COMP207 Database Development

Tutorial 6 (Week 8)

Relational Algebra-2

Which is the Best Plan?

- Physical Query Plan Operators?
  - Scanning tables
    - ‘table-scan’ – basic approach to read all of the relation block-by-block
    - ‘index-scan’ – retrieve data blocks via an index
  - Sorting-While-Scanning (e.g. ORDER BY …)
    - ‘sort-scan’
      - If indexed on a B+ tree then scan the index to produce an ordered relation
      - If relation is small and fits into main memory – retrieve and use a memory sort algorithm
      - If relation large – use a merge-sort

Query Tree
Step-1: Set of Relations

- EMPLOYEE (Ssn, Fname, LastName, Bdate, Address, Sex, Salary, Dno)
- DEPARTMENT (Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- PRESENTATION (Pnumber, Pname, Plocation, Dnum)

Query Tree
Step-2: Define the SQL Query

“For every presentation located in Stafford, list the presentation number, controlling department number and the manager’s last name, address and date-of-birth”

- Uses relations DEPARTMENT, EMPLOYEE and PRESENTATION
- Needs a SELECT-PROJECT-JOIN query

Query Tree
Step-3: The Algebra for the Query

(SQL from Step-2)

- SELECT P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate FROM PRESENTATION AS P, DEPARTMENT AS D, EMPLOYEE AS E

  WHERE P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation='Stafford';

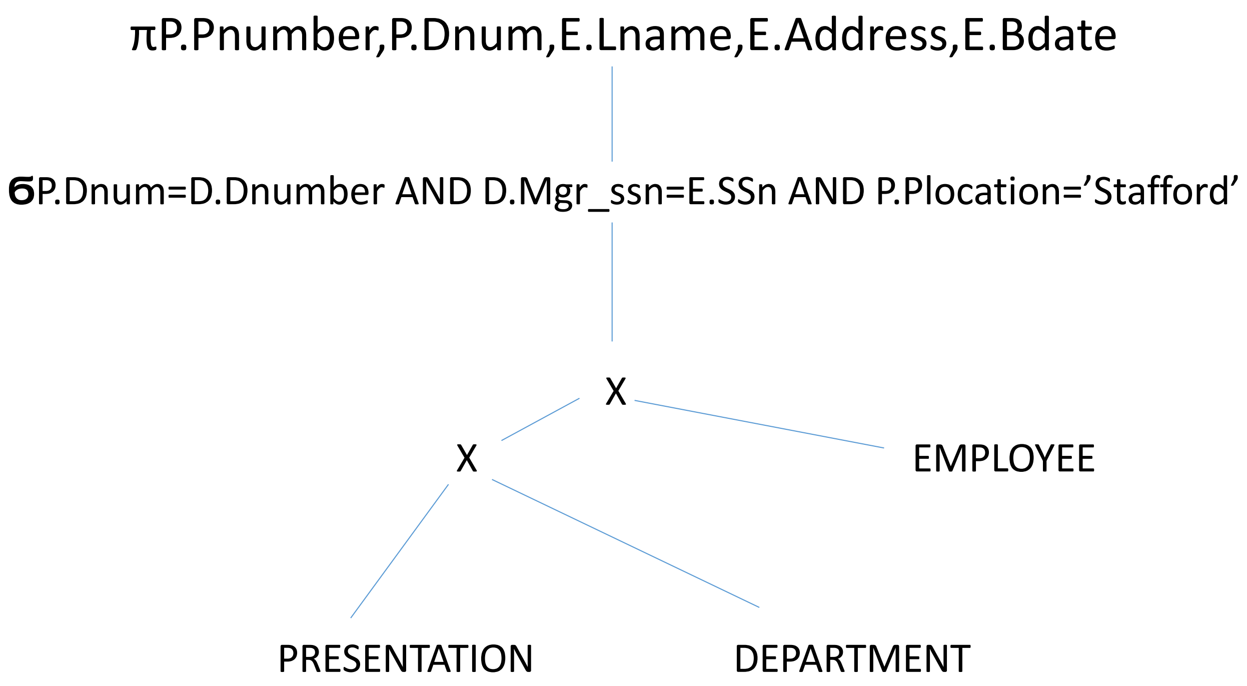
Algebra for the SQL from Step-2

- π Pnumber,Dnum,Lname,Address,Bdate (((Ϭplocation='Stafford'(PRESENTATION)) ⋈ Dnum=Dnumber(DEPARTMENT)) ⋈ Mgr_ssn=ssn(EMPLOYEE))

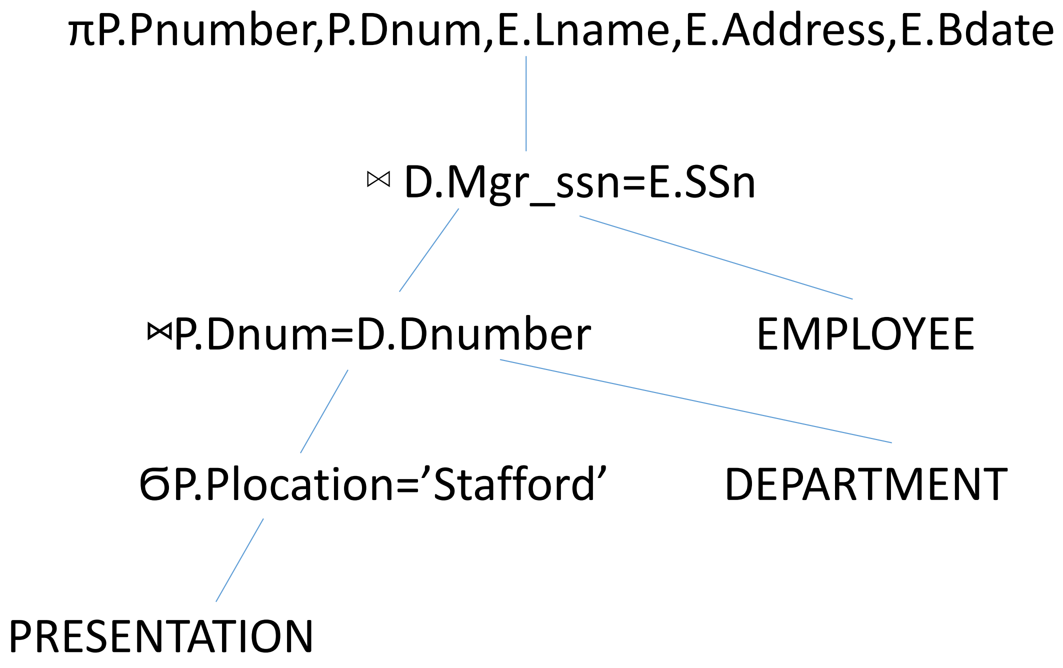
Query Tree
Step-4: Query Tree

- SQL query is scanned and parsed to generate an Initial Query Tree
- Initial Query Tree is optimised using heuristic rules and modified to improve performance by producing an ‘Equivalent Query Tree’
- Equivalent Query Tree gives a different relational algebra expression: Same result, but more efficient

Query Tree
Step-5: Initial Query Tree


πP.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
ϭP.Dnum=D.Dnumber AND D.Mgr_ssn=E.SSn AND P.Plocation='Stafford'
X
X
EMPLOYEE
PRESENTATION
DEPARTMENT

Query Tree
Step-6: Equivalent Query Tree



Query Tree Materialisation

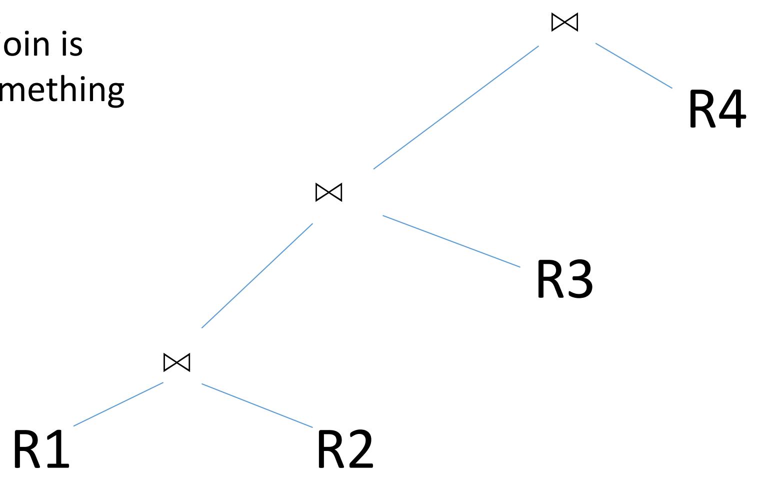
- π title,year(Ϭlength>=100 AND studio='Fox'(Film))
- Start at lowest levels (bottom of tree)
- Lowest levels have inputs which are the relations
- Lowest level operations execute - store on disk as temporary relations
- Temporary relations are 'materialised' as a relation of 'Fox' tuples and another one of '>= 100' tuples
- Inputs to the next level of the tree are these materialised relations

Left Deep Join Tree

- To avoid estimating cost for every possible tree, query optimiser prunes the possible trees typically to a left-deep tree (can use right-deep tree)
- These trees are suitable for pipelining
- The tree with the lowest cost is chosen to execute the query

Left Deep Join Tree

Only left side of join is allowed to do something



Left child is the outer relation, right child is inner relation.
In each join, one of the inputs is a base relation
Inner relations are materialised as we need to examine each tuple of inner relation against each tuple of the outer relation

Query Tree Materialisation

- JOIN can be implemented using two SELECTs - one on each of the two input files and then PROJECT onto a results file
  - Uses two input files and one output file
- Pipelining (‘Stream-based processing’)
  - Removes overhead to produce a temporary file (‘materialisation’) holding intermediate result and then reading that result back in again
  - Results are passed upward during execution ‘on the fly’
  - Intermediate results are stored in buffers and then discarded
  - Runs through the entire tree for tuples extracted rather than doing the operation on all tuples

Query Tree
Materialisation - Pipelining

- Ϭ position = ‘Manager’ ^ salary > 20000 $^{\text{(Staff)}}$
- Ϭ position = ‘Manager’ (Ϭ salary > 20000 $^{\text{(Staff)}}$)
- Assume index on salary, no pipelining
- First select (inner) result is materialised (stored in a temporary relation)
- Second select (outer) is applied to the materialised relation
- Assume index on salary – pipelined
- Second select result is applied to the first select result as it executes using a pair of buffers to pass the first select result to the second select result