Eksamen 2020 inf3100

1– Some SQL

1.1 Triggers

Assume that we have a modern people´s registry

preg(personID, name, lastname)

and a modern marriage registry

mreg(mID, pID, prevLastname, newLastname, mDate)

where

• the mDate is a date/time attribute showing the marriage date and when the last name is changed (if it is changed at all), and

• the pID in mreg is the ID of the person for which the marriage and the change is registered, referring to the personID in preg.

A person can be married and re-married after a divorce many times, and the modern rules accept last name change for both male and female (so you don´t have to worry about the gender, which is not in the relations here).

Create a trigger that checks whether the new and the old last names match before an update to the preg, registers (logs) the old last name and the new last name for that specific marriage and person in the marriage register mreg and updates the people´s registry with the new last name.

Solution:

```
CREATE OR REPLACE FUNCTION log_last_name_changes ()
        RETURNS trigger AS $llnc$
        BEGIN
        IF NEW.lastname <> OLD.lastname THEN
                INSERT INTO mreg(pID, prevLastname, newLastname, mDate)
                VALUES(OLD.personID,OLD.lastname,NEW.lastname,now());

        END IF;
        RETURN NEW;
END;
$llnc$ language plpgsql;

CREATE TRIGGER last_name_changes
BEFORE UPDATE ON preg
```

1.2 Arrays

How can you remove the additional mreg in part 1.1 and create instead a preg that has a list of last names and their corresponding marriage or change dates for each new last name?

Write the new relation and explain briefly how it would function.

Svar:

```
CREATE TABLE preg(
        PersonID INT  NOT NULL,
        name text  NOT NULL,
        lastname text [ ] [ ]
        PRIMARY KEY (PersonID)
);
```

Ved å gjøre lastname til en 2-dimensjonal array kan vi bruke den til å lagre lastname og tilhørende forandrings dato eller giftemåls ID. Avhengig av om man velger forandring dato eller giftemål ID som sekunder attribut til lastname og om man sorterer det ascending eller descending kan man velge først eller nåværende etternavn personen med PersonID X hadde eller etternavnet vet et spesifikt ekteskap.

2 – Indices

This is a short question with a long answer but try to keep it as brief as possible and use your own words in the explanations:
List up the major categories and types of indices that we have seen and explain their respective advantages and disadvantages, as well as their uses.

Svar:
De størst kategoriene og typene indices er dense indices, sparse indices, primary indices, cluster indices, secondary indices, inverted indices, B trees, hash tables, multidimensional indices, tree structures, R trees, Hash-like structures og bitmap indices.

Dense indices kan bruke i alle tilfeller hvor man kan implementere indices. Fordelen med dense er at den har direct access til dataen og ved exist-queries trenger man kun sjekke indexen. Ulempene er at indeksen tar større plass og at vet oppdateringer eller forandringer så må de påvirkede indices også oppdateres.

Sparse indices kan brukes på sorterte elementer. Fordelene er at tar mindre plass på grunn av at den ikke indekserer til hvert element, kan være raskere en dense hvis indekseringen er bra og man må ikke alltid oppdatere indeksen hvis ikke først elementet i datablokken forandres. Ulempene er at for tilgang til elementene og ved exist-queries må man inn å lete i datablokken og som sagt for bruk så funker denne index typen dårlig på usorterte elementer.

Primary indices kan brukes når datafilen er sortert på search keyen og hver search key er kun for et entry i datafilen. Fordelene er at det er simpelt å implementere og ulempene er at hver search key kan kun brukes for et entry.

Cluster indices kan brukes når datafilen er sortert på search keyen. Fordelene er at det er mindre indekser som gjør at søket går raskere enn for primary og search keyen behøver ikke å være unik. Ulempen er at å finne senere records på en search key blir mer komplekst.

Secondary indices brukes når en datafil ikke er sortert eller sortert på en annen attribut. Fordelen er at det kan brukes når filen usortert eller sortert på en annen attribut, men ulempen er at søk er raskere når filen er sortert på atributten vi leter etter.

Inverted indices brukes for søk etter tekst og er mye brukt i søkemotorer. Fordelen er at det er en index metode som gir raske søk på tekst, deler av tekst eller tall. Ulempene er at det krever mere arbeid når datafiler legges til databasen.

B-trees er standard index for postgres og en mye brukt i databaser. Fordelene er at det veldig raskt for intervall søk, bruker et hierarkisk index for minimalisering av lesning fra disk og holder den holdes balansert av en rekursiv algoritme. Ulempen er at du må alltid starter i roten av treet og fjerning av elementer som ikke er leaf-nodes er mer komplisert.

Hash Tables er en annen form for indexing og brukes på databaser hvor det er mer effektivt å bruke den. Fordelene er at hash tables er raskere på noen søkemetoder som søk på spesifikke search keyes og vi kan få færre disk operasjoner enn med vanlige indices og B-trees. Ulempene er at flere av samme entry kan føre til at vi har flere blokker per bøtte og hash table er dårlig for intervall søk.

Multidimensional indices er bra når du har søk med en key av mange search keys. Fordelene er at søk med sett av flere nøkler er mer raskerer, men noen av ulempene er at det blir unødvendig kompleksitet hvis vi ikke trenger flere search keys eller ikke like effektivt hvis vi må se gjennom mange på dei først nivåene av multidimensjonelle indices.

Tree structures de forskjellige tree structure er bar for forskjellige ting men de er blant annet brukt i spatial databases. Tree-like strukturer har samme fordeler og ulemper som for B-trees.

Hash-like structures som grid files er bra for lagring med 1 eller flere nøkler. Fordelene er at bare de riktige dataene blir hentet og du kan drive med søk med 1 eller flere nøkler og finkre bra for intervall søk også. Ulempene er at grid files tar mye plass og å putt inn og fjerning av filer er vanskelig.

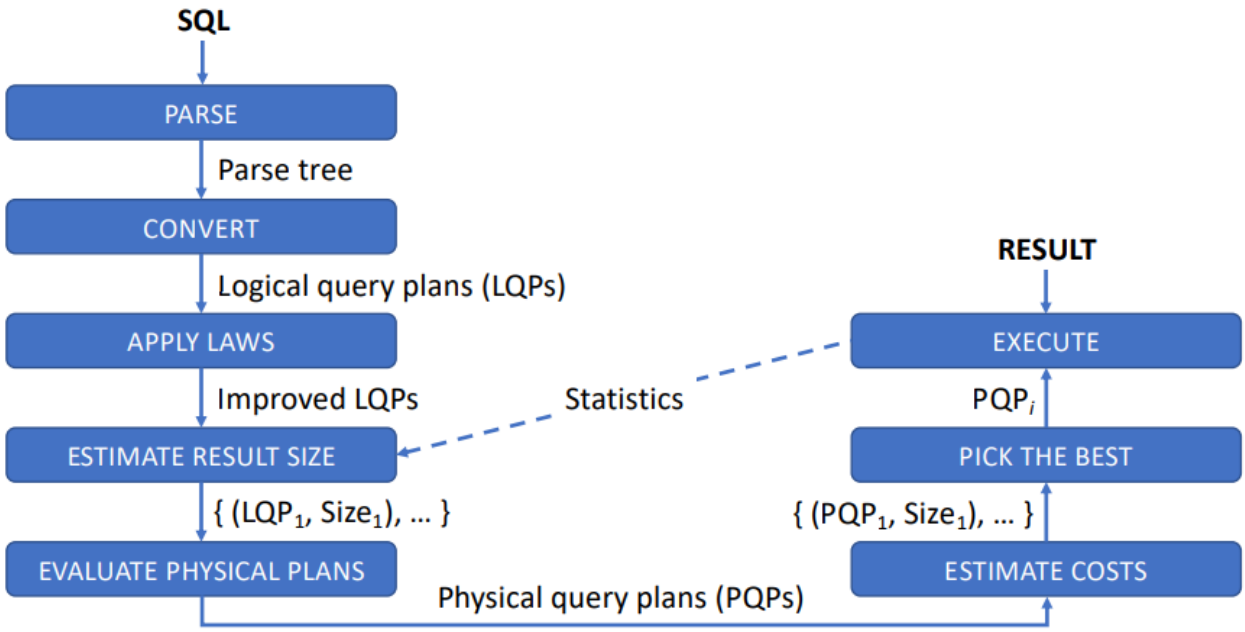
Bitmap indices brukes best når det ikke er så stor variasjon i attributten som med navn, det passer bedre for attributter som kjønn. Fordelene er at søk er raske sel for veldig store tabeller så lenge det ikke ser stor kardinalitet. Det gir også muligheten til å kombinere flere bitmap indices sammen i søk i databasen. Ulempene er at det ikke er effektivt får små tabeller og mye forandringer i tabellen(insert/update/delete) kan skape problemer. I tillegg så når tabellen er stor så tar det en del tid og en del arbeid å vedlikeholde bit mappen når nye ting legges til.

3 – Relational Algebra & Query Compilation

3.1 Query Processing Overview

We went through the steps of query processing as an introduction to query compilation. Sketch the steps with the inputs/outputs for each step and the relations between them, and describe their role/function briefly in your own words.

Svar:



Hentet fra uio powerpoint onsdag 10 februar 2020 slide 3, gitt i link 1 i refenangser.

Parse går vi gjennom sql setningen vi får og deler det opp i atom og syntaktisk kategori som den setter sammen til et Parse tree. Deretter går den i convert. Her forandrer vi parse tree om til "gjennomføringsplaner" i flere trinn. Her lager den mange forskjellige planer. Først lages mange logisk planer. Disse forskjellige logiske query planene sendes videre til apply laws for optimalisering. Her bruker vi Algebraic laws for å optimalisere de logiske query planene som vi fikk sendt inn og sender videre disse optimaliserte logiske query planene til estimate result size. I estimer resultat størrelse ser vi på de optimaliserte logiske query planene og estimerer hva størrelsen for de forskjellige blir. Dette sender vi til evaluate physical plan hvor vi velge en metode for å velge den "billigste" logiske query planen som ble sendt inn med size. Noen metoder er exhaustive, heuristic, branch-and-bound, hill climbing, dynamic programming og sellinger-style optimizations. Når vi har funnet "billigste" planene lager vi physiske query plans som sendes til estimate cost. Her estimates kostnadene av de forskjellige physical query planene og de physical query planene og deres størrelse sendes videre til pick the best. I pick the best går vi gjennom de forskjellige physical query planene og deres size for velge den beste. Denne physical query planen blir sendt til execute hvor den kjøres og gir resultatet. Statistisk data blir også lagret for senere bruk i estimering av størrelse.

3.2 Relational Algebra Expressions

Assume two relations R1(A, B, C, D, E) and R2(F, G, H). Assume that attributes E and F are of the same type, that attribute A is a character or string (variable or fixed) and that attribute H is a numeric value.

Write the equivalent of the following SELECT sentence in relational algebraic terms.

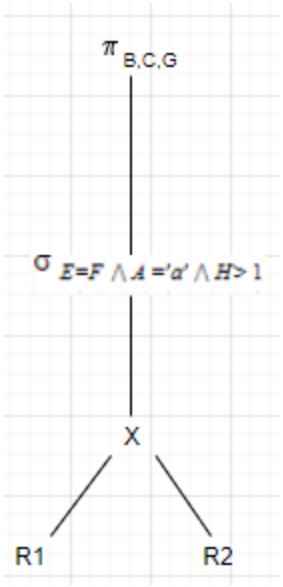
```
SELECT B, C, G
FROM R1, R2
WHERE E = F AND A = ´a´ AND H > 1
```

Svar:

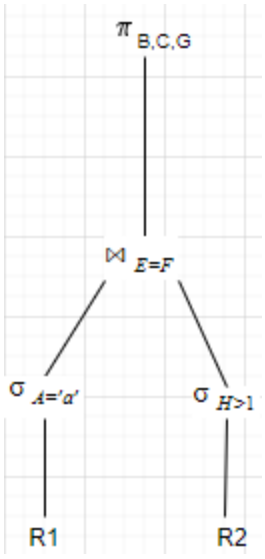
$$\pi_{B,C,G}(\sigma_{E=F \wedge A='a' \wedge H>1}(R1 \times R2))$$

3.3 Estimating Size (Logical Query Plans)

3.3.a Draw the relational algebraic tree directly from the relational algebraic expression you wrote in 3.2.



3.3.b Draw a second alternative relational algebraic tree of the same relational algebraic expression in 3.2, improving (optimizing) upon the first one by reordering the hierarchy of operations on the tree. Note that we are looking for your understanding of basic optimization rules, so make sure that the trees in 3.3.a and 3.3.b are different and that there is room for improvement from one to the other.

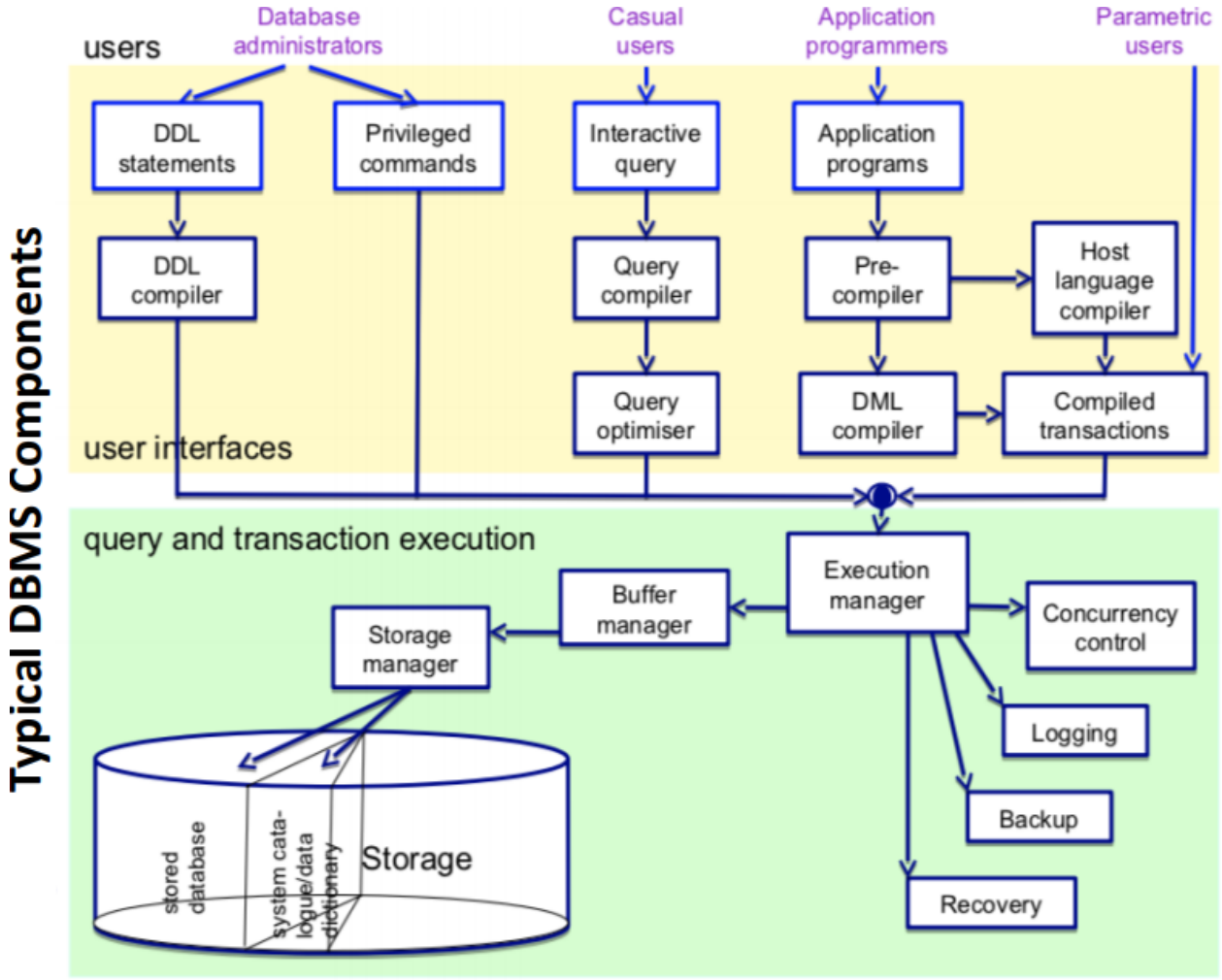


3.3.c Compare the two, explaining the improvement criteria, what you did and why, and explaining or demonstrating the improvement. You may assume that both relations have 10 tuples (rows). You can sketch for the purpose of explaining if you need to.

Svar:
Forbedringen mellom 3.3.a og 3.3.b er av vi har flyttet select lengre ned i relational algebraic tree og at vi splittet select conditions til flere select for å hente mindre antall tupler. Vi tenker at kostnaden er relatert til størrelsen på søke rommet vårt. Ved 3.3.a lager vi først en stor tabel ved å ta kryss produktet av R1 og R2, deretter gikk vi gjennom tabellen og hentet riktige tuplene og gir dem som output. Ved 3.3.b går vi først gjennom R1 og R2 og henter tuplene som stemmer med conditionene våre. Deretter equi-joiner vi disse tuplene og gir resultatet av equi-join som output. Tabellen vi lager for 3.3.b er mye mindre, som gjør at kostnaden er mindre hvis størrelse er kriteriet vårt.

4 – DBMS Architecture

Sketch the typical DBMS architecture we saw a few times during the beginning of the course and explain the role and job of each path and component of the architecture.



Bilde hente fra powerpoint fra onsdag 15 januar 2020 slide 3, gitt i link 2 i refenangser.

Øverst fra brukere har vi queries. Brukere kan sende inn queries for lesing av data(select) eller skriving av data(insert/update). Noen brukere vil også ha tilgang til Access control og DDL (Data Definition Language). Dette blir så kompilert av en compiler og eventuelt optimalisert før det sendes til Execution Manager. Execution manager leser tupler fra disk eller annet lagringsmedium og gjennomfører oppgaven i minnet. I Storage er tabellene lagret i filer, disse er gjerne delt opp i fixed størrelse blokker. Dette lar oss ha en array-lignende struktur som gjør at vi vet hvor blokken ligger i forhold til fil start. All data som hentes fra Storage blir passet på av Buffer manager som også passer på blokker som er i minnet. Buffer manager holder styr på om blokker er modifisert, hvem som er i bruk og hvor lenge blokker holdes i minnet. Det er vanlig at mange brukere jobber på samme database samtidig derfor har vi concurrency control Modulen.

Denne jobber for å minimalisere ventetid, fasiliterer valg av isolasjonsnivå for transaksjoner og interagerer ofte med buffer manager for å finne ut hva som leser og/eller modifiseres av hvem. Logging holder logg over alle transaksjoner og backup av viktig data sånn at hvis systemet krasjer eller serverne dør mens transaksjoner pågår så kan recovery fikse opp feilene ved enten rulle den avbrutte transaksjonen tilbake eller fullføre det som ikke ble gjennomført.

5 – Logs and their Uses

5.1 Log Functions

For what purposes do we use database logs? Use your own words to explain.

Svar:
Database logger bruker får å unngå å være i en ukonsistent fase, hvis en transaksjon blir avbrutt på en eller annen måte så må vi kunne fikse det. Dette gjør vi ved å skrive en logg før vi gjennomfører handlinger på disk. Dette kalles Write-Ahead Logging. Avhengig av hva vi skriver på loggen så gjør vi enter undo eller redo for å havne ut av denne ukonsistente fasen. Hvis den gamle verdien er i loggen så ruller vi tilbake tilstanden så at disken er som den var før transaksjonen begynnte, hvis den nye verdien er logget så endrer vi på disken sånn at den er i tilstanden den skulle ha vært i etter at transaksjonen var ferdig. Logging gjøres for at ikke viktig info skal gå tapt og at dataen kan settes til tilstanden den var i ved et eksakt tidspunkt.

5.2 Log Types

List the log types we have seen during the semester, explaining their uses and the differences between them.

Svar:
Vi har Undo, Redo og kombinasjonen Undo/Redo. I undo så skriver vi de gamle verdiene til log før vi skriver verdiene inn i tabellen. Når alle verdiene er i tabellen skrives commit til log. Denne ligging metoden kan føre til random writes og potensielt tap av data hvis prosessen blir forstyrret. Ved tap av data kan vi lese i loggen og gjenopprette sånn at Atomicity og Consistency opprettholdes for databasen. Redo skrives hele transaksjonen til log før den skrives inn i tabellene. Dette gjør at ingen forandringer skjer, da trenger vi ingen rollback hvis vi starter en transaksjon uten en commit eller abort, men hvis en avbrytelse skjer så har vi hele transaksjonen i log så vi vet hva de nye verdiene er, dette gjør at vi kan fullføre transaksjonen som ble avbrutt. Dette gjør at vi kan opprettholde Atomicity, Consistency og Durability for databasen. En ulempe med redo over undo er at hele transaksjonen må være i minne fram til vi får en commit som gjør at vi ikke alltid kan skrive til databasen når vi vil. Undo/Redo har fordelene fra begge med sine egne ulemper. Vi må logge for begge, men i tilfeller av avbrudd uten commit kan vi bruke undo til å havne tilbake til original tilstand, mens hvis vi har commit i logen kan vi bruke redo til å fullføre den avbrutte transaksjonene.

5.3 Understanding the Workings of a Log

You listed and explained the components of a typical DBMS architecture. Think of that architecture.

5.3.a Where does the log reside?
Svar:
Loggen er lagret på disk akkurat som tabellene

5.3.b Which components of a DBMS are involved in managing a log?
Svar:
Databasen sin Concurrency control and transaksjon manager samarbeid med buffer manager for logging av transaksjoner.

6 – Concurrency Control & Serialization
6.1 The Concepts Define and explain the concepts serial, serializable and conflict serializable.
Svar:
Gjennomføringen av en gruppe transaksjoner kan være serial hvis vi kun gjennomfører en transaksjon om gangen og vi må gjøre oss helt ferdig før vi begynner på neste.

En gjennomføring av en gruppe transaksjoner er serializable hvis gjennomførelsen gir samme svar som en serial gjennomføring kunne ha gjort.

Hvis vi har to gjennomføringsplaner så kan de kalles conflict equivalents hvis de kan forandres til hverandre gjennom en serie av bytte mellom naboliggende operasjoner så lenge de ikke lager konflikt hvis de bytter plass i gjennomføringsplanen. Hvis en gjennomføringsplan er conflict equivalents til er serial gjennomføringsplan så kaller vi den Conflict serializable.

6.2 Conflicts in Execution Plans
List, explain and exemplify the three types of conflicts in execution plans. Note that exemplify means give examples (at least one for each).
Svar:
1. Read-write conflict: en gruppe operasjoner i form av ...R_i(A) ...W_k(A) ...
or ...W_i(A) ...R_k(A) ... (where i ≠ k). Med andre ord så er det en read og en write av forskjellige transaksjoner på samme verdi som kan skape problemer.
2. Write-write conflict: en gruppe operasjoner i form av ...W_i(A) ...W_k(A) ...
(where i ≠ k)
3. Intra-transaction conflict: et par av form operasjoner ...O_i(A) ...O_i(B) ... where O_i
is W_i or R_i

Referangser:

1: henter 1.06.2020
https://www.uio.no/studier/emner/matnat/ifi/IN3020/v20/Lectures/052---query-compilation-part-1.pdf

2: henter 1.06.2020
https://www.uio.no/studier/emner/matnat/ifi/IN3020/v20/Lectures/012---architecture.pdf