CS698F Advanced Data Management

Instructor: Medha Atre

Announcements

- Assignment-1:
  - Paper and topics due by Aug 30, 11:59pm.
  - Sept 6 – 2 presentations
  - Sept 8 – 2 presentations
- Course project topic due: Aug 25, 11:59pm
  - Send a short proposal citing paper/s chosen and the broad theme of your course project.

Recap

- Query plan generation using relational algebra rules.
  - Left-deep vs bushy plans.
- Popular types of indexes.
  - Tree indexes – B+ trees, B trees
  - Hash indexes – static, linear, extendible

Recap

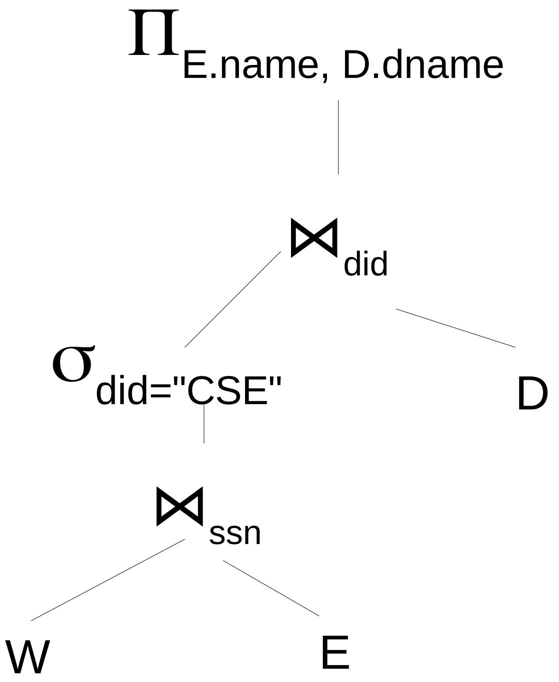
- Special types of indexes.
  - Bitmaps – when #unique values in a column are small.
  - Bloom filters – when "negative" queries are frequent – queried value does *not* exist. May give false positive answer, but never false negative.
- Choice of indexes
  - Depend on "*workload*" – types of queries plus data characteristics

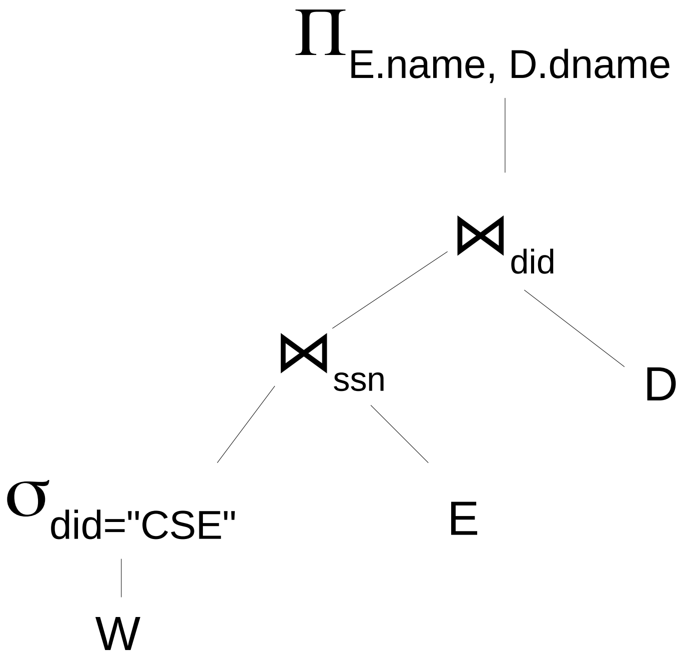
Query Optimization 101

- Query rewriting a.k.a. considering various query plans for the *same effective results*.
  - Relational algebraic equivalences help
- Indexes on the tables a.k.a. access methods
  - Types of indexes – B+ trees, Hash index, others we will see in the contexts of different data types.
- Join methods and their costs
  - Nested-loop, sort-merge, index-nested-loop join, hash join etc.
- Finally combining the above two together for cost optimization.

Which join methods?

SELECT E.name, D.dname
FROM WorksIn2 as W, Employees as E, Department as D
WHERE
W.did="CSE" **AND** W.did=D.did **AND** W.ssn=E.ssn





Types of joins

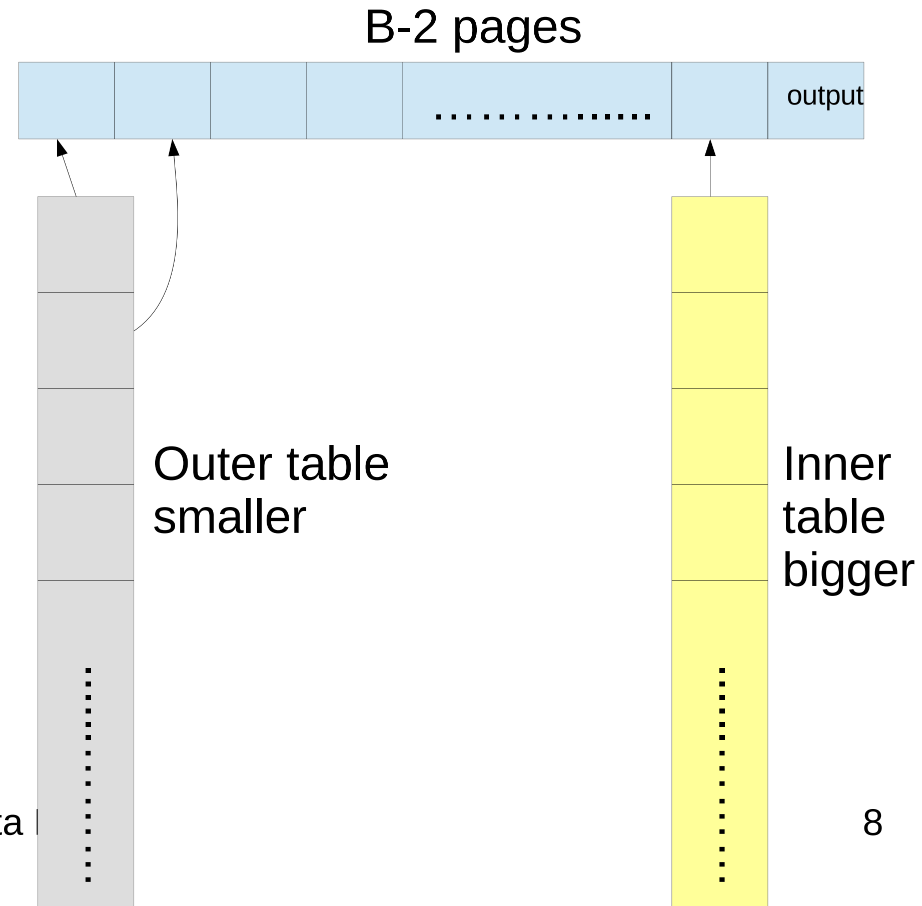
- Block-nested-loop join
  - When none of the tables have indexes and none of them are sorted on the join attributes.
- Index-nested-loop join
  - When one relation has an index on the join attribute.
- Merge-join
  - When both the relations have respective indexes on the joined attributes.
- Sort-merge-join
  - Sort both the relations on the join attribute first and then merge.
- Hash-join
  - Partition the attribute values from both the tables into *k* buckets and then join pairwise bucket.

Block-nested-loop join

Let there be *B* buffer pages available.

```
While (R not done) {
  for each page of B-2 pages of R do {
    for each page of S do {
       match in-memory tuples of B-2
       pages of R with S' one page tuples
       Add ⟨r,s⟩ to the result page
    }
  }
}
```

Cost: M + M*N
M: pages in outer relation
N: pages in inner relation



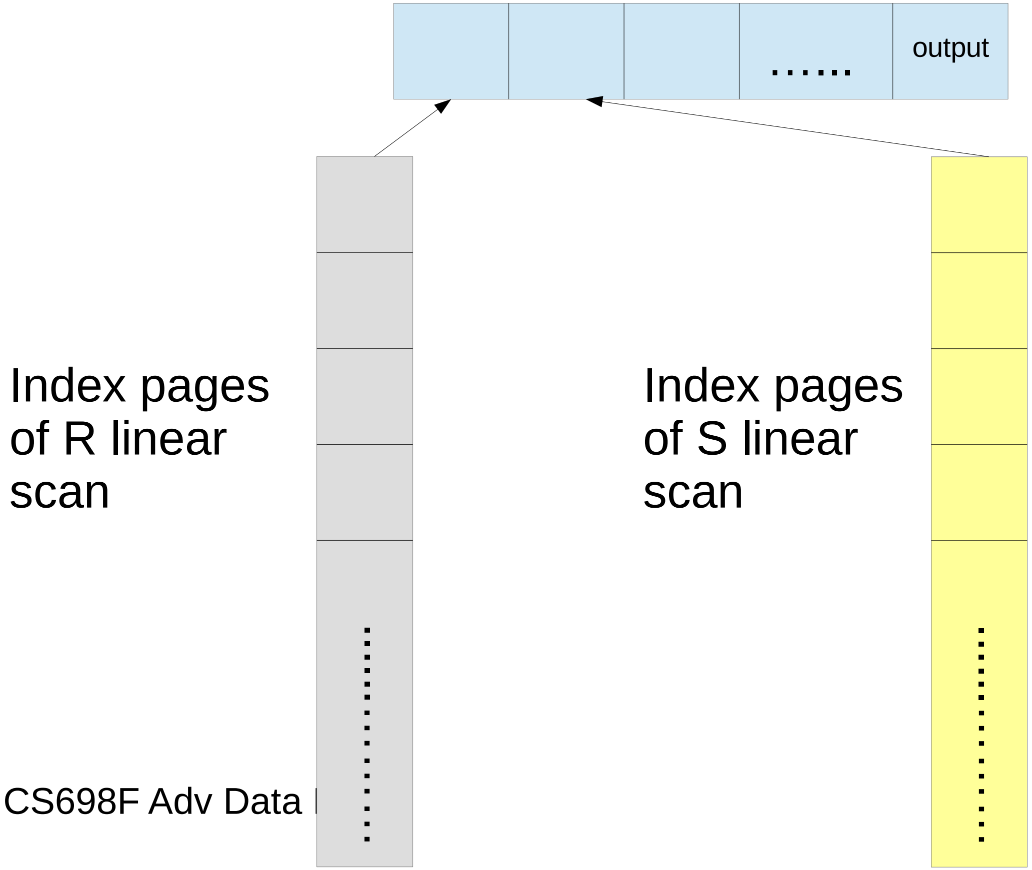
Index-nested-loop join

- Similar to block-nested-loop join
- Difference – the relation that has index is *always* the inner relation!
  - Why?
- Cost analysis: outer relation scanning – M pages
- Inner relation scanning – depends on the index
  - B+ tree – height of the tree, typically 2–4 for about 1 million entries.
  - Hash-index – 1 or 2 I/Os – depending on the hash-levels and type.
- For each page of outer relation and each tuple in it, do an index lookup
- Cost: B+ tree -- M + M*(#tuples-per-page) * (2 to 4)
- Cost: Hash-index – M + M*(#tuples-per-page) * (1.2)

Why would you Choose index-nested loop join over block nested one?

Merge-join

- When both the relations have respective indexes on the join columns
- Cost: 2 (M+N)



Sort-Merge-Join

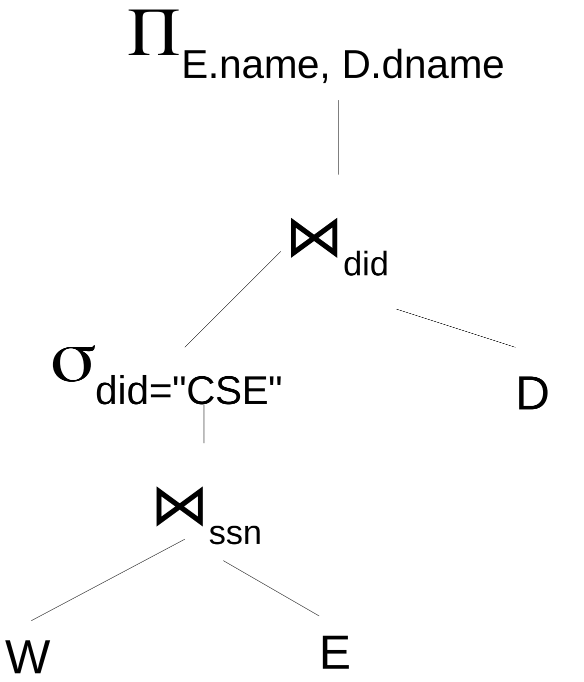
- Sort the two relations first and then do a merge-join
- I/O cost of sorting
  - $2 * M (\log_{B-1} M + 1)$
  - $2 * N (\log_{B-1} N + 1)$
- Cost of merging: $2 * (M + N)$
- Total cost: $2 * (M (\log_{B-1} M + 1) + N (\log_{B-1} N + 1) + M + N)$

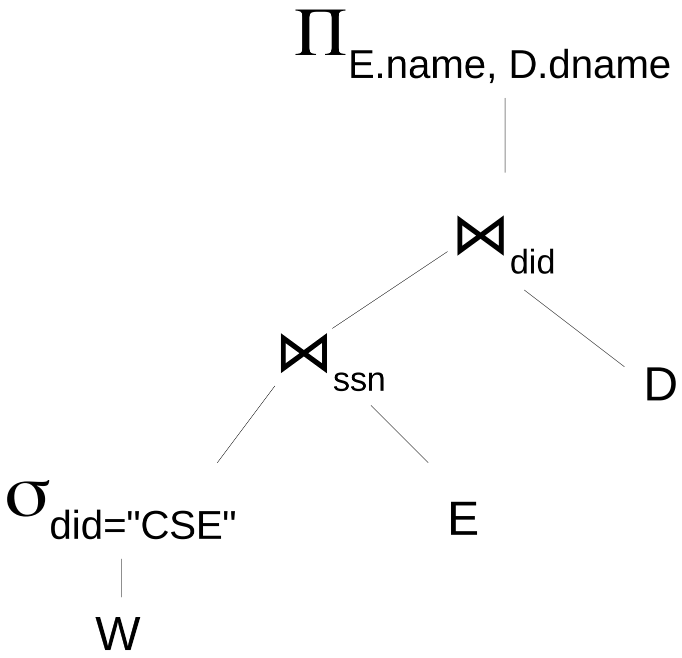
Hash-Join

- Hash and partition the two relations in *k* buckets each
  - Cost: 2 * (M + N)
- Scan each partition pairwise (corresponding i^{th} partition of R and S) and join
  - Cost: 2 * (M + N) – once for reading the partition and once for writing out the join results.
- This looks very good, then why not just do a hash-join *always*?

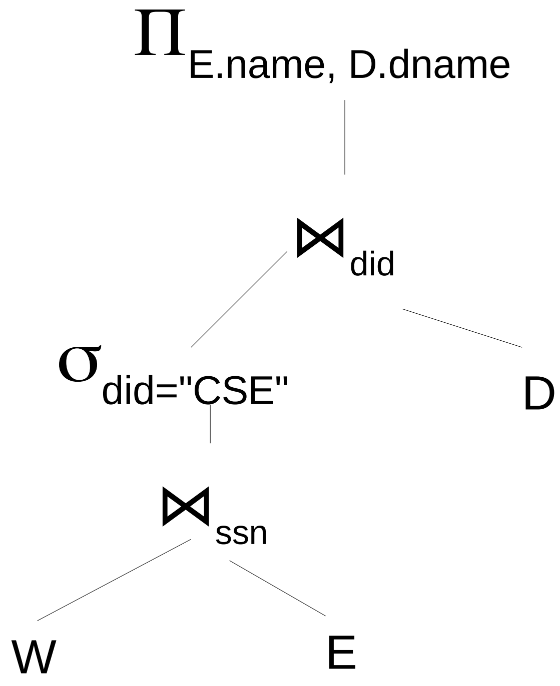
Cost estimation in detail

SELECT E.name, D.dname
FROM WorksIn2 as W, Employees as E, Department as D
WHERE
W.did="CSE" **AND** W.did=D.did **AND** W.ssn=E.ssn



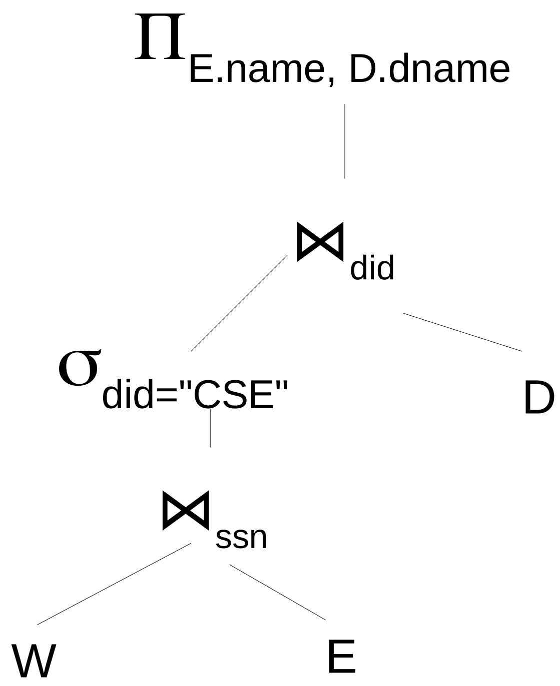


Cost estimation



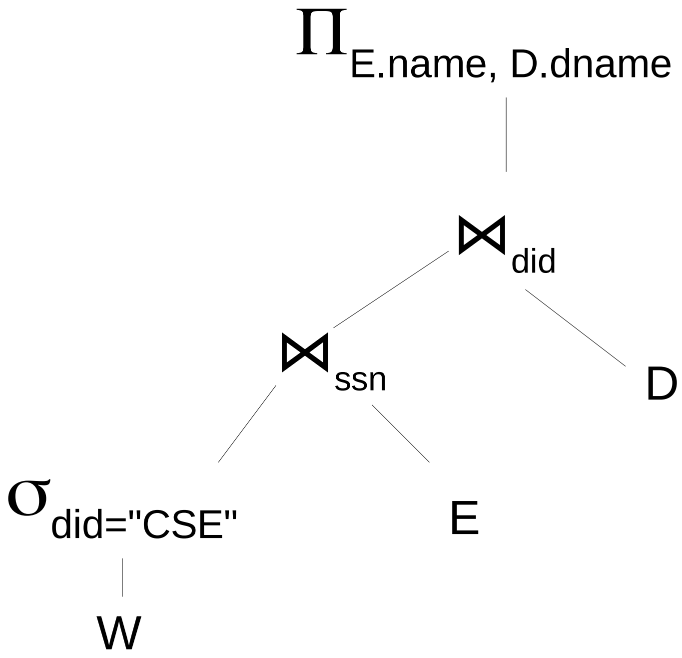
- Consider first join $\bowtie_{ssn}$
  - Does W and E have an index on *ssn*?
  - Is *ssn* the primary key of any of the relations?
- From the above, estimate the number of tuples to be processed.
- Using #tuples (in turn #pages), consider various join methods
  - Estimate the cost of various joins
  - Pick the least cost one store this cost in a dynamic prog memoization table!

Cost estimation



- Move a step higher – selection condition $\sigma_{did="CSE"}$
  - No index, tuplewise scan over temp table of prev join
  - Cost: No added cost!
    - Why?
- #tuple estimates: thumb rule 1/10 * #tuple estimates from prev join.
- Move a step higher – join condition
  - Similar analysis as previous join

Cost estimation



- Consider $\sigma_{did="CSE"}$
  - Does *W* have an index on *did*?
- #tuple estimates:
  - If index: exact # tuples
  - If not: 1/10 of all tuples
- Move a step higher – join $\bowtie_{ssn}$
  - Take #tuples estimate from prev selection
  - Does E have an index on ssn?
  - Consider various join plans with #tuples from W *after* selection and from E (depending on if index or not)
- So on…..

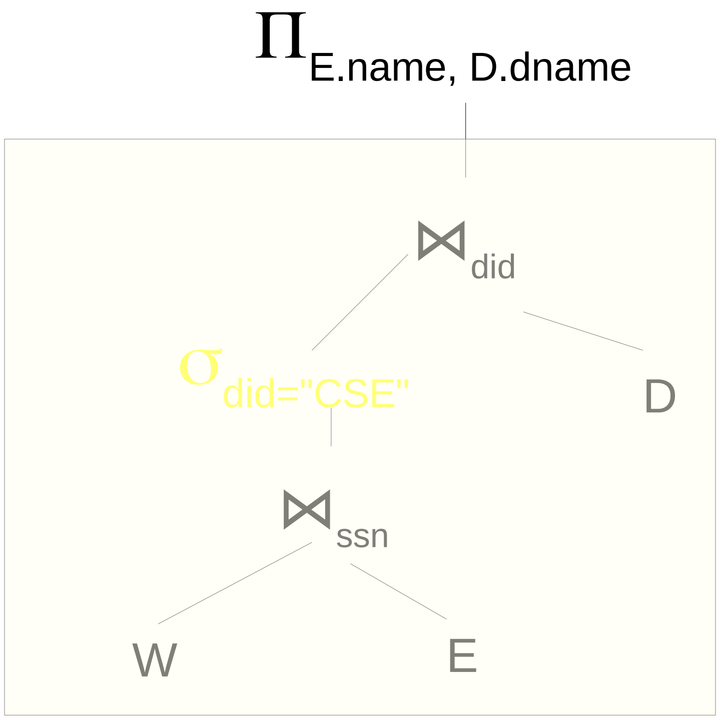
Thumbrules!

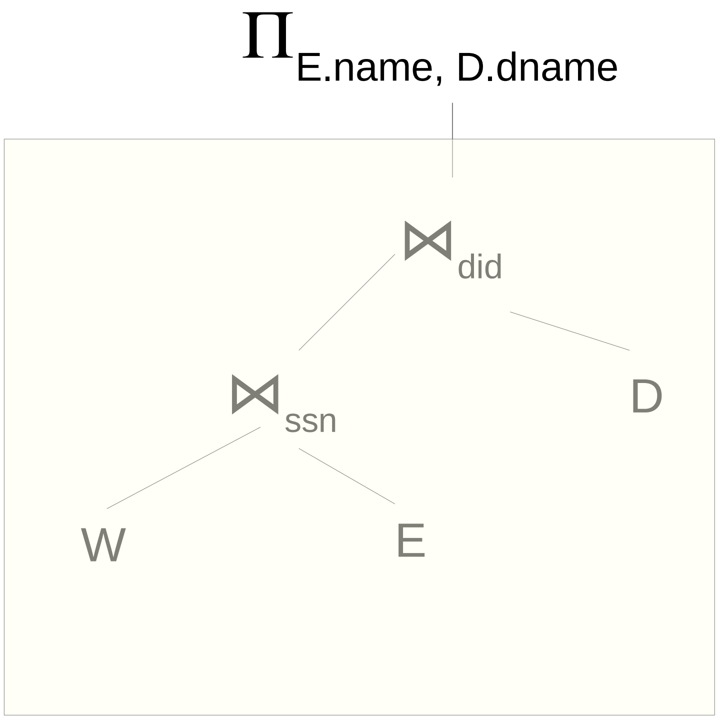
- Not possible to do *exact* result cardinality estimate
  - Hence DB query optimization has been researched for a long time.
- Thumbrules *WHERE* clause:
  - Column = value (selection)
    - If index: 1/Nkeys(I) * #tuples, no index: 1/10 * #tuples
      - Why?
  - Column1 = column2 (join)
    - If index on both: 1/max(Nkeys(I1), Nkeys(I2)) * #tuples(T1) * #tuples(T2)
      - Why?
    - If no index: 1/10 * #tuples(T1) * #tuples(T2)

Improvements

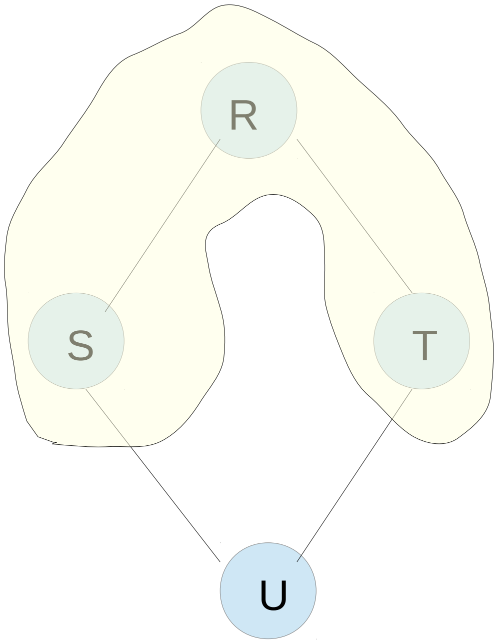
- Improved statistics
  - Histograms: maintain cardinalities for each unique value, if not uniform distribution
  - Useful when small # of unique values distributed over a large number of rows
- Join tables:
  - If some joins are observed to be frequent, preserve their join results.
  - Mining into the query logs, and pattern recognition!

Pattern recognition in queries





Pattern recognition in queries



- Techniques like approximate pattern match.
- Subgraph isomorphism
- Since query graphs are very small NP properties do not matter.
- One time activity!