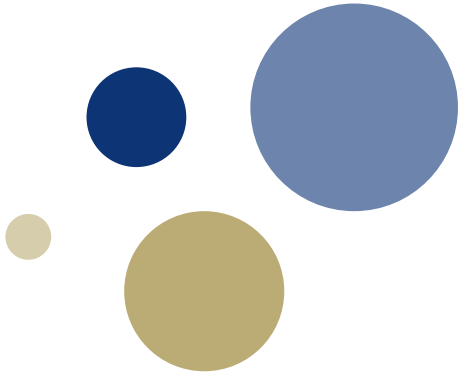
NTNU

Kunnskap for en bedre verden



# Algoritmer for queryprosessering og optimalisering (seksj. 12, 13 og 14 i notatet)

Svein Erik Bratsberg, IDI/NTNU

# Fra Query til resultat



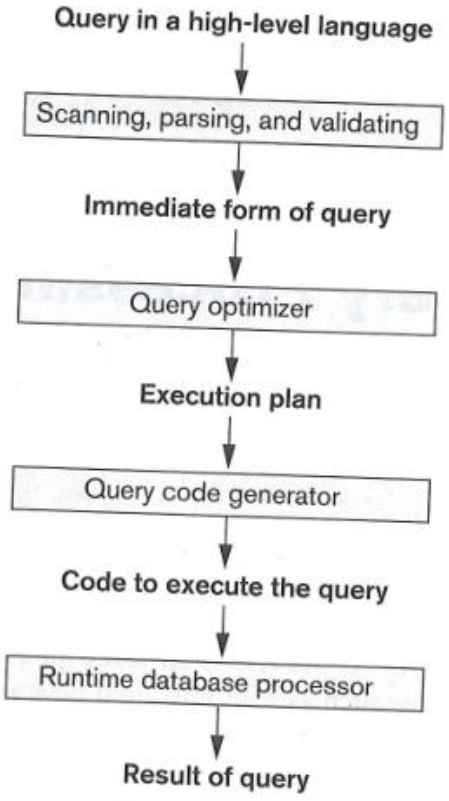


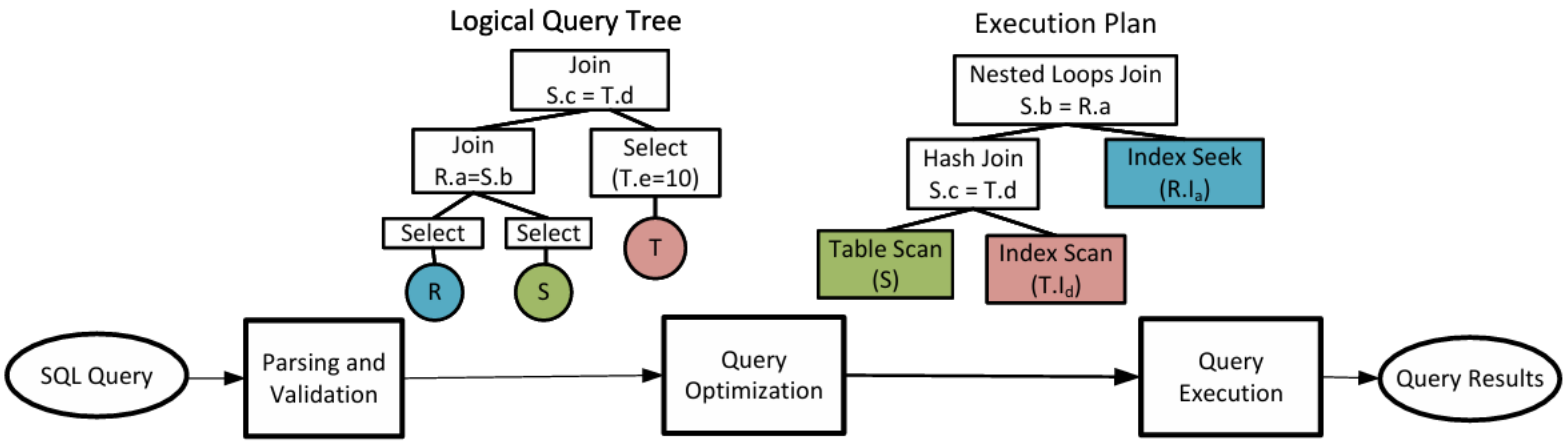

**Figure 18.1**
Typical steps when processing a high-level query.

**Code can be:**

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

# Fra Algebra til utføringsplan



# Teknikker for å utføre relasjonsalgebraoperasjoner

- **Indeksering**: Bruk WHERE-uttrykk til å trekke ut små mengder poster (seleksjon, join)
- **Iterasjon**: Ofte er det raskest å scanne hele tabeller
- **Partisjonering**: Sortering og hashing av input gir operasjoner på mindre datamengder
- NTNU er kjent som pionerer på hashbaserte metoder (partisjonering). Kjell Bratbergsengen, VLDB 1984.

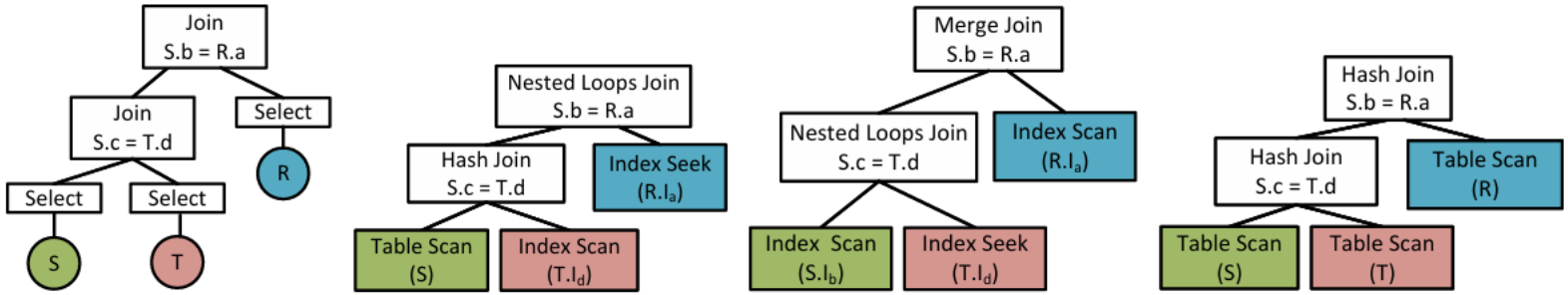
# Statistikk om data

- For hver tabell
  - # rader
  - # blokker
- For hver indeks
  - # nøkkelverdier
  - # blokker
  - Histogrammer
- For hvert B+-tre
  - Trehøyde
  - LowKey
  - HighKey
  - # blokker

# Aksessvei

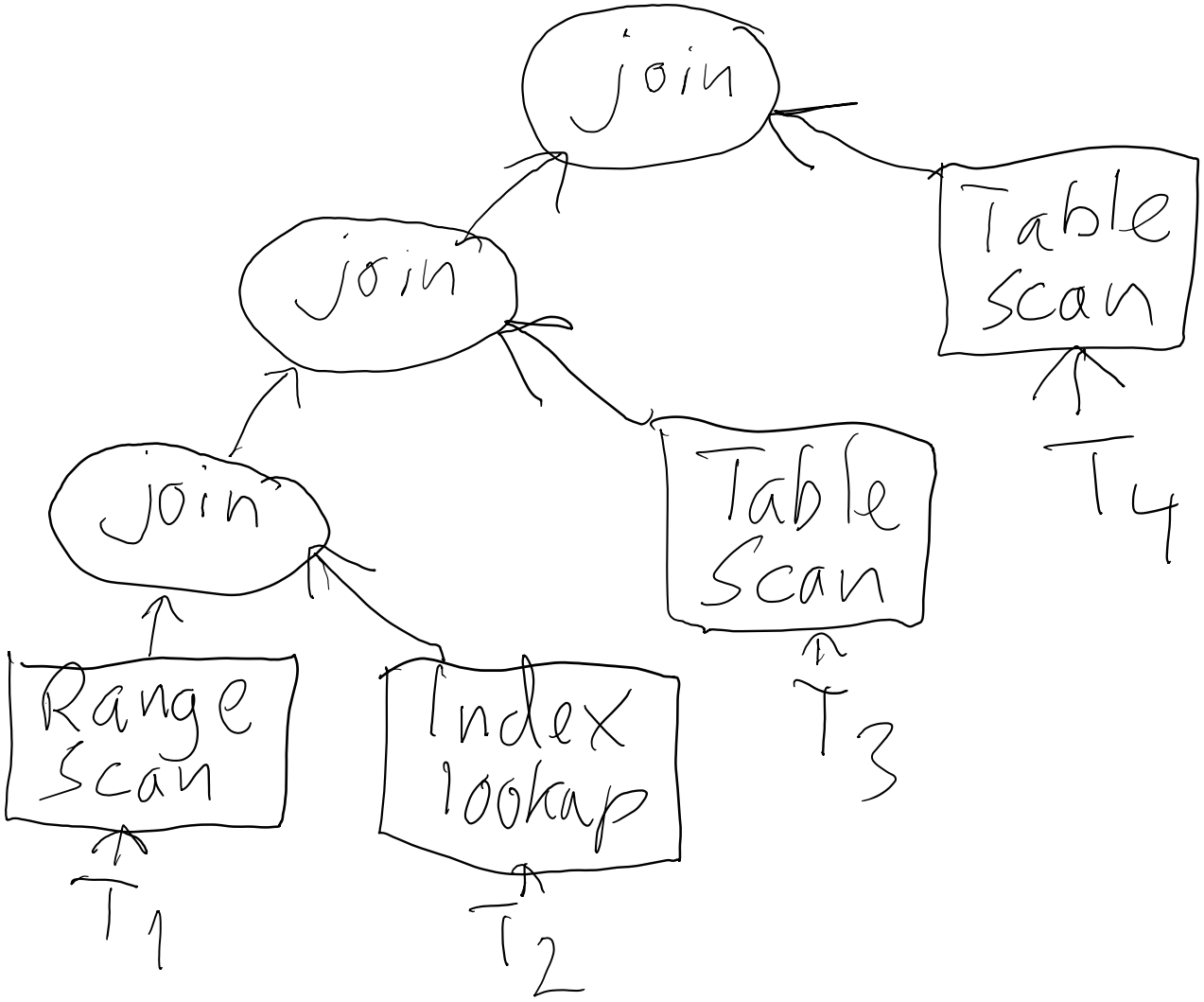
- Access path / search method (E & N)
- Optimalisatoren velger den billigste aksessveien
- Måles i *antall blokker* som aksesseres (+ CPU-bruk)
- **1. Filscan (tabellscan)**
- **2. Indeks**
  - Indeksscan
  - Rangescan
  - Indeks lookup

# Forskjellige utføringsplaner




**(a)** Logical plan $L$ **(b)** Execution plan $P_1$ **(c)** Execution plan $P_2$ **(d)** Execution plan $P_3$

# Utføringstre – left deep plans



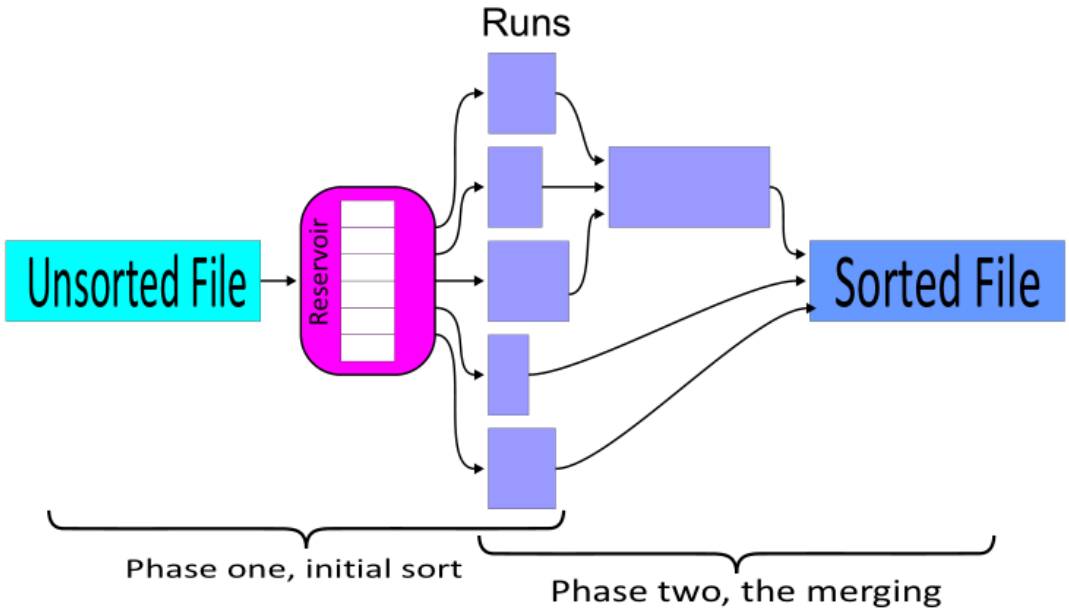
# Bruk explain i MySQL

- MySQL leser tabellene i den gitte rekkefølgen
- ALL: Full table scan

```
mysql> explain select * from reg natural join loper;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+---------------------------------
---------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
|
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+---------------------------------
---------+
|  1 | SIMPLE      | loper | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL |    3 |   100.00 | NULL
|
|  1 | SIMPLE      | reg   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   10 |    10.00 | Using where; Using join buffer
(hash join) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+---------------------------------
---------+
2 rows in set, 1 warning (0.01 sec)
```

# Flettesortering

- Merge-sort: Sortering av store datamengder
- 2 faser



# Flettesortering (2)

- 1. Partisjonering
  - Sorterer deler (partisjoner) som får plass i RAM
    - Antall deler (partisjoner): $n_R$
    - Antall blokker av data: $b$
    - Tilgjengelige buffer: $n_B$
- 2. Fletting
  - Flett sammen sorterte delfiler i et antall pass
  - Flettegrad $d_M$ – antall delfiler som kan flettes i hvert pass
  - Flettepass – antall pass som filene må flettes i
- Regner ut total I/O – antall blokker som leses/skrives

# Metoder for enkle seleksjoner (18.3.1)

- Lineært fil/tabellscan
- Bruk B+-tre eller hashindeks (clustered index)
- Bruk sekundærindeks (unclustered index)

# Metoder for utføring av join

- J1: Nested-loop join
  - For hver blokk i den ene tabellen
    - Scan hele den andre tabellen og se etter match
- J2: Single-loop join (index nested loop)
  - Loop gjennom den ene tabellen og bruk en indeks for å slå opp i den andre
- J3: Sort-merge join:
  - Hvis begge tabellene er sorterte på joinattributtene, kan vi bare flette de
  - Hvis ikke, kan de sorteres først, og så flettes
- J4: Partition-hash join
  - Partisjoner tabellene ved hashing på joinattributtene
  - Får mange små partisjoner som kan joines parvis i RAM