Innlevering 1 IN3020

Pilasilda A. George
pilasila@uio.no

05.februar.2019

Oppgave 1

A

```
SELECT fornavn, etternavn, adresse FROM person p
WHERE p.fnr IN (
        SELECT fn.fnr
        FROM ekteskap e
        INNER JOIN ForrigeNavn fn
                ON fn.dato = e.dato
                        AND (fn.fnr = e.fnr1 OR fn.fnr = e.fnr2)
        WHERE e.etternavn1 NOT LIKE e.etternavn2
        AND e.etternavn1 NOT LIKE fn.etternavn
        AND e.dato >= '2000-1-1'
        AND e.dato < '2010-12-31';
```

B

```
--antar at det kun finnes reverserte etternavn og ikke på formen
--Andersen, Hansen à Andersen-Hansen, Hansen
SELECT fornavn, etternavn, adresse
FROM person p
WHERE p.etternavn IN (
        SELECT concat(e.etternavn, '-', e.etternavn2)
        FROM ekteskap e
        INNER JOIN ForrigeNavn fn
                ON fn.dato = e.dato
                        AND e.fnr1 = fn.fnr
        INNER JOIN ForrigeNavn fn2
                ON fn2.dato = e.dato
                        AND e.fnr2 = fn2.fn
        WHERE e.etternavn1 NOT LIKE e.etternavn2;
```

Oppgave 2

A

```
WITH recursive sti(fra, til, stien) AS(
                    SELECT s1.person, s2.person, array[s1.person, s2.person]
                    FROM selskapsinfo s1, selskapsinfo s2
                    WHERE s1.person = 'Olav Thorsen'
          AND s1.person <> s2.person     AND
s1.selskap = s2.selskap
UNION ALL
                    SELECT s.fra, s2.person, s.stien || s2.person
                    FROM sti s, selskapsinfo s1, selskapsinfo s2
                    WHERE s1.selskap = s2.selskap
                         AND s.til <> 'Celina Monsen'
                         AND s1.person <> s2.person
                         AND s.til = s1.person
                         AND s2.person <> ALL(s.stien)
)
SELECT min(cardinality(s.stien) -1)AS personSti
FROM sti s
WHERE s.til = 'Celina Monsen';
```

B

```
WITH recursive sykel(start, slutt, personen, selskapet) AS
(
        SELECT s1.person, s2.person, array[s2.person], array[s2.selskap]
        FROM selskapsinfo s1, selskapsinfo s2
WHERE s1.rolle = 'daglig leder' AND(
                        s2.rolle = 'styreleder' OR s2.rolle = 'nestleder' OR s2.rolle = 'styremedlem')
                        AND s1.selskap = s2.selskap AND s1.person NOT LIKE s2.perso n
UNION ALL
        SELECT s.start, s2.person, s.personen|| s2.person, s.selskapet || s2.selskap
        FROM selskapsinfo s1, selskapsinfo s2, sykel s
        WHERE s1.rolle = 'daglig leder' AND (s2.rolle = 'styreleder' OR s2.rolle = 'nestleder'
                        OR s2.rolle ='styremedlem') AND s.slutt = s1.person AND s1.selskap = s2.selskap
                        AND s1.person NOT LIKE s2.person AND s2.selskap NOT LIKE ALL(s.selskapet)
                        AND s2.person NOT LIKE ALL(s.personen)
)
SELECT start || personen as personsykler, selskapet
FROM sykel
WHERE cardinality(personen) between 3 AND 5 AND start = slutt;
```

Oppgave 3

A

```
SELECT s.maintitle, s.firstprodyear, COUNT(s.firstprodyear) AS episode
FROM series s
        LEFT JOIN episode e
        ON s.seriesid = e.seriesid
        WHERE s.firstprodyear = (SELECT MAX(firstprodyear) FROM series)
GROUP BY  s.maintitle, s.firstprodyear;
```

B

```
WITH antall_typer(parttype, antall) AS (
        SELECT distinct parttype, COUNT(*)
        FROM filmparticipation
        GROUP BY parttype), totalt AS  (
                SELECT COUNT(parttype) AS antalldeltagere
                FROM filmparticipation
)
SELECT parttype, antall, ROUND(antall*100.0/antalldeltagere, 1) AS prosent
FROM antall_typer, totalt
ORDER BY prosent DESC;
```

C3

--Finner mannlige og kvinnelige skuepiller men usikker på hvordan man skal finne samtlige filmer hun har spilt i dette fungerer derfor ikke, men mangler en liten del. Det jeg mangler er å finne hvor mange filmer det er per kvinnelige skuespiller per regissør. Dette antallet skal matche antallet for hvor mange filmer regissøren har regissert.

```
WITH reg AS(
  Select fp.filmid, t1.pid
  From filmparticipation fp inner join
  (SELECT p.personid as pid
  FROM person p
  NATURAL JOIN filmparticipation f
  WHERE f.parttype LIKE 'director' AND p.gender = 'M'
  GROUP BY p.personid
  HAVING  COUNT(f.filmid) > 5) as t1 on t1.pid = fp.personid
  Order by pid;
)

SELECT p1.personid,reg.filmid
FROM reg INNER JOIN filmparticipation fp ON reg.filmid = fp.filmid INNER JOIN
person p1 ON fp.personid = p1.personid
WHERE fp.parttype LIKE 'cast' AND p1.gender = 'F';
```

Oppgave 4 A

Explain analyze

```
HashAggregate  (cost=11294.63..11303.13 rows=850 width=30) (actual time=127.280..127.286 rows=6 loops=1)
  Group Key: s.maintitle, s.firstprodyear
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=1153.84..1153.85 rows=1 width=4) (actual time=14.863..14.863 rows=1 loops=1)
          ->  Seq Scan on series  (cost=0.00..1006.67 rows=58867 width=4) (actual time=0.002..5.736 rows=58867 loops=1)
  ->  Hash Right Join  (cost=1164.48..10092.39 rows=6453 width=22) (actual time=127.261..127.265 rows=6 loops=1)
        Hash Cond: (e.seriesid = s.seriesid)
        ->  Seq Scan on episode e  (cost=0.00..7756.02 rows=446402 width=4) (actual time=0.032..41.137 rows=446402 loops=1)
        ->  Hash  (cost=1153.84..1153.84 rows=851 width=26) (actual time=20.203..20.203 rows=6 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 9kB
              ->  Seq Scan on series s  (cost=0.00..1153.84 rows=851 width=26) (actual time=15.926..20.174 rows=6 loops=1)
                    Filter: (firstprodyear = $0)
                    Rows Removed by Filter: 58861
Planning Time: 0.381 ms
Execution Time: 127.374 ms
(15 rows)
```

En explain plan leses alltid nedenifra og oppover, fordi man ikke kan lese sekvensielt nedover eller oppgaver fordi hva som skjer i spørringen avhenger av andre ting.

HashAggregate: Det første som skjer når spørringen kjøres er at den velger en aggregat type, i dette tilfellet velges hashaggregat, denne typen har som fordel at den er raskere enn groupAggregat, men ulempen er at den benytter mye minne.

Group key: Det siste som skjer i spørringen er gruppering av tittel og produksjonsår, derfor sjekkes dette først.

Init plan: subspørringen i spørringen kan kjøres seperat og er ikke avhengig av andre verdier i spørringen. Derfor leses dette ved hjelp av en seq scan, seq scan leser alle tupler i tabellen og gjennomfører en aggreatfunksjon. Resultatet som kommer ut av subspørringen lagre og blir brukte til resten av spørringen.

Seq scan: det utføres en seq scan på series, dette innebærer at alle tupler i seriestabellen blir les, etter det regnes hash av det som blir lest i seq scan, det gjøres ved en hash right join.

Hash right join: spørringen har betingelse på at vi joiner på e.serieid = s.serieid dette sammenlignes, ved series sin hash og episode sin hash som kommer beskrivelse under. Hvis disse betingelsene gir en match vil det bli et nytt tuppel som oppfyller disse betingelsene. **Seq scan 2:** det utføres så en seq scan av episode tabellen og regnes også her hash for hver tuppel. Ser også ut som at denne spørringen blir lagret i buckets, så en bucketsort algoritme.

B

```
pilasila=> \d series
        Table "public.series"
    Column     |  Type   | Modifiers
---------------+---------+-----------
 seriesid      | integer |
 maintitle     | text    | not null
 firstprodyear | integer |
Indexes:
    "seriespkey" UNIQUE CONSTRAINT, btree (seriesid)
    "idx_series" btree (maintitle, firstprodyear)
    "seriesmaintitleindex" btree (maintitle)
Foreign-key constraints:
    "seriesfkey" FOREIGN KEY (seriesid) REFERENCES filmitem(filmid)
Referenced by:
    TABLE "episode" CONSTRAINT "episodefkeyseriesidseries" FOREIGN KEY (seriesid) REFERENCES series(seriesid)
```

Lagde en btree index på maintitle og firstprodyear, kan se at execution time blir mye mindre med et btree index. Årsaken til at jeg valgte btree index er fordi jeg ved flere anledninger kom borti eksempler og forklaringer hvor de mener at btree index brukes for spørringer hvor det utføres flere operasjoner som å finne max og andre aggregat funksjoner blant annet, for operasjoner som dette vil btree index egne seg best da dette vil gi best «performance» altså kjøres tid på O(1). I følge postgres manualen også handler btreer etter likhet og rekkevidde spørringer av data som kan bli sortert. I følge PostgreSQL vil btree index brukes når kolonner er involvert i en sammenlignings operator som = eller >. For spørringer hvor vi bare vil hente ut noe og ikke gjøre noe «utregninger» vil hashindex egne seg bedre. I bildet under kan man se at execution time er betydelig mye mindre etter btree index sammenlignet med det det var før.

```
HashAggregate  (cost=91.20..91.55 rows=35 width=31) (actual time=3.027..3.042 rows=22 loops=1)
  Group Key: s.maintitle, s.firstprodyear
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=40.46..40.47 rows=1 width=4) (actual time=1.676..1.677 rows=1 loops=1)
          ->  Seq Scan on series  (cost=0.00..35.37 rows=2037 width=4) (actual time=0.010..0.670 rows=2037 loops=1)
  ->  Hash Right Join  (cost=40.90..50.47 rows=35 width=23) (actual time=2.840..2.958 rows=22 loops=1)
        Hash Cond: (e.seriesid = s.seriesid)
        ->  Seq Scan on episode e  (cost=0.00..8.41 rows=441 width=4) (actual time=0.016..0.157 rows=441 loops=1)
        ->  Hash  (cost=40.46..40.46 rows=35 width=27) (actual time=2.511..2.512 rows=22 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 10kB
              ->  Seq Scan on series s  (cost=0.00..40.46 rows=35 width=27) (actual time=1.790..2.464 rows=22 loops=1)
                    Filter: (firstprodyear = $0)
                    Rows Removed by Filter: 2015
Planning Time: 1.138 ms
Execution Time: 3.220 ms
```