament el mateix temps en arribar a la meta.

Implementeu el(s) disparador(s) necessari(s) per al cas de l'esdeveniment d'inserció de files a la taula *arribadaAtletes*. Aquestes insercions poden NO fer-se en l'ordre d'arribada a la meta dels atletes. Podeu fer servir la plantilla de triggers de PostgreSQL inclosa.

```
CREATE TRIGGER nomTrigger
[BEFORE/AFTER] [UPDATE (of column)/DELETE/INSERT] ON taula
[FOR EACH ROW/FOR EACH STATEMENT] execute procedure nomp();

CREATE FUNCTION nomp() RETURNS trigger AS $$
    DECLARE
        ...
    BEGIN
        ...
    END;
$$ LANGUAGE plpgsql;
```

2.1

```
CREATE or replace FUNCTION fn_novaArribadaAtleta_21() RETURNS trigger AS $$
   DECLARE
     nomEquipAtleta equips.nomEquip%type;
   begin
        -- INSERTAR O ACTUALITZAR LA TAULA D'arribadaEquips

        -- Cal recuperar el nom de l'equip de l'atleta
        select nomEquip into nomEquipAtleta
        from atletes
        where idatleta = new.idatleta;

        -- Si tenim valor fem UPDATE incrementant en 1 el numAtletes,
        -- sinó fem INSERT amb 1 atleta
        update arribadaEquips set numAtletes = numAtletes+1
        where idCompeticio = new.idCompeticio
              and nomEquip = nomEquipAtleta;

        if not found then
            insert into arribadaEquips
               values (new.idCompeticio, nomEquipAtleta, 1);
        end if;
       return null;
   END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_novaArribadaAtleta_21
AFTER INSERT ON arribadaatletes
      FOR EACH ROW execute procedure fn_novaArribadaAtleta_21();
```

2.2

```
CREATE or replace FUNCTION fn_novaArribadaAtleta_22() RETURNS trigger AS $$
   DECLARE
     maxpos ranquing.posicio%type;

   begin
        -- INSERTAR O ACTUALITZAR A LA TAULA DE ranquing
        select max(r.posicio) into maxpos
        from ranquing r
        where r.idcompeticio = new.idcompeticio and r.temps <= new.temps;

        -- no hi ha temps inferior o igual, la posició ha de ser 1
        if maxpos is null then
          insert into ranquing
           values (new.idCompeticio, new.idAtleta, 1, new.temps);
        else
          --temps màxim inferior, la posició ha de ser màxima +1
          insert into ranquing
            values (new.idCompeticio, new.idAtleta, maxPos+1,new.temps);
        end if;

        -- desplaçem el ranquing dels arribats en temps superiors
        update ranquing set posicio = posicio+1
        where idcompeticio = new.idCompeticio and temps > new.temps;

        return null;
   END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_novaArribadaAtleta_22
AFTER INSERT ON arribadaatletes
      FOR EACH ROW execute procedure fn_novaArribadaAtleta_22();
```

4. (3.5 punts)

4.1 Suposeu les dues situacions següents per a la base de dades de l'exercici 3:

- Les taules *equips* i *atletes* estan cadascuna en un espai virtual de taula.
- Les taules *equips* i *atletes* estan en un espai virtual d'agrupació.

a) Expliqueu quines diferències hi ha a nivell físic entre les dues situacions anteriors.

b) Doneu un exemple de consulta SQL que es vegi afavorida per cadascuna de les situacions anteriors.

4.2 Suposeu ara que la taula *atletes* està en un espai virtual de taula i té esquema:

atletes(idAtleta, nomAtleta, dataNaixement, nomEquip)

A la taula hi ha 120.000 atletes, i les pàgines de dades tenen un factor de bloqueig de 20. Hi ha 1000 atletes amb una data de naixement anterior a l'any 2000. Sobre la taula hi ha definits:

- un índex arbre B+ agrupat per *idAtleta* d'ordre d=255 i ple al 75%.
- un índex arbre B+ per *dataNaixement*. Es tracta d'un índex no agrupat amb ordre d=146, i ple al 80%.

a) Expliqueu com el SGBD resoldrà la consulta següent, doneu el cost en número de pàgines accedides, i mostreu el detall de com es calcula aquest cost.

SELECT * FROM atletes WHERE idAtleta = 34567 OR nomAtleta = 'Joan Pi'

b) Expliqueu com el SGBD resoldrà la consulta següent, doneu el cost en número de pàgines accedides, i mostreu el detall de com es calcula aquest cost.

SELECT * FROM atletes WHERE dataNaixement <= '31-12-1999'

4.1.a) En el primer cas, cada taula estarà emmagatzemada en un fitxer, les pàgines de cada fitxer contindran files únicament de la taula corresponent.

En el segon cas, les dues taules estaran emmagatzemades en el mateix fitxer, les pàgines de la taula estaran combinades les files de les dues taules tenint en compte el nom de l'equip (és a dir, hi haurà una fila corresponent a un equip, seguida de les files corresponents als atletes de l'equip, i a continuació un altre equip,...).

4.1.b) En el primer cas,

SELECT * FROM equips

En el segon cas,

SELECT * FROM equips natural inner join atletes WHERE nomEquip='BCN'

4.2.a) En aquest cas l'índex definit sobre idAtleta no ajuda en res a la resolució de la consulta, i per aquest motiu el SGBD farà una accés seqüencial a les pàgines de dades

cost = 120.000 files / 20 files per pàgina = 6.000 pàgines

4.2.b) En aquest cas s'usarà l'índex per dataNaixement, baixant per l'arbre B+, recorrent les fulles corresponents a valors on dataNaixement <= '31-12-1999', i per cada valor s'usarà el RID per buscar la fila corresponent al valor.

El cost de resoldre la consulta usant el índex seria:

n = ⌈2*d*ocupació⌉ = ⌈2*146*0,8⌉ = ⌈ 233,6 ⌉ = 234 valors per node

ap = n+1 = 234+1 = 235 apuntadors per node intern

h = ⌈ log235 120.000 ⌉ = ⌈ 2,14 ⌉ = 3 nodes = 3 pàgines

F = ⌈ valors a llegir en els nodes fulla / 234 valors per node⌉ =

⌈ 1000/234 nodes fulla⌉ = ⌈ 4,27 ⌉ = 5 pàgines

D = tantes pàgines com número de files que compleixen la condició

dataNaixement <= '31-12-1999' = 1000 pàgines

cost = 3+(5-1)+1000 = 1007 pàgines

Temps: 3 hores

Notes 25 de gener. Revisió d'examen: 26 de gener a la tarda (presencial)

Cada pregunta s'ha de lliurar en un full separat

1. (1 punts) Donada la base de dades següent:

```
artistes (nom, genere, anyDebut, anyRetirada)
albums(titol, anyAlbum, artista)
     {artista} referencia artistes
```

Considereu el codi Java/JDBC següent que esborra artistes que compleixen certes condicions:

```
try {
   PreparedStatement s, s2;
   ResultSet r, r2;
   int num = 0;
   // c és la connexió a la BD
   // Seleccionem els artistes que es van retirar abans del 1960
   s = c.prepareStatement("select distinct nom from artistes "+
                          "where anyRetirada < 1960");
   r = s.executeQuery();
   // Preparem un statement s2 per comprovar si un artista té àlbums
   s2 = c.prepareStatement("select * from albums where artista = ?");
   while (r.next()) {
      // Per cada artista resultant de s, executem s2 per comprovar si té àlbums
      s2.setString(1, r.getString("nom"));
      r2 = s2.executeQuery();
      if (r2.next()) {
          // Té àlbums, no l'esborrem, fem rollback i acabem
          System.out.println("Error: Algun dels artistes a esborrar té àlbums");
          c.rollback();
          num = 0;
          break; }
      else {
          // No té àlbums, executem s3 per a esborrar-lo i seguim
          Statement s3 = c.createStatement();
          num = s3.executeUpdate("delete from artistes where nom = '" +
                                 r.getString("nom") + "'"); }
   }
   if (num > 0) System.out.println("Esborrats els artistes antics i sense àlbums");
   // TANCAMENT DE LES ESTRUCTURES, COMMIT I DESCONNEXIÓ
}
catch (SQLException se) {
      System.out.println("Excepció: " + se.getSQLState() + ":" + se.getMessage());
}
```

Proposeu una implementació alternativa a aquest fragment de codi, de manera **que tingui el mateix comportament**, i **que compleixi els criteris de qualitat** de l'assignatura. **Justifiqueu** els canvis que heu fet.

Recordeu que podeu usar els mètodes/codis d'excepció de JDBC estudiats a classe i que estan inclosos a continuació:

Statement

- `Statement createStatement();`
- `ResultSet executeQuery(String sql);`
- `int executeUpdate(String sql);`

PreparedStatement

- `PreparedStatement prepareStatement(String sql);`
- `ResultSet executeQuery();`
- `int executeUpdate();`
- `void setXXX(int posicio, XXX valor);`

ResultSet

- `boolean next();`
- `XXX getXXX(String nomColumna)`

SQLException

- `// 23502 -not_null_violation`
- `// 23503 -foreign_key_violation`
- `// 23505 -unique_violation`
- `// 23514 -check_violation`
- `String getSQLState();`

SOLUCIÓ:

Una solució que compleix tots els criteris seria:

```
try {
      // c és la connexió a la BD
      Statement s = c.createStatement();
      // Intentem esborrar els artistes que es van retirar abans del 1960
      // Si algún d'ells té un àlbum a la BD, saltarà una excepció de clau forana
      int num = s.executeUpdate("delete from artistes where anyRetirada < 1960");
      if (num>0) System.out.println("Els artistes antics i sense àlbums han estat
esborrats ");
      // TANCAMENT DE LES ESTRUCTURES, COMMIT I DESCONNEXIÓ
}
catch (SQLException se) {
      if(se.getSQLState().equals("23503"))
            System.out.println("Error: Algun dels artistes a esborrar té àlbums");
         else {
            System.out.println("Excepció: " + se.getSQLState() + ":"
                                          + se.getMessage());
      }
}
```

El que s'ha fet és:

- Eliminar el primer select (PreparedStatement s) perquè és innecessari: es pot afegir com a condició del delete. A més, té un distinct innecessari, i s'hauria de fer amb un Statement.
- Eliminar el segon select (PreparedStatement s2) perquè és innecessari: es pot detectar l'error capturant una excepció de violació de clau forana produïda pel delete.
- Com que es pot fer tot amb una sentència de delete que s'executarà una sola vegada (ja no tenim cap bucle), hem utilitzat un Statement. A la solució inicial, l'opció correcta hauria estat un PreparedStatement (si el bucle fos necessari).

2.(3 punts) Suposeu una base de dades amb les taules i contingut següents.

```
articles (idArticle, nom,    marca,      stock)
            (42,    'ps5', 'sony',      1)
            (43,    'ps4', 'nintendo',  100)

cistelles (idClient, idArticle, quantitat)
             (31,    42,         null)
             (55,    42,         null)
```

Considereu l'horari que apareix a continuació.

T1	T2	T3
	select * from articles where marca = 'sony';	
		select * from articles where idArticle = 42;
	update cistelles set quantitat = 1 where idClient = 31 and idArticle = 42;	
		update cistelles set quantitat = 1 where idClient = 55 and idArticle = 42;
	update articles set stock = stock - 1 where idArticle = 42;	
		select * from articles where idArticle = 42;
		commit
update articles set marca = 'sony';		
commit		
	abort	

2.1 Tenint en compte que el grànul és la fila (per identificar un grànul, utilitzeu la clau primària de la fila). Es demana:

a) Doneu l'horari en termes d'operacions de baix nivell sobre grànuls (no poseu operacions sobre el grànul IC, a no ser que hagueu vist que hi ha alguna interferència Fantasma a l'horari donat).

b) En cas que hi hagi interferències a l'horari, indiqueu, per cada interferència: nom de la interferència, transaccions implicades, grànuls afectats, i el nivell mínim d'aïllament per evitar-la. En cas que no n'hi hagi, indiqueu els horaris serials equivalents.

2.2 Suposeu ara que T1 treballa amb nivell d'aïllament REPEATEABLE READ, i T2 i T3 treballen amb READ COMMITTED.

a) Doneu l'horari anterior aplicant els nivell d'aïllament indicats. L'horari ha d'incloure, a més de les operacions que executen les transaccions (R, RU, W, COMMIT, ABORT), les operacions de petició i alliberament de reserves (LOCK, UNLOCK), i l'ordre d'execució de totes aquestes operacions. Quan més d'una transacció espera per un mateix grànul, considereu que la política de la cua és FIFO (First In, First Out).

b) Doneu el graf d'espera just abans de l'abort de T2.

2.3 En cas que hi hagi interferències a l'horari donat a l'apartat 2.2, cal que indiqueu per cada interferència: nom de la interferència, transaccions implicades, grànuls afectats. En cas que no n'hi hagi, indiqueu els horaris serials equivalents.

SOLUCIÓ 2.1

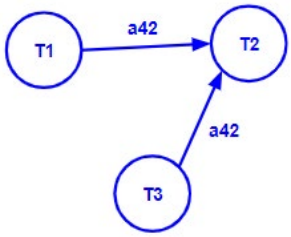
T1	T2	T3
	R(a42)	
		R(a42)
	RU(c3142)	
	W(c3142)	
		RU(c5542)
		W(c5542)
	RU(a42)	
	W(a42)	
		R(a42)
		commit
RU(a42)		
W(a42)		
RU(a43)		
W(a43)		
commit		
	abort	

Sí, existeix tres interferències:

- Entre T1 i T2 lectura no confirmada sobre a42. Nivell d'aïllament: Read Committed.
- Entre T2 i T3 lectura no repetible sobre a42. Nivell d'aïllament: Repeteable Read.
- Entre T2 i T3 lectura no confirmada sobre a42. Nivell d'aïllament: Read Committed.

SOLUCIÓ 2.2

T1 Repeateable Read	T2 Read Committed	T3 Read Committed
	L(a42, S)	
	R(a42)	
	U(a42)	
		L(a42, S)
		R(a42)
		U(a42)
	L(c3142, X)	
	RU(c3142)	
	W(c3142)	
		L(c5542, X)
		RU(c5542)
		W(c5542)
	L(a42, X)	
	RU(a42)	
	W(a42)	
		L(a42, S)
LOCK(a42, X)		
	ABORT+U(c3142, a42)	
		R(a42)
		U(a42)
		COMMIT+U(c5542, a42)
RU(a42)		
W(a42)		
L(a43, X)		
RU(a43)		
W(a43)		
COMMIT+U(a42, a43)		



SOLUCIÓ 2.3

No hi ha cap interferència.

- Les de lectura no confirmada s'han evitat pels nivells d'aïllament usats en les transaccions.
- La de lectura no repetible no s'evita amb Read Committed, però s'evita circumstancialment degut a l'Abort.

Els horaris serials equivalents són: T3;T1

3.(3 punts) Considereu la taula `Empleats(id, nom, ciutat)`. Hi ha un índex arbre B+ agrupat per `id` i dos índexs arbre B+ no agrupats, un per `nom` i un per `ciutat`. La taula té 100.000 tuples, i els id dels empleats estan repartits uniformement entre 1 i 100.000. Hi ha 1.000 empleats de Barcelona, i d'aquests 200 tenen el id més gran que 20.000. Els índexs són tots d'ordre 50, estan ocupats en mitjana al 80%, i a cada pàgina de dades hi ha 5 tuples.

3.1 Donada la consulta següent, calculeu (mostreu i justifiqueu clarament els càlculs) el cost en nombre de pàgines (d'índex i de dades) que es llegiran si la consulta es resol:

a) usant l'índex per `id`
b) usant l'índex per `ciutat`
c) usant els 2 índexs alhora

```
SELECT *
FROM Empleats e
WHERE e.id > 20000 AND e.ciutat ='Barcelona'
```

3.2 Considereu ara la consulta següent. Quin hauria de ser el valor de X per tal de que la consulta resolta usant l'índex per `id` tingui un cost menor que resolent la consulta amb un recorregut seqüencial? Mostreu i justifiqueu clarament els càlculs que heu fet per obtenir el valor de X.

```
SELECT *
FROM Empleats e
WHERE e.id > X AND e.ciutat ='Barcelona'
```

SOLUCIÓ:

3.1)

a) $n = 2*d*\text{ocupació} = 2*50*0,8 = 80$ valors per node
$h = \lceil \log_{81} 100000 \rceil = 3$
$D = 80000/5 = 16000$

Cost total = 3 + 16000

b)

n = 2*d*ocupació = 2*50*0,8 = 80 valors per node
h = ⌈ $\log_{81}$ 100000 ⌉ = 3
F = ⌈ valors a llegir en els nodes fulla / 80⌉ = ⌈ 1000/80 ⌉ nodes fulla = 13

Cost total = 3 + 12+ 1000 = 1015 pàgines

c)

n = 2*d*ocupació = 2*50*0,8 = 80 valors per node
h = ⌈ $\log_{81}$ 100000 ⌉ = 3
F = id a llegir en els nodes fulla / 80 = 80000/80 nodes fulla = 1000

Cost índex per id = h + (F-1) = 3 + 999 = 1002
Cost índex per ciutat = h + (F-1) = 3 + 12 = 15 (aquesta F s'ha calculat a 3.1.b)

Cost total: 15 + 1002 +200 pàgines

3.2)

Cost Rec. Seq: 100000/5 = 20000 pàgines

Índex per id: 3 + ⌈ (100000- X)/5 ⌉

$$3 + \lceil (100000 - X)/5 \rceil < 20000$$

Qualsevol X més gran que 20 farà que la consulta amb índex tingui un cost menor (es considera també correcta la solució X més gran que 15 a la que s'arriba si no es considera l'arrodoniment).

4.(3 punts) Considereu una base de dades amb les taules següents. Per simplificar, suposeu que les dates estan implementades com a `integer`.

```
clients (dni, nom, telèfon, mail)
tipusHabitacions (idHotel, idTipus, descripció)
        -- hi ha una fila a la taula per cada tipus d'habitació
        que hi ha en un hotel
reserves (dni, dataIni, dataFi, idHotel, idTipus)
        {idHotel, idTipus} referencia tipusHabitacions
        {dni} referencia clients
        {dataFi} is not null
        -- les reserves es fan d'un tipus d'habitació d'un hotel
        des d'una dataIni fins a la dataFi. Per exemple hi pot
        haver una reserva d'una habitació doble a l'hotel Ritz
        entre les dates 4 i 8 feta pel client amb dni 333.
        -- per simplificar suposarem que les dates estan
        implementades com un número enter. Per exemple, una
```

```
        reserva entre les dates 4 i 8, ocuparà una habitació en
        les dates 4,5,6,7 i 8.
ocupacioHotels (idHotel, idTipus, data, reservades)
        {idHotel, idTipus} referencia tipusHabitacions
        check (reservades>=1)
        -- l'atribut reservades indica quantes habitacions del
        tipus idTipus de l'hotel idHotel estan reservades en una
        data concreta.
        -- hi ha una fila a la taula ocupacioHotels quan hi ha
        una o més reserves d'habitacions d'un tipus en un hotel
        i una data. Per exemple, hi haurà una fila si en la data
        7 hi ha 5 habitacions dobles reservades a l'hotel Ritz.
        Però no hi haurà cap fila corresponent a l'hotel Ritz,
        habitació individual, i data 10, si per aquest hotel,
        tipus d'habitació i data no hi ha cap habitació
        reservada.
```

Implementeu amb disparadors el comportament següent que **s'ha de fer complir quan s'insereix una reserva a la taula reserves**:

- No hi pot haver dues reserves solapades en dates fetes per un mateix client. En cas que no es compleixi caldrà generar una excepció. Dues reserves estan solapades si en els intervals entre la seva data d'inici i data fi, s'inclou una mateixa data (per exemple, la reserva ('dni1', 10, 20, 'hotel1', 'individual') està solapada amb la reserva ('dni1', 18, 25, 'hotel2', 'individual') concretament en les dates 18, 19 i 20).
- Hi ha d'haver una fila a la taula ocupacióHotels per un hotel, tipus d'habitació i data a partir de que hi hagi hi ha una o més reserves fetes d'una habitació del tipus, a l'hotel, en la data.
- El valor de l'atribut derivat *reservades*, de la taula *ocupacioHotels*, ha de correspondre a la quantitat d'habitacions del tipus que hi ha reservades en l'hotel en aquella data.

Podeu fer servir la plantilla de triggers següent.

```
CREATE OR REPLACE FUNCTION actualitza_lloger() RETURNS TRIGGER AS $$
DECLARE
  ... ...
BEGIN
  ... ...
EXCEPTION
  WHEN { raise_exception | unique_violation | foreign_key_violation | check_violation | others }
  THEN
   SELECT texte INTO missatge FROM missatgesExcepcions WHERE num=?;
```

```
    RAISE EXCEPTION '%', missatge;
   ...
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER disparador {BEFORE | AFTER} {esdeveniment[OR ...]} ON taula
FOR [EACH] {ROW | STATEMENT} EXECUTE PROCEDURE actualitza_lloger();
```

SOLUCIÓ:

```
CREATE OR REPLACE FUNCTION novaReservaBef() RETURNS TRIGGER AS $$
BEGIN
    if (exists(select * from reserves
               where new.dni = dni and
                    ((new.dataIni >=dataini and new.dataIni<=dataFi)
                  or (new.dataFi >=dataIni and new.dataFi<=dataFi)
                  or (new.dataIni <=dataIni and new.dataFi>=dataFi))))
      then raise exception 'reserva solapada';
     end if;
     return new;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER disparadorBef BEFORE INSERT ON reserves
FOR EACH ROW EXECUTE PROCEDURE novaReservaBef();

CREATE OR REPLACE FUNCTION novaReservaAft() RETURNS TRIGGER AS $$
DECLARE
    dataA integer;
BEGIN
    for dataA in new.dataIni..new.dataFi
      loop
            update ocupacioHotels
            set ocupades = ocupades + 1
            where idHotel = new.idHotel
                  and idTipus = new.idTipus
                  and data = dataA;
            if not found then
                  insert into ocupacioHotels
                  values (new.idHotel,new.idTipus,dataA,1);
            end if;
      end loop;
    return null;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER disparadorAft AFTER INSERT ON reserves
FOR EACH ROW EXECUTE PROCEDURE novaReservaAft();
```

Temps: 3 hores

Notes 27 de juny. Revisió d'examen: 29 de juny a la tarda (presencial)

Cada pregunta s'ha de lliurar en un full separat

1. (1 punts) Donada la base de dades següent:

```
create table professors
(dni char(50),
nomProf char(50) unique,
sou integer check (sou<=2000),
primary key (dni));

create table despatxos
(modul char(5),
numero char(5),
superficie integer not null check(superficie <=25),
primary key (modul,numero));

create table assignacions
(dni char(50),
modul char(5),
numero char(5),
instantInici integer,
instantFi integer,
primary key (dni, modul, numero, instantInici),
foreign key (dni) references professors,
foreign key (modul,numero) references despatxos,
check (instantInici < instantFi));
```

Considereu el següent fragment de codi Java/JDBC. Implementeu el cos del programa per tal que per cada professor de la base de dades que tingui assignacions a un despatx amb una superfície més gran que 20 que no sigui del mòdul Omega, s'augmenti el seu sou en 100. Cal que el programa tregui un missatge d'excepció si el sou d'algun professor passa a ser superior a 2000. En cas que no hi hagi cap excepció el programa indicarà el número de professors modificats i si no se n'ha modificat cap, es mostrarà un text que ho indiqui.

```
try {
 /* Tenim ja una connexió c establerta amb la base de dades */

 //codi a implementar

 c.commit();
 }
catch (ClassNotFoundException ce) {
        System.out.println ("Error al carregar el driver");}
catch (SQLException se) {
        System.out.println ("Excepcio: "+ se.getMessage());}
```

Recordeu que podeu usar els mètodes/codis d'excepció de JDBC estudiats a classe:

Statements

- `Statement createStatement();`
- `ResultSet executeQuery(String sql);`
- `int executeUpdate(String sql);`

PreparedStatement

- PreparedStatement prepareStatement(String sql);
- ResultSet executeQuery();
- int executeUpdate();
- void setXXX(int posicioParametre, XXX valor);

ResultSet

- boolean next();
- XXX getXXX(String nomColumna)

SQLException

- // 23502 -not_null_violation
- // 23503 -foreign_key_violation
- // 23505 -unique_violation
- // 23514 -check_violation
- String getSQLState();

SOLUCIÓ:

```
try {
    /* Tenim ja una connexió c establerta amb la base de dades */
    Statement st = c.createStatement();
    int numP = st.executeUpdate("UPDATE professors "+
      "SET sou = sou + 100 "+
      "WHERE exists (SELECT *"+
            "FROM assignacions a, despatxos d "+
            "WHERE a.dni = professors.dni "+
                 "AND a.modul = d.modul AND a.numero = d.numero "+
                 "AND d.superficie > 20 AND d.modul != 'Omega')");
    if (numP==0)
      {System.out.println("Error");}
    else
      {System.out.println("S'han incrementat "+numP)+" sous;}
    c.commit();
    }
catch (ClassNotFoundException ce) {
    System.out.println ("Error al carregar el driver");}
catch (SQLException se) {
   if(se.getSQLState().equals("23514"))
        System.out.println("La sou no pot ser més gran de 2000");
    else
        System.out.println ("Excepcio: "+ se.getMessage());} }
```

2. (3 punts)

2.1 Considereu la transacció següent: T1: RU(A);W(A);R(A);COMMIT;

a) Justifiqueu si pot existir un horari on intervingui només aquesta transacció i sigui NO serialitzable

b) Justifiqueu si pot existir un horari on intervingui només aquesta transacció i sigui NO recuperable

SOLUCIÓ

No poden existir. Veure definicions de serialitzable i recuperable en les transparències.

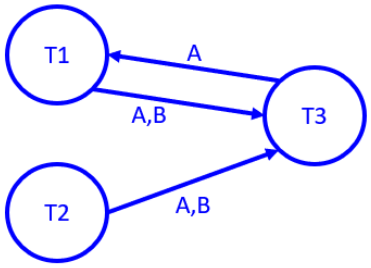
2.2 Considereu l'horari següent:

a) Indiqueu si és serialitzable, justificant breument la resposta a partir del graf de precedències corresponent a l'horari.

b) Indiqueu si es produeix alguna interferència. En cas afirmatiu, digueu quines són, sobre quins gränul i quin és el nivell mínim d'aïllament que podria evitar cadascuna d'elles. En cas negatiu, justifiqueu la resposta.

Temps	T1	T2	T3
10		R(E)	
20	RU(A)		
30		R(A)	
40			RU(A)
50	R(B)		
60			RU(B)
70	W(A)		
80			W(A)
90		R(B)	
100			W(B)
110		RU(C)	
120		W(C)	
130		COMMIT	
140	RU(A)		
150	W(A)		
160	COMMIT		
170			COMMIT

a) No és serialitzable atès que es produeixen cicles en el graf de precedències.



b) Es produeixen tres interferències:

- Entre T1 i T3 es produeix una interferència d'actualització perduda sobre el gränul A (op. 70). El nivell mínim d'aïllament que l'evitaria seria READ UNCOMITTED
- Entre T1 i T3 es produeix una interferència de lectura no confirmada sobre el gränul A (op. 140). El nivell mínim d'aïllament que l'evitaria seria READ COMITTED
- Entre T1 i T3 es produeix un interferència d'anàlisi inconsistent sobre els gränuls A i B (op. 50, 80, 100, 140). El nivell mínim d'aïllament que l'evitaria seria REPETEABLE READ.

2.3 Suposeu ara que sobre l'horari de l'apartat anterior, s'incorpora un mecanisme de control de concurrència basat en reserves S, X i que les transaccions T1 i T3 treballen a un nivell d'aillament REPEATABLE READ i T2 ho fa a un nivell d'aïllament READ COMMITED.

a) Doneu l'horari resultant

b) És serialitzable l'horari resultant? Justifiqueu la resposta.

Temps	T1	T2	T3
10		LOCK(E,S)	
20		R(E)	
30		UNLOCK(E)	
40	LOCK(A,X)		
50	RU(A)		
60		LOCK(A,S)	
70			LOCK(A,X)
80	LOCK(B,S)		
90	R(B)		
100	W(A)		
110	RU(A)		
120	W(A)		
130	COMMIT + U(A,B)		
140		R(A)	
150		UNLOCK(A)	
170			RU(A)
180			LOCK(B,X)
190			RU(B)
200			W(A)
210		LOCK(B,S)	
220			W(B)
230			COMMIT + U(A,B)
240		R(B)	
250		UNLOCK(B)	
260		L(C,X)	
270		RU(C)	
280		W(C)	
290		COMMIT + U(C)	

b) Hi ha un anàlisi inconsistent entre T2 i T3 pels grànuls A i B, l'horari no és serialitzable.

2.4 Suposeu l'horari resultant de l'apartat anterior. Considereu si és o no possible afegir una transacció T4 que provoqui una lectura no confirmada sobre el grànul A amb la transacció T1. Si es pot produir la interferència, indiqueu el nivell d'aïllament de la nova transacció T4 i doneu l'horari resultant. Si no és possible justifiqueu la resposta.

L'únic nivell d'aïllament possible per T4 seria el READ UNCOMMITTED. Com que el READ UNCOMMITED no fa LOCKS per les operacions de lectura, es pot provocar la interferència: fent un R(A) a T4 en qualsevol moment després de que T1 actualitzi el grànul A però abans que el torni a actualitzar per segona vegada; o bé fent un R(A) a T4 abans que T1 faci la primera actualització, i fent un abort de T4 desrprés de que T1 hagi fet el commit.

Temps	T1	T2	T3	T4
10		LOCK(E,S)		
20		R(E)		
30		UNLOCK(E)		
40	LOCK(A,X)			
50	RU(A)			
60		LOCK(A,S)		
70			LOCK(A,X)	
80	LOCK(B,S)			
90	R(B)			
100	W(A)			
105				R(A)
106				COMMIT
110	RU(A)			
120	W(A)			
130	COMMIT + UNLOCK(A,B)			
	La resta de files ja no canvien			

3. (3 punts) Considereu l'esquema de la base de dades següent:

```
CREATE TABLE lots
       (idLot INT primary key,
       preuLot REAL,
       quantsProductes INT,
       check (preuLot < 100 or quantsProductes > 3));

CREATE TABLE productes
       (idProd INT primary key,
       preuProd REAL NOT NULL,
       idLot INT NOT NULL references lots);

CREATE FUNCTION incrPreuPrBef() RETURNS trigger AS $$
 BEGIN
   if (TG_OP = 'INSERT' and NEW.preuprod<50) then
            NEW.preuprod := NEW.preuprod+5;
            RETURN NEW;
   end if;
   RETURN NULL;
 END; $$ LANGUAGE plpgsql;

 CREATE FUNCTION incrPreuPrAft() RETURNS trigger AS $$
 BEGIN
   if (TG_OP = 'INSERT') then
       update lots
       set preuLot = preuLot + NEW.preuProd,
            quantsProductes = quantsProductes + 1
       where idLot = NEW.idLot;
  else
       update lots
       set preuLot = preuLot + (NEW.preuProd-OLD.preuProd)
       where idLot = NEW.idLot;
   end if;
 RETURN NULL;
 END; $$ LANGUAGE plpgsql;

 CREATE TRIGGER incrPreuTrBef
 BEFORE INSERT OR UPDATE OF preuProd ON productes
 FOR EACH ROW EXECUTE procedure incrPreuPrBef();

 CREATE TRIGGER incrPreuTrAft
 AFTER INSERT OR UPDATE OF preuProd ON productes
 FOR EACH ROW EXECUTE procedure incrPreuPrAft();
```

3.1 Suposeu que s'executen una a una les sentències SQL següents (en cada apartat - a i b - partint de la base de dades buida). Indiqueu quin serà el resultat de les sentències ***select * from lots; select * from productes;*** en el cas de ser afegides i executades just després de cadascun dels inserts de cadascun dels apartats. Justifiqueu breument la resposta, explicant les accions que fa el SGBD a conseqüència de cada inserció.

a)

```
insert into lots values (1,0,0);
insert into productes values(1,30,1);
insert into productes values(2,50,1);
insert into productes values(3,40,1);
insert into productes values(4,35,1);
```

b)

```
begin transaction;
insert into lots values (1,0,0);
insert into productes values(1,30,1);
insert into productes values(2,50,1);
insert into productes values(3,40,1);
insert into productes values(4,35,1);
commit;
```

3.2 Doneu una implementació equivalent al trigger after anterior, amb la mateixa funcionalitat, i que sigui For Each Statement. Justifiqueu quina de les dues maneres d'implementar el trigger s'adequa més als criteris de qualitat de l'assignatura.

3.1.a

Sentència que s'executa	Lots	Productes	Accions del SGBD
insert into lots values (1,0,0);	(1,0,0)		Només s'executa el insert
insert into productes values(1,30,1);	(1,35,1)	(1,35,1)	El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu del lot al que pertany el producte.
insert into productes values(2,50,1);	(1,35,1)	(1,35,1)	El trigger before retorna null, i per tant no s'acaba fent la inserció del producte ni tampoc s'executa el trigger after amb la modificació del preu del lot
insert into productes values(3,40,1);	(1,80,2)	(1,35,1) (1,45,1)	El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu del lot al que pertany el producte.
insert into productes values(4,35,1);	(1,80,2)	(1,35,1) (1,45,1)	El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu En modificar el preu del lot es dóna error de c perquè la fila 1 de lots tindrà el preu més gra no tindrà més de 3 productes.

3.1.b

Sentència que s'executa	Lots	Productes	Accions del SGBD
begin transaction; insert into lots values (1,0,0);	(1,0,0)		S'inicia la transacció i només s'executa el insert de lot.
insert into productes values(1,30,1);	(1,35,1)	(1,35,1)	El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu al que pertany el producte.
insert into productes values(2,50,1);	(1,35,2)	(1,35,1)	El trigger before retorna null, i per tant no s'acaba fent la inserció del producte ni tampoc s'executa el trigger after amb la modificació del preu del lot
insert into productes values(3,40,1);	(1,80,2)	(1,35,1) (1,45,1)	El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu del lot al que pertany el producte
insert into productes values(3,40,1);			El trigger before canvia el preu de producte que s'insereix. S'executa l'insert del producte. El trigger after causa que es modifiqui el preu del lot, En modificar el preu del lot es dóna error de check perquè la fila 1 de lots tindrà el preu més gran que 100 i no tindrà més de 3 productes. Al ser dins d'una transacció es torna a l'estat abans de l'inici de la transacció.

3.2

```
CREATE FUNCTION incrPreuPrAft() RETURNS trigger AS $$
 BEGIN
       update lots
       set quantsProductes = select count(*)
                             from productes p
                             where p.idLot = lots.idLot,
            preuLot = select sum(p.preuProd)
                      from productes p
                      where p.idLot = lots.idLot;
       RETURN NULL;
 END; $$ LANGUAGE plpgsql;

 CREATE TRIGGER incrPreuTr
 AFTER INSERT OR UPDATE OF preuProd ON productes
 FOR EACH STATEMENT EXECUTE procedure incrPreuPrAft();
```

- Aquesta solució no és incremental, ja que ha de modificar el preu i la quantitat de productes **de tots els lots**, per tant és pitjor que la For Each Row que només modifica el preu dels lots que tenen productes que han canviat de preu.

4. (3 punts) Considereu la taula `R(a,b,c,d)`. L'atribut `a` és clau primària i sobre l'atribut `c` hi ha definida una restricció unique. La taula té 500.000 tuples. Hi ha un índex definit sobre l'atribut `a`, d'ordre 125, 80% ocupació en mitjana, i a cada pàgina de dades hi ha 5 tuples.

Mostreu clarament com heu fet els càlculs que es demanen als apartats següents.

4.1 Quina és la quantitat de pàgines necessàries per emmagatzemar l'índex i la taula?

Taula: 500000/5 = 100000 pàgines de dades

Índex: 2d* 0.8 = 200

fulles: 500000/200 = 2500

Nivell 1: 2500/201 = 13

Nivell 2: 13/201 = 1

2514 pagines d'índex

4.2 Suposeu que hi ha 4000 tuples que compleixen la condició de select següent. Quin seria el cost de resoldre la consulta si l'índex fos no agrupat? I si fos agrupat?

```
SELECT *
FROM R
WHERE a >= 800000
```

Cost no agrupat = h + (F-1) + D

h = $\lceil \log_{201} 500000 \rceil = 3$

F – 1= 4000/200 -1 = 19

Cost = 3 + 19+ 4000

Cost agrupat = h + D

D= 4000/5 = 800

Cost= 3 + 800

4.3 Considereu ara la següent expressió SQL, i calculeu el cost en els dos casos anteriors, és a dir, si l'índex fos agrupat o no agrupat. Justifiqueu la resposta.

```
Exists (SELECT a
        FROM R
        WHERE a >= 800000 )
```

No agrupat : h =3

Agrupat: h = 3

No és necessari accedir a la taula per resoldre la consulta, n'hi ha prou amb saber si hi ha algun valor que compleix la condició.

4.4 Considereu ara la següent expressió SQL:

```
Exists (SELECT a,c
        FROM R
        WHERE a >= 800000 and c = 10)
```

I suposeu que hi ha un índex agrupat per l'atribut `a`, un índex no agrupat per l'atribut `c`, que tots dos índexs tenen ordre d=125 i 80% d'ocupació.

Quin mètode usaríeu per resoldre la consulta? Justifiqueu la resposta mostrant que el mètode escollit és el de menor cost.

El mètode millor seria accedir per l'índex de c, recuperar la única tupla que compliria la condició (hi ha una restricció unique per aquest atribut) i accedir al fitxer de dades per comprovar la condició sobre l'atribut a.

- $Cost_c$: h +1

Els altres mètodes tindrien sempre un cost igual o superior. En el cas pitjor:

- índex per a = h + 19 + 4000
- intersecció RIDs = h + 19 + h + (1-1)