QUERY PROCESSING AND OPTIMISATION

CT230 Database Systems I

RECALL: Definition of Query Processing

Transforms SQL (high level language) in to a **correct** and **efficient** low level language representation of relational algebra

Each relational algebra operator has code associated with it which, when run, performs the operation on the data specified, allowing the specified data to be output as the result

Representing the relational algebra solutions with a query tree

What is a tree?

A tree is a collection of data arranged as a finite set of elements - called **nodes** - such that:

The tree is empty or the tree contains a distinguished node, called the **root node,** and all other nodes are arranged in subtrees such that each node has a parent node. Nodes typically contain *data* and some pointers to other nodes

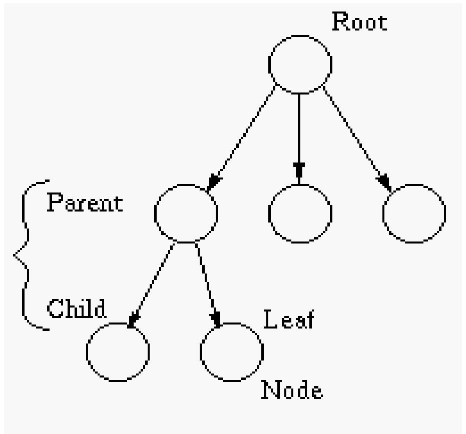


Figure: Tree data structure

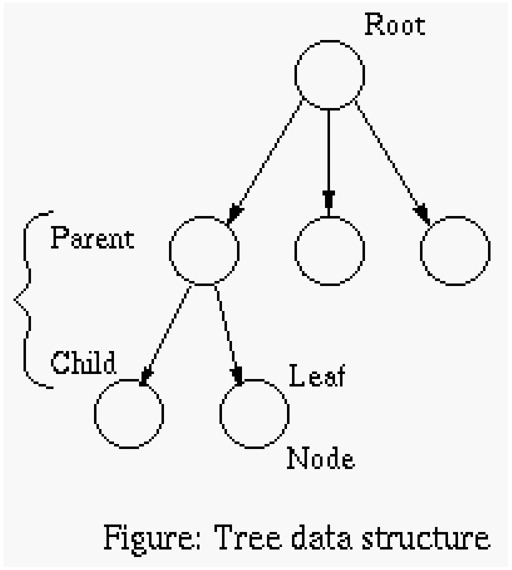
TREES

Nodes may be:

root: no node points to it

inner: has parent and child nodes

leaves: has no child nodes



Tree data structures (a grouping of data) are used frequently in computing allowing data to be stored in a non-linear (non-list) way.

They are often (but not always) **binary trees** where each node can have at most two child nodes

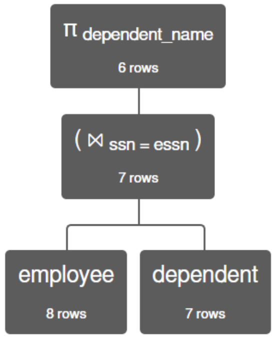
QUERY TREE

A **query tree** is a binary tree that corresponds to a relational algebra expression where:

- (input): tables are at the leaf nodes
- relational algebra operators are at internal nodes
- (output/result): the root of the tree returns the result (often with one final relational algebra operator)

The sequence of operations is directed from leaves to root and from left to right – e.g. the bottom-most, left-most side of tree is executed first

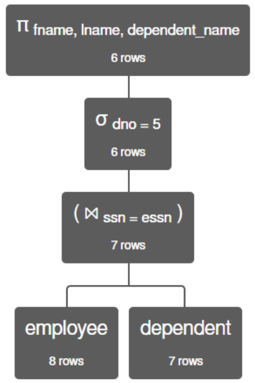
EXAMPLES: all dependent names



$\pi_{\text{dependent\_name}}(\text{employee} \bowtie_{\text{ssn = essn}} \text{dependent})$

dependent.dependent_name
'Michael'
'Alice'
'Elizabeth'
'Theodore'
'Joy'
'Abner'

EXAMPLES:
employees from department 5 and their dependents



$\pi_{\text{fname, lname, dependent\_name}}\ (\ \sigma_{\text{dno} = 5}\ (\ \text{employee} \bowtie_{\text{ssn} = \text{essn}} \text{dependent}\ )\ )$

employee.fname	employee.lname	dependent.dependent_name
'John'	'Smith'	'Michael'
'John'	'Smith'	'Alice'
'John'	'Smith'	'Elizabeth'
'Franklin'	'Wong'	'Alice'
'Franklin'	'Wong'	'Theodore'
'Franklin'	'Wong'	'Joy'

How to Translate SQL to Relational Algebra?

- SELECT *attributes* corresponds to π
- Joins correspond to relational algebra joins $\bowtie$ with any join conditions specified <u>as part of the join</u>
- Any conditions in a WHERE clause correspond to a sigma σ relational algebra operator with associated conditions
- In addition, have rules for aggregate functions (sum, avg, count, etc.) and GROUP BY, HAVING and subqueries but we won’t consider these

Executing query represented by query tree: one approach:
Materialization Evaluation

Traverse tree from bottom to top, left to right. At each stage:

- Execute internal node operation whenever data for its child nodes are available
- Replace the internal node operation (and all child nodes) by the table resulting from executing the operation

Note: Results of operations are saved as temporary tables and are used as inputs to other operators

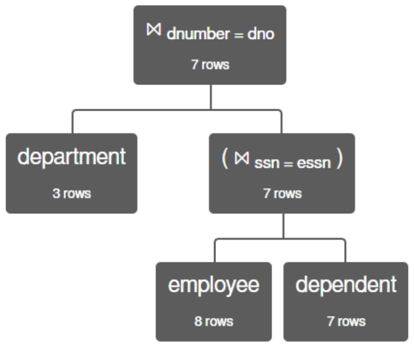
HOW TO DRAW A QUERY TREE?

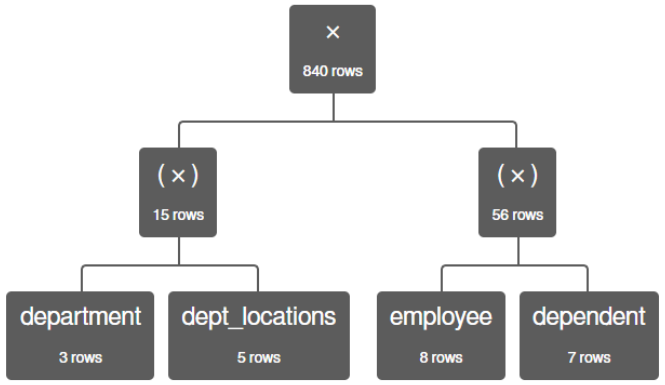
Must remember the order of execution – from bottom to top, completing each level and then left to right of tree – therefore:

- the first operations – fetching tables – should be at the leaves of trees.
- the last operator – often π or aggregate functions - should be at the root of the table.
- joins must be applied to tables (2 at a time) and should be at internal nodes.
- any other operators should be at one or more internal nodes.

IMPORTANT

When Joining or multiplying more than two tables … operators can only be applied to 2 operands at a time





department $\bowtie_{\text{dnumber = dno}}$ (employee $\bowtie_{\text{ssn = essn}}$ dependent)

(department × dept_locations) × (employee × dependent)

ANNOTATING TREE

Each relation algebra operation can be evaluated using one of several different algorithms and each relational algebra expression can be evaluated in many ways.

** An **evaluation plan** is an annotated expression/query tree specifying the execution strategy for a query.

EXAMPLE 1
Consider the following SQL solution and relational algebra translation

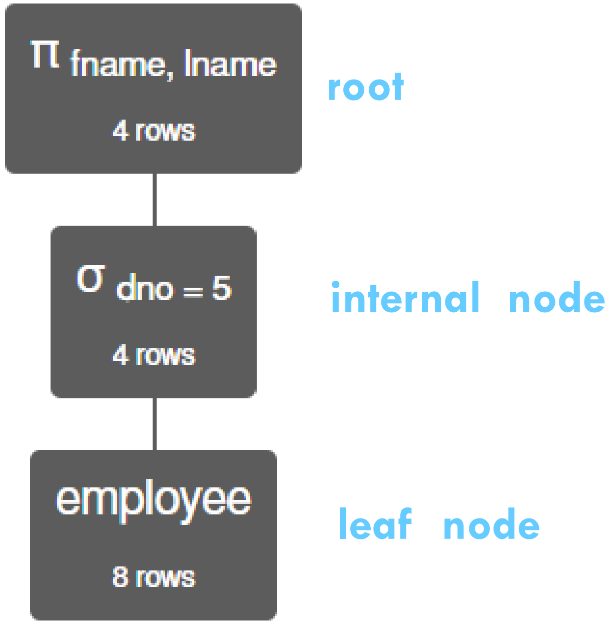
```
SELECT fname, lname
FROM   employee
WHERE  dno = 5;
```

$\pi_{\text{fname, lname}}(\sigma_{\text{dno = 5}}\ \text{employee})$

Query tree representation

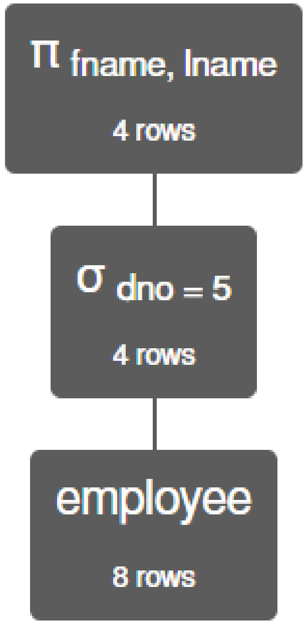
```
SELECT fname, lname
FROM   employee
WHERE  dno = 5;
```

$\pi_{fname,\ lname}(\sigma_{dno\ =\ 5}\ employee)$



employee.fname	employee.lname
'John'	'Smith'
'Franklin'	'Wong'
'Ramesh'	'Narayan'
'Joyce'	'English'

Query tree representation with evaluation plan

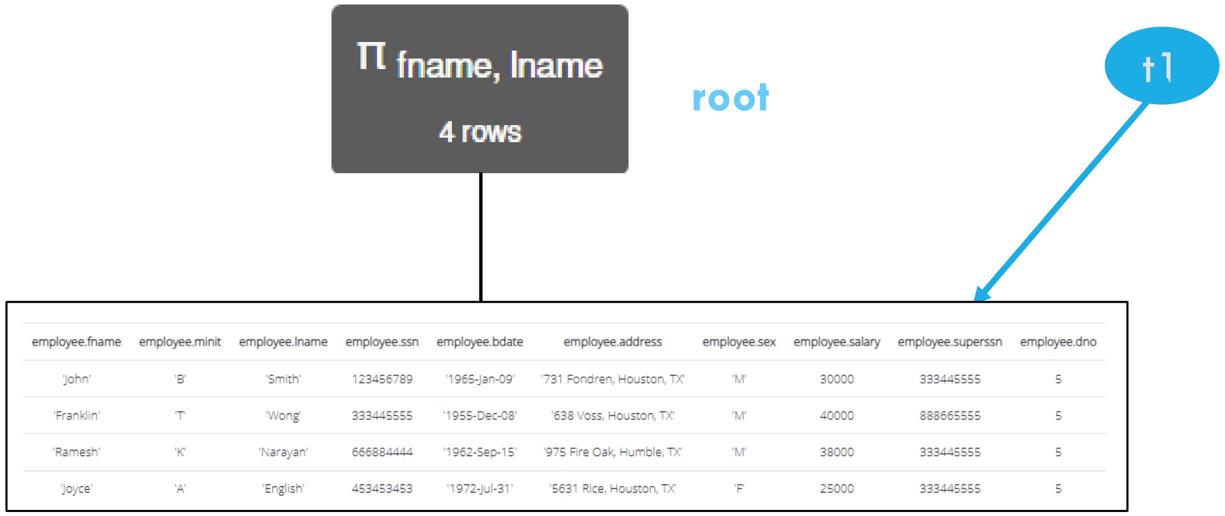


for each tuple in t1 retrieve fname, lname

linear search on condition. -write to t1

file scan: employee

How materialization evaluation works ...



employee.fname	employee.minit	employee.lname	employee.ssn	employee.bdate	employee.address	employee.sex	employee.salary	employee.superssn	employee.dno
'John'	'B'	'Smith'	123456789	'1965-Jan-09'	'731 Fondren, Houston, TX'	'M'	30000	333445555	5
'Franklin'	'T'	'Wong'	333445555	'1955-Dec-08'	'638 Voss, Houston, TX'	'M'	40000	888665555	5
'Ramesh'	'K'	'Narayan'	666884444	'1962-Sep-15'	'975 Fire Oak, Humble, TX'	'M'	38000	333445555	5
'Joyce'	'A'	'English'	453453453	'1972-Jul-31'	'5631 Rice, Houston, TX'	'F'	25000	333445555	5

Example 2

UBIK database

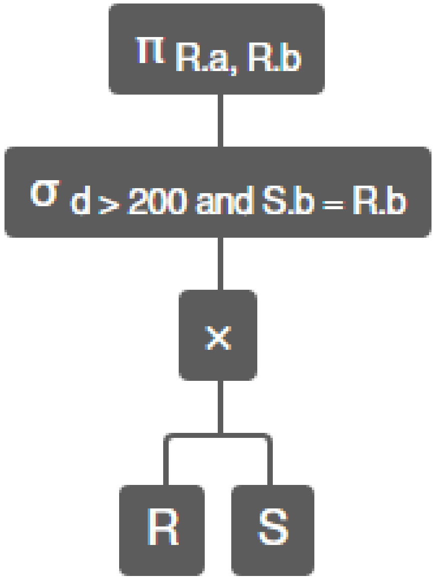
https://dbis-uibk.github.io/relax/calc/local/uibk/local/0

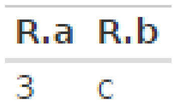
Consider the following SQL query:

```
SELECT  R.a, R.b
FROM    R, S
WHERE d > 200 AND S.b=R.b
```

And the relational algebra translation:

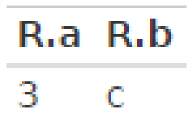
$$\pi_{R.a,\ R.b}\ \sigma_{d\ >\ 200\ \text{and}\ S.b\ =\ R.b}\ R \times S$$





R.a	R.b
3	c

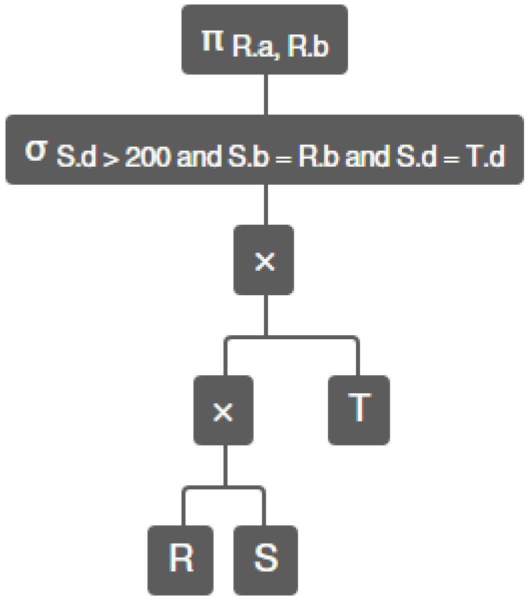
Example 3 UBIK database



R.a	R.b
3	c

Consider the following SQL query:

```
SELECT R.a, R.b
FROM  R, S, T
WHERE S.d > 200 AND
        S.b=R.b   AND
        S.d = T.d
```



$\pi_{R.a,\ R.b}$
$\sigma_{S.d > 200 \text{ and } S.b = R.b \text{ and } S.d = T.d}$
×
×
T
R
S

And the relational algebra translation:

$$\pi_{R.a,\ R.b}\ \sigma_{S.d > 200 \text{ and } S.b = R.b \text{ and } S.d = T.d}\ R \times S \times T$$

EXAMPLE 4:
Translating SELECT FROM WHERE (with no subqueries) to Relational Algebra

Given a general SELECT statement of the form:

SELECT *attributeList*

FROM R1 INNER JOIN R2 ON *joinCondition*

WHERE *condition*

translates to:

$$\pi_{\texttt{attributeList}}\ (\sigma_{\texttt{condition}}(\texttt{R1 JOIN}_{\texttt{joinCondition}}\ \texttt{R2}))$$

NOTE: An SQL statement may have many equivalent relational algebra expressions.

Example 5: Consider the following (Company Schema):

List all salaries greater than 50000

The SQL solution:

```
SELECT salary
FROM  employee
WHERE salary > 50000;
```

Translating this SQL to Relational Algebra

```
SELECT salary

FROM employee

WHERE salary >50000;
```

Option 1:

$\pi_{\texttt{salary}} (\sigma_{\texttt{(salary> 50000)}} \texttt{employee}))$

retrieve tuples with salary > 50000

retrieve salary column

Option 2:

$\sigma_{\texttt{(salary > 50000)}} (\pi_{\texttt{salary}} \texttt{employee})$

retrieve salary column

retrieve tuples with salary > 50000

DIFFERENCES BETWEEN THESE?

$\pi_{\texttt{salary}}\ (\sigma_{\texttt{(salary> 50000)}}\ \texttt{employee))}$

$\sigma_{\texttt{(salary > 50000)}}\ (\pi_{\texttt{salary}}\ \texttt{employee)}$

EXAMPLE 6:

Given the following problem based on the Company schema write the associated SQL code (using joins), a correct relational algebra expression translation and a query tree representing the relational algebra expression:

List the names of all employees who work on projects located in Stafford

EXAMPLE 7:

Given the following problem based on the Company schema write the associated SQL code (using joins), a correct relational algebra expression translation and a query tree representing the relational algebra expression:

List the location of all departments managed by manager Franklin Wong

ISSUES TO CONSIDER WITH QUERY TREES:

- Size of temporary tables
- Algorithms used for execution plan

OPTIMISATION

- Different query trees for a given query can have different *costs*
- Different evaluation plans for a given query can have different *costs*
- Optimisation techniques attempt to choose the best among a number of potential query trees

APPROACH 1:
Compare cost estimates across different solutions

- Cost is usually measured as the total elapsed time for answering a query
- One approach is to calculate cost estimates for each possible query tree
- The query tree with the lowest cost estimate should then be chosen

How to calculate cost estimates?

Cost factors include CPU speed, disk access time, network communication time, etc.

Disk access is typically the predominant cost and can be measured by number of blocks read/number of blocks written per query.

MAIN COST ESTIMATE USED:
Number of block transfers where each block contains a number of records

Number of blocks transferred from disk depends on:

- Size of buffer in main memory - having more memory reduces need for more disk accesses.
- Indexing structures used (primary, secondary, etc.)
- Whether all blocks of a file must be transferred or not
  - e.g., if search can be done on primary key of index file or on secondary index then only retrieve blocks that satisfy search condition

- As is typical in Computing, often use worst case estimates, knowing that any *actual* cost cannot exceed a worst case estimate.

DBMS CATALOG

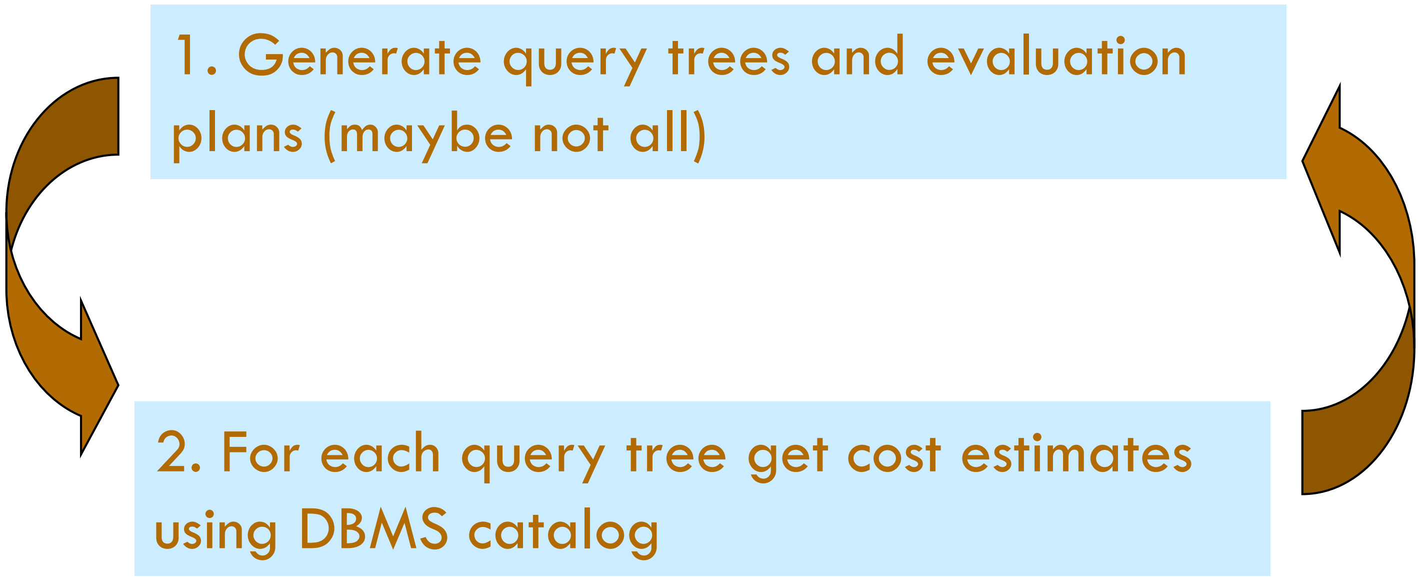
The DBMS catalog stores statistical information about each table such as table sizes, indexes (and their depths) etc.

The statistical information on the tables and attributes used in a query, can be found in the *DBMS catalog* and these are used to calculate cost estimates also.

In DBMS catalog, for each table R information is stored on:

- Number of tuples/records in table R
- Number of blocks containing tuples of table R
- Size of a record in bytes
- Blocking factor
- Information on number of distinct values per attribute and number of values that would satisfy set of equality operations on attribute (by having averages, min, max, etc.)
- Information on indices (index types, index field values, etc.)

STEPS FOR APPROACH 1



1. Generate query trees and evaluation plans (maybe not all)

2. For each query tree get cost estimates using DBMS catalog

Resulting in a set of cost estimates such that the best can be chosen and the query tree with the lowest cost estimate can then be picked as the single best query tree and evaluation plan.

THEREFORE:

To choose among plans, the optimiser has to estimate cost of each evaluation plan.

Two aspects to this:

For each node of tree:

- estimate cost of performing associated operation
- estimate size of result and if it is sorted

APPROACH 1: SUMMARY

- Cost-based optimisation, while good, is expensive:

  As query complexity increases so does the different number of query trees and plans possible and each query tree requires its own cost estimates

N.B. It is important that the amount of time an optimiser spends on calculating the best solution (optimising) is not longer than the amount of time which would elapse if executing a solution picked at random

APPROACH 2:
Heuristic Optimisation

- Optimiser often uses heuristics to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimisation transforms the query-tree by using a set of rules that typically (but not always) improve execution performance.
- Some cost based estimation is also performed – as part of the heuristic optimisation and to choose between a reduced set of trees and/or evaluation plans.

STEPS FOR APPROACH 2:

1. Create a *canonical query tree.*

2. Apply a *set of heuristics* to the tree to create a more efficient query tree.

3. Create cost estimates of this query tree, if appropriate, to ensure best evaluation plan.

DEFINITION:
Canonical query tree

A canonical query tree is an *inefficient* query tree representing relational algebra expressions which can be created directly from the SQL solution following a sequence of quick and easy steps:

- Uses CARTESIAN product instead of JOINS
- Keeps all conditions (σ) together in one internal node
- π becomes root node

Steps to create a canonical query tree with SELECT/FROM/WHERE clauses and no sub-queries:

1. All relations in FROM clause become leafs of the tree. They should be combined with a Cartesian product (x) of the relations.

* *Remember:* Only 2 relations can be involved in a Cartesian product at a time (binary tree)

2. All conditions in the WHERE clause and any JOIN conditions in WHERE or FROM clause become a sequence of relational algebra expressions in **one** inner node of the tree (with inputs from previous step)

3. All conditions from the SELECT clause become a relational algebra expression in the root node

EXAMPLE 8 *with implicit join*

List the names of employees in research department

```
SELECT fname, lname
FROM   employee, department
WHERE  dno = dnumber AND
       dname = ‘Research’;
```

Creating the **canonical query tree …**

EXAMPLE 8 *with explicit join*

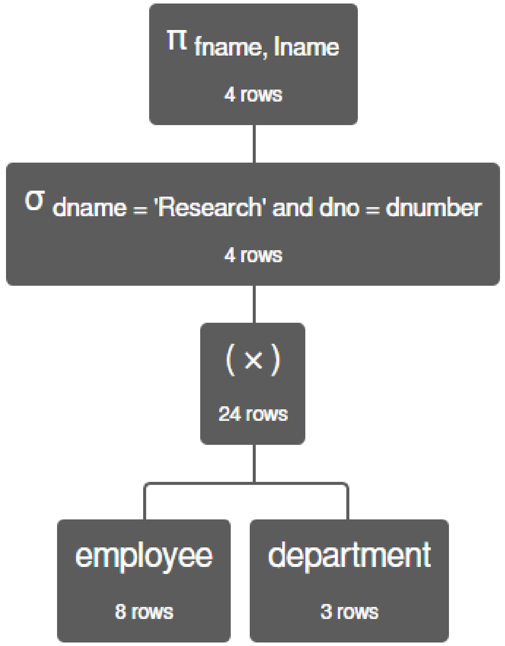
List the names of employees in research department

```
SELECT fname, lname
FROM   employee INNER JOIN department ON
       dno = dnumber
WHERE  dname = 'Research';
```

Creating the **canonical query tree** …

CANONICAL TREE REPRESENTATION:

```
SELECT   fname, lname
 FROM    employee INNER JOIN department
         ON dno = dnumber
 WHERE   dname = 'Research';
```



NOTE:

This would be very inefficient if executed directly because of the Cartesian product operations.

Recall Cartesian product:

R x S

Returns tuples comprising the concatenation of every tuple in R with every tuple in S

CONSIDER EXAMPLE 7 AGAIN

Draw the canonical query tree for the SQL query in Example 7:

List the location of all departments managed by manager Franklin Wong

HEURISTIC OPTIMISATION

Heuristic Optimisation **MUST** transform this canonical query tree into a final query tree that is efficient to execute:

- In general, heuristic optimisation tries to apply the most restrictive operators as early as possible in the tree (furthest down the tree) and to reduce the size of the temporary tables/results created that move "up" the tree.
- Heuristic Optimisation must include rules for equivalence among relational algebra expressions that can be applied to the initial tree.

HEURISTIC OPTIMISATION ALGORITHM:

Input: A canonical query tree

Process:

1. Decompose any σ with AND conditions into individual σ
2. Move each σ operator as far down the query tree as possible.
3. Rearrange the leaf nodes so that most restrictive σ can be applied first (using information from DBMS catalog) and so that future JOINS are possible.

Note: "most restrictive" means those operators that result in relations with the fewest tuples or with the smallest absolute size - these operations should happen first – that is – at the lowest level of the tree and on the left hand side of the tree.

4. Combine CARTESIAN PRODUCT operators with σ (sigma) to form JOIN operators where appropriate (replacing all x)

5. Decompose π and move each π as far down the tree as possible, possibly creating new π operators in the process.

(6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.)

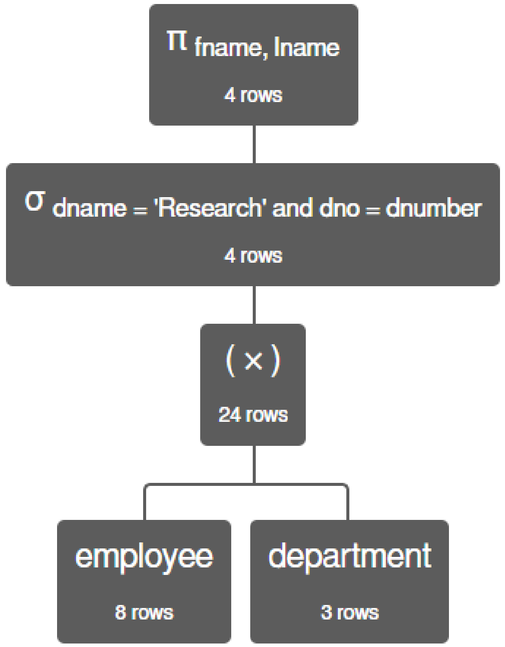
(7. Add evaluation plan)

Output: An efficient query tree

Back to EXAMPLE 8:

List the names of employees in research department

```
SELECT fname, lname
FROM   employee INNER JOIN department
       ON dno = dnumber
WHERE  dname = 'Research';
```

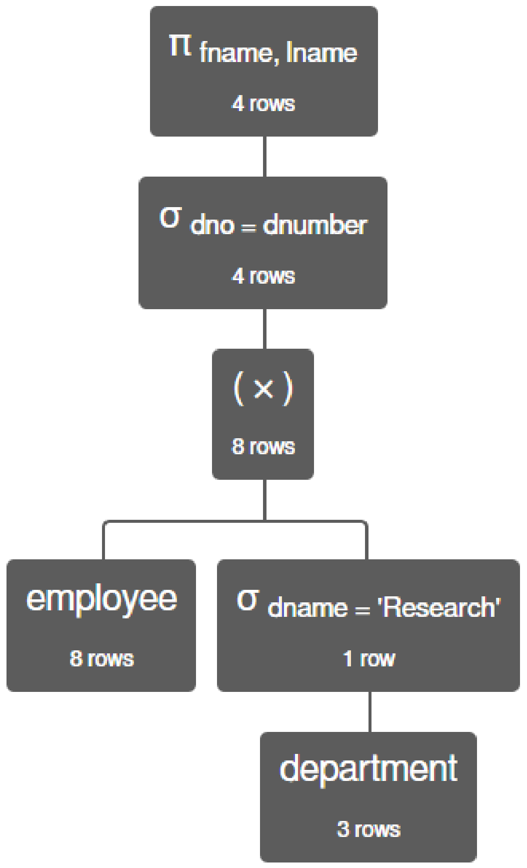


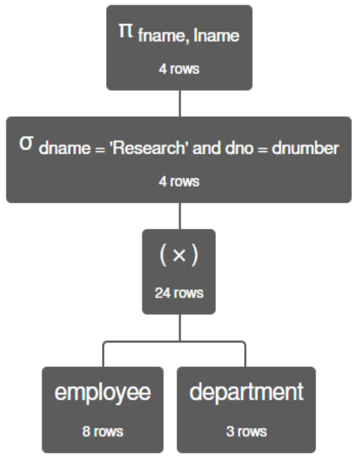
OPTIMISATION HEURISTIC 1 & 2: Decompose conditions and apply sigma (σ) operators as early as possible

- **"Move σ down tree"** thus eliminating <u>unwanted tuples.</u>
- Heuristic 1 tries to reduce the size of the tables to be combined as much as possible:
- Therefore, if a selection operator (σ) occurs *after* a Cartesian product or a join, check to see if it can occur *before* these operations

Example 8:

Move (σ) sigma



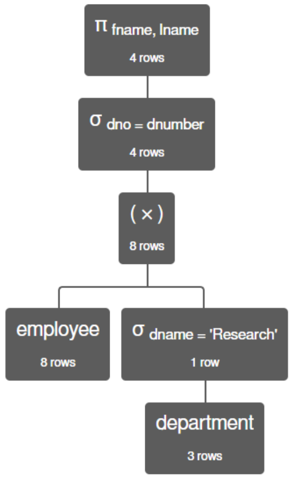


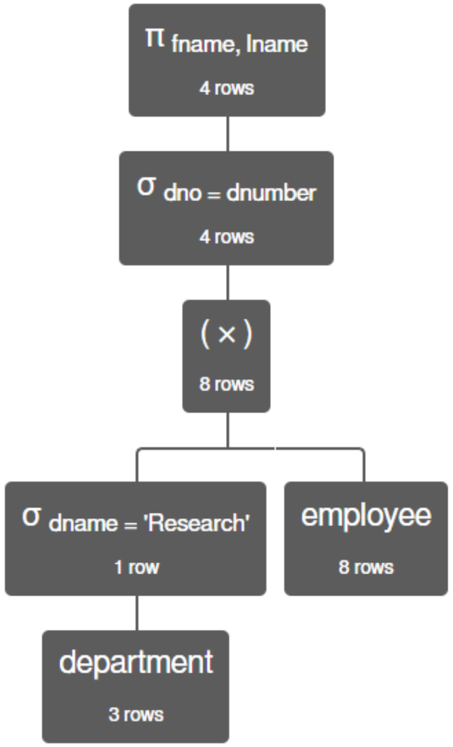
OPTIMISATION HEURISTIC 3:
Rearrange the leaf nodes so that most restrictive sigma opeartors can be applied first

If we don't have any information from DBMS catalog

- we might leave nodes as they are
- Use database schema (number of columns) to make a good estimate
- Use sample data (number of rows) and database schema (number of columns) to make a good estimate

EXAMPLE 8:
REARRANGE LEAF NODES





OPTIMISATION HEURISTIC 4:
Replace Cartesian product (x) and appropriate selects (σ) with JOIN

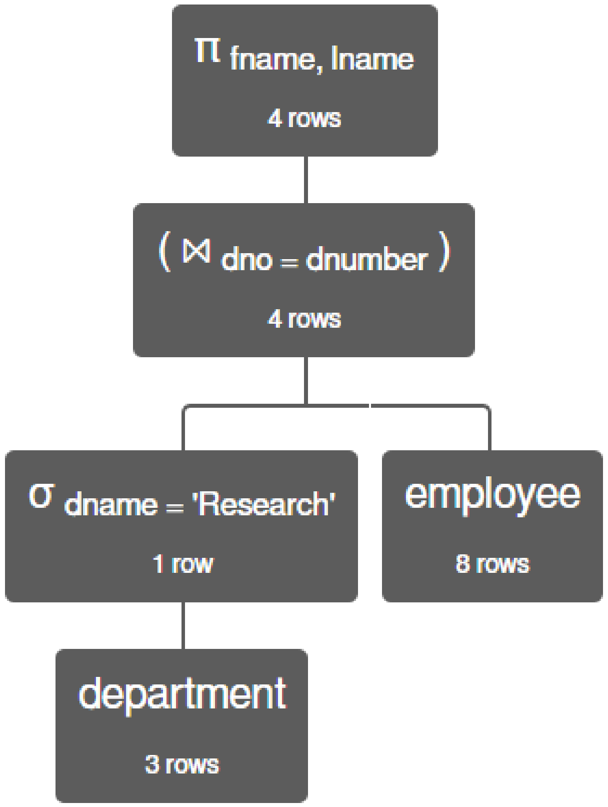
* First must ensure the leaf nodes are ordered such that this can happen – if not re-order leaf nodes and ensure to keep any select operators with the appropriate leaf node

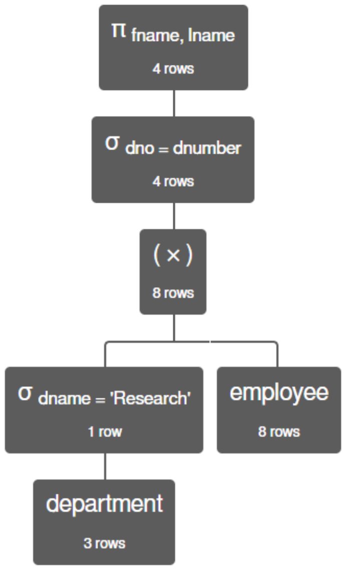
$\sigma_{\texttt{condition}}\texttt{(r1 X r2)}$

Is equivalent to:

$\texttt{R1 JOIN}_{\texttt{condition}}\texttt{ R1}$

EXAMPLE 8:
REPLACE X





OPTIMISATION HEURISTIC 5: Apply Pi (π) operators as early as possible

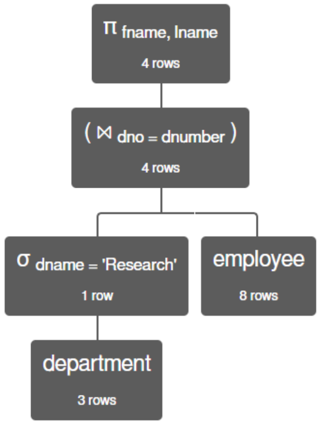
- **Motivation: "Move π down the tree"** (project) to eliminate unwanted columns
- The heuristic ensures that the size of the tables to be joined are as small as possible (reduces number of attributes/columns)

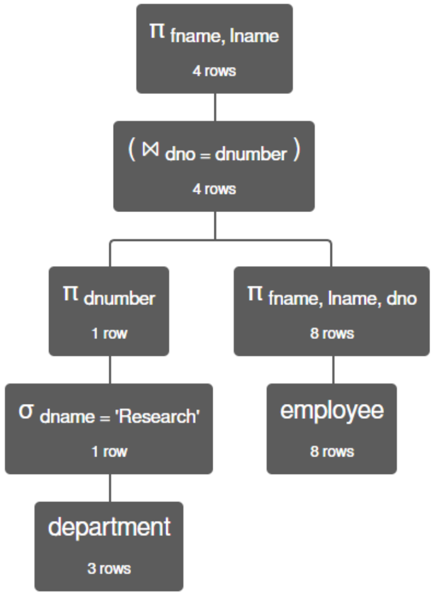
Therefore:

- for each π check if that π can be carried out before the join
- for each table check if additional π can be introduced (these may not be stated explicitly in the query)

N.B. MUST ensure that all needed columns further up in the tree are retained (even if they are not immediately necessary)

EXAMPLE 8:
Move Pi





$\pi_{\text{fname, lname}}$ ((π_{dnumber} $\sigma_{\text{dname = 'Research'}}$ department) $\bowtie_{\text{dno = dnumber}}$ $\pi_{\text{fname, lname, dno}}$ employee)

EXAMPLE 9

Using the COMPANY relational schema and interpretation as defined in lectures develop an SQL query to satisfy the following information need:

"List the names of employees with salaries greater than 30000, who work on projects for greater than 25 hours where the projects are located in Houston or Bellaire"

Using query optimisation heuristics develop a query tree which represents an efficient evaluation strategy for the developed query.

SQL SOLUTION:

```
SELECT      fname, minit, lname
FROM        project, employee, works_on
WHERE       pno = pnumber AND essn = ssn AND
            hours > 25 AND salary > 30000 AND
            (plocation = ‘Houston’ OR
            plocation = ‘Bellaire’);
```

CANONICAL QUERY TREE SOLUTION

π fname, minit, lname

  (σ pno = pnumber AND essn = ssn AND

    hours > 25 AND salary > 30000 AND

    plocation = 'Houston' OR plocation = 'Bellaire'

  (project x employee x works_on)

)

OPTIMISATION HEURISTIC 1 & 2:
Decompose conditions and apply sigma (σ) operators as early as possible

OPTIMISATION HEURISTIC 3:
Rearrange the leaf nodes so that most restrictive sigma opeartors can be applied first and that future joins can be performed

OPTIMISATION HEURISTIC 4:
Replace Cartesian product (x) and appropriate selects (σ) with JOIN

OPTIMISATION HEURISTIC 5:
Apply Pi (π) operators as early as possible

EXAMPLE 10: (Winter 2017)

(Given the movie schema from the exam paper)

(c) Using joins, create a SQL query to answer the following information need. Using this SQL query, create a canonical query tree, explaining the steps you take in creating the tree and highlighting what parts of the SQL query are represented by the root, leaves and inner nodes of the tree.

For movies of genre 'Sci-Fi', released in 2016 or 2017, with an average rating greater than 7, list the movie title, movie category and the names of the actors who star in the movie.

(d) Using the canonical query tree from part (c), and with respect to *heuristic-based optimisation*, develop a query tree that represents an efficient evaluation strategy for the SQL query. Explain the steps taken, describing each heuristic used.

SCHEMA:

movie(id, title, relYear, category, runTime, director, studioName, description, rating)

actor(aID, fName, surname, gender)

stars(movieID, actorID)

movGenre(movieID, genre)

For movies of genre 'Sci-Fi', released in 2016 or 2017, with an average rating greater than 7, list the movie title, movie category and the names of the actors who star in the movie.

SQL SOLUTION:
(Note: can use implicit or explicit joins)

```
SELECT title, category, fname, surname
FROM   movie INNER JOIN movGenre ON id = movieGenre.movieID
       INNER JOIN stars ON id = stars.movieID
       INNER JOIN actor ON aid = actorID
WHERE  genre = 'Sci Fi' AND
       rating > 7 AND
       (relYear = 2016 OR relYear = 2017);
```

SUMMARY: IMPORTANT TO KNOW

- Basic relational algebra operators.
- Mapping between relational algebra operators and SQL.
- Mapping between relational algebra expression and query tree.
- Mapping from SQL to Canonical Query tree.
- Heuristic optimisation steps to map Canonical Query tree to efficient query tree.
- *N.B. Do not mix up SQL code and Relational Algebra expressions*