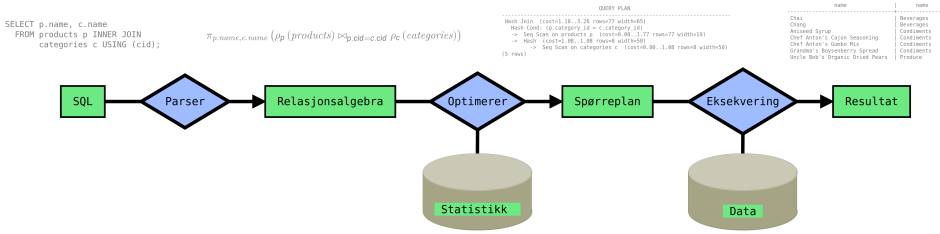
IN2090 – Databaser og datamodellering

15 – Indekser og Spørreprosessering

Leif Harald Karlsen
leifhka@ifi.uio.no

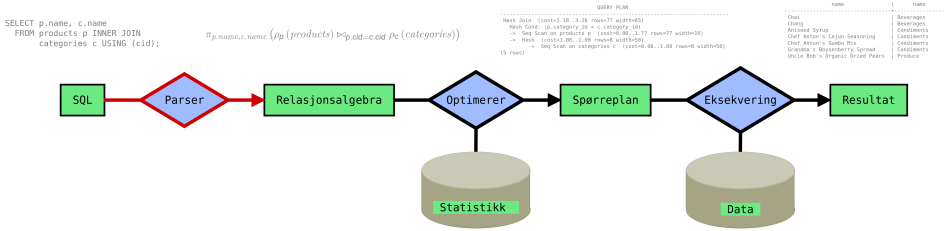
Universitetet i Oslo

Spørreprosessering: Oversikt



- En spørring går gjennom flere steg før den til slutt blir evaluert over dataene
- Disse stegene sørger for at spørringen blir besvart så effektivt som mulig
- Vi skal også se at vi av og til må be databasen lage datastrukturer for å effektivt kunne besvare enkelte spørringer

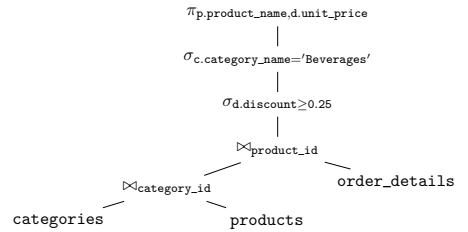
Fra SQL til relasjonell algebra



- Det første som skjer er at spørringen sjekkes syntaktisk (f.eks. tabellene og kolonnene finnes i databasen, typene er riktige, osv.)
- Deretter oversettes spørringen til et spørre-tre over (en utvidet) relasjonell algebra

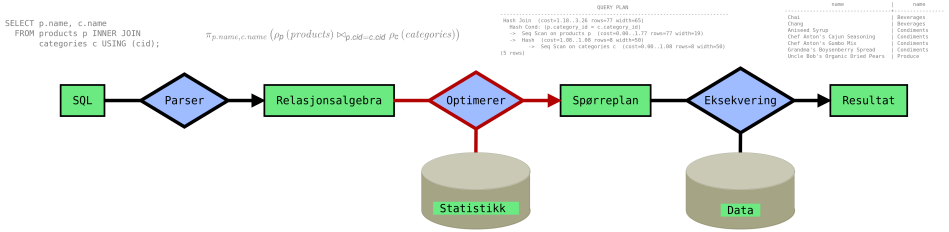
Oversettelse: Eksempel

```sql
SELECT p.product_name, d.unit_price
FROM categories AS c JOIN
  products AS p USING (category_id) JOIN
  order_details AS d USING (product_id)
WHERE c.category_name = 'Beverages'
  AND d.discount >= 0.25;
```



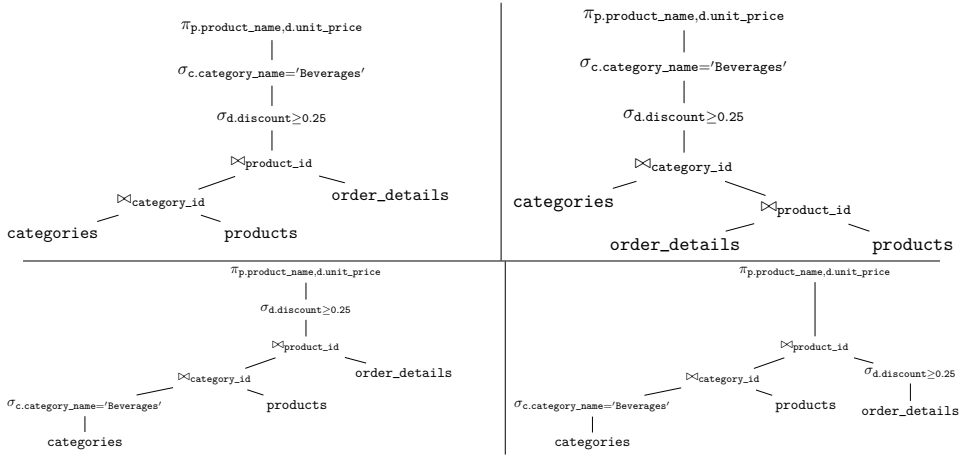
$$\pi_{\texttt{p.product\_name,d.unit\_price}}\Big(\sigma_{\texttt{c.category\_name}='\texttt{Beverages}'}\Big(\sigma_{\texttt{d.discount}\geq 0.25}(\texttt{categories} \bowtie_{\texttt{category\_id}} \texttt{products} \bowtie_{\texttt{product\_id}} \texttt{order\_details})\Big)\Big)$$

Ulike spørringer – Likt resultat

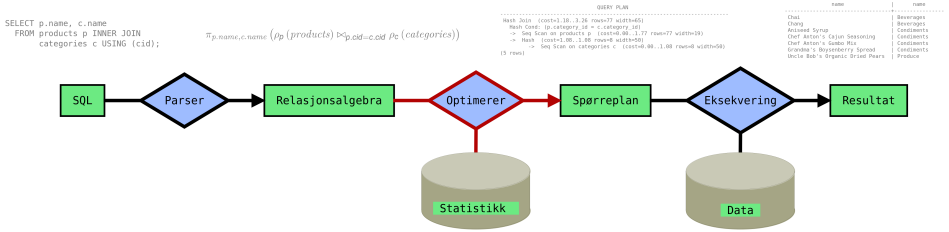


- Spørringen uttrykt i relasjonell algebra kan manipuleres algebraisk
- Dette brukes for å genere forskjellige men ekvivalente spørringer
- Altså, spørringer som gir samme svar, men ser forskjellige ut
- Forskjellige spørringer kan ha ulik kompleksitet
- De ulike spørringene skal så (i neste steg) bli tilordnet en ca. kostnad
- Vi vil så velge den spørringen som er billigst å eksekvere

Ulike spørringer: Eksempel

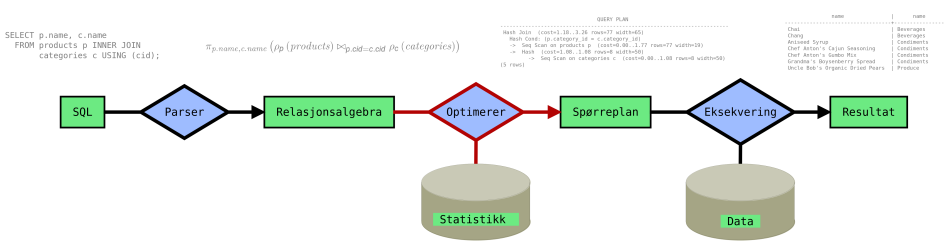


Fra spørring til kostnad



- De ulike spørringene blir så tilordnet en kostnad
- Kostnadsevalueringen bruker statistikk over databasen
- F.eks. antall rader i hver tabell, antall ulike verdier i hver kolonne, osv.
- Bruker her også skranker (f.eks. `UNIQUE`, `CHECK`)
- Høyere kostnad betyr lengre eksekveringstid
- Databasen velger så den spørringen med lavest kostnad

Spørreplaner



- Det siste som skjer i dette trinnet er at det blir laget en spørreplan for den valgte spørringen
- Dette er en mer detaljert plan for hvordan spørringen skal eksekveres

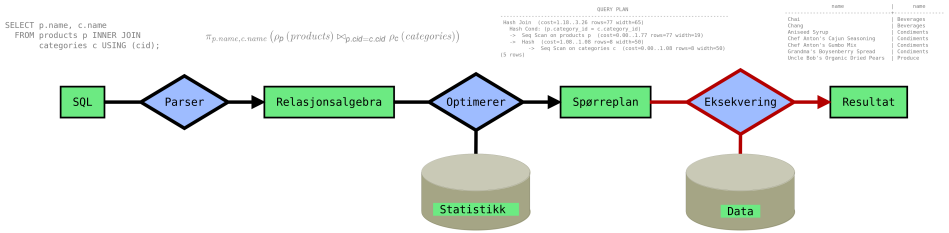
EXPLAIN

- Av og til kan det være nyttig å få se denne spørreplanen
- F.eks. dersom man lurer på hvordan spørringen vil bli eksekvert
- Eller dersom man ønsker et ca. estimat på hvor komplisert spørringen blir å eksekvere
- Dette kan gjøres ved å skrive `EXPLAIN` foran spørringen
- Spørringen blir da ikke eksekvert
- (Merk: Ikke pensum å kunne forstå spørreplaner!)

EXPLAIN: Eksempel

```
psql=> EXPLAIN SELECT p.product_name, d.unit_price
FROM categories AS c JOIN
  products AS p USING (category_id) JOIN
  order_details AS d USING (product_id)
WHERE c.category_name = 'Beverages'
  AND d.discount >= 0.25;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=3.32..43.04 rows=20 width=21)
   Hash Cond: (d.product_id = p.product_id)
   ->  Seq Scan on order_details d  (cost=0.00..38.94 rows=154 width=6)
         Filter: (discount >= '0.25'::double precision)
   ->  Hash  (cost=3.20..3.20 rows=10 width=19)
         ->  Hash Join  (cost=1.11..3.20 rows=10 width=19)
               Hash Cond: (p.category_id = c.category_id)
               ->  Seq Scan on products p  (cost=0.00..1.77 rows=77 width=21)
               ->  Hash  (cost=1.10..1.10 rows=1 width=2)
                     ->  Seq Scan on categories c  (cost=0.00..1.10 rows=1 width=2)
                           Filter: ((category_name)::text = 'Beverages'::text)
(11 rows)
```


Evaluering



- Til slutt evalueres spørringen over databasen
- Databasen har så svært effektive algoritmer for joins, oppslag, sorteing, osv.
- Merk: Databasen trenger kun én algoritme per operator i den (utvidede) relasjonelle algebraen

ANALYZE

- Dersom vi ønsker å vite hvor lang tid en spørring faktisk tar å eksekvere, samt detaljert analyse av hver del av spørreplanen kan vi bruke `EXPLAIN ANALYZE`
- Får da også informasjon om minnebruk
- Da vil spørringen bli eksekvert, og databasen samler så nøyaktig informasjon om eksekveringen
- Dersom en spørring tar lang tid kan dette brukes for å finne ut hvilken del av spørringen som er komplisert
- Kan også brukes for å finne manglende indeksstrukturer (mer om dette straks)

`ANALYZE`: Eksempel

```
psql=> EXPLAIN ANALYZE SELECT p.product_name, d.unit_price
FROM categories AS c JOIN
  products AS p USING (category_id) JOIN
  order_details AS d USING (product_id)
WHERE c.category_name = 'Beverages'
  AND d.discount >= 0.25;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=3.32..43.04 rows=20 width=21) (actual time=0.130..1.066 rows=32 loops=1)
   Hash Cond: (d.product_id = p.product_id)
   ->  Seq Scan on order_details d  (cost=0.00..38.94 rows=154 width=6) (actual time=0.031..0.887 rows=154 loops=1)
         Filter: (discount >= '0.25'::double precision)
         Rows Removed by Filter: 2001
   ->  Hash  (cost=3.20..3.20 rows=10 width=19) (actual time=0.085..0.085 rows=12 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Hash Join  (cost=1.11..3.20 rows=10 width=19) (actual time=0.034..0.077 rows=12 loops=1)
               Hash Cond: (p.category_id = c.category_id)
               ->  Seq Scan on products p  (cost=0.00..1.77 rows=77 width=21) (actual time=0.008..0.022 rows=77 loops=1)
               ->  Hash  (cost=1.10..1.10 rows=1 width=2) (actual time=0.016..0.016 rows=1 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on categories c  (cost=0.00..1.10 rows=1 width=2) (actual time=0.008..0.012 rows=1 loops=1)
                           Filter: ((category_name)::text = 'Beverages'::text)
                           Rows Removed by Filter: 7
 Planning Time: 0.567 ms
 Execution Time: 1.146 ms
(17 rows)
```

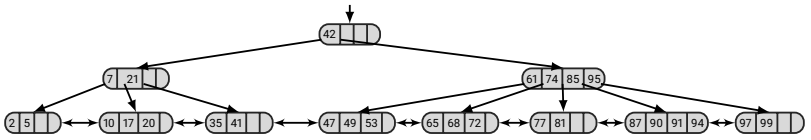

Spørringer og kompleksitet

- Hvordan utfører en database et oppslag på en bestem verdi?
- Eller en join mellom to tabeller?
- Begge disse problemene er egentlig et søk etter bestemte verdier i en kolonne
- For at databasen skal kunne utføre disse operasjonene effektivt (spesielt over veldig store tabeller) trenger vi datastrukturer som gjør søket mer effektivt
- Slike datastrukturer heter indeksstrukturer

Indeksstrukturer

- En indeksstruktur er en datastruktur som lar databasen hurtig finne bestemte rader i en tabell, basert på verdiene i en (eller flere) kolonner
- Har to hovedtyper indekser: Hash-baserte og tre-baserte
- Databaseindekser skiller seg litt fra andre datastrukturer fordi de ikke lagres i minne, men på disk
- Å lese fra disk tar ca. 10,000 ganger lengre tid enn fra RAM (avhenig av disktype og minnetype)
- Se f.eks. `https://gist.github.com/hellerbarde/2843375`
- De er derfor optimert for å utføre så få diskoppslag som mulig
- Databasen finner selv ut når indeksen bør brukes

B-tre-indekser



- Trestruktur hvor hver node kan ha mange barn
- Nodene har samme størrelse som en disk-blokk
- Minimerer antall oppslag på disk
- Hver verdi i løvnodene har pekere til dens tilhørende rad i den tilhørende tabellen
- Kan utføre effektive oppslag på konkrete verdier
- Samt effektive intervall søk

Hash-indekser

- En hash-indeks bruker en hash-funksjon for å oversette en verdi til en minneadresse
- På minneadressen ligger så en liste med pekere til rader som har denne verdien
- Hash-indekser er mer effektive på oppslag på konkrete verdier
- Men kan ikke brukes for intervaller (må da gjøre ett oppslag for hver mulige verdi i intervallet)

Andre indeksstrukturer

- Det finnes mange andre indeksstrukturer
- Ulike strukturer er tilpassed ulike datatyper
- Egne strukturer for f.eks.:
  - Søk i tekst
  - Romlige data og koordinater i høyere dimensjoner
  - Sammensatte strukturer (JSON, XML, osv.)

Nøkler og indekser

- Når man markerer en kolonne med `PRIMARY KEY` blir det automatisk opprettet en B-tre-indeks på denne kolonnen
- Joins over primærnøkler er derfor alltid relativt effektive
- Men, det kan hende man ønsker å gjøre søk, oppslag eller joins over kolonner som ikke er primærnøkler
- Vi må da lage indeksene selv

Lage indekser med SQL

- For å lade en indeks på en kolonne trenger man bare å skrive

  ```
  CREATE INDEX <index_name> ON <table>(<columns>);
  ```

  hvor `<index_name>` er navnet på indeksen, `<table>` er et tabellnavn og `<columns>` er en liste med kolonner man ønsker å indeksere
- Databasen lager da en passende indeks (typisk B-tre)
- F.eks.:

  ```
  CREATE INDEX price_index ON products(unit_price);
  ```

- Merk: Dersom man lister opp flere kolonner blir indeksen over alle kolonnene samtidig

Relasjonelle databasers sukksess

- Spørringene vi skriver er deklarative
- De uttrykker *hva* ikke *hvordan*
- Databasen bruker algebra og statistikk for å finne den mest effektive måten å eksekvere spørringen på
- Den samme spørringen kan altså bli eksekvert på forskjellige måter dersom dataene endrer seg
- Relasjonelle databaser er et felt som kombinerer avansert teori (algebra, logikk, statistikk) med sofistikerte algoritmer og data strukturer

Hva bør repeteres?

Menti