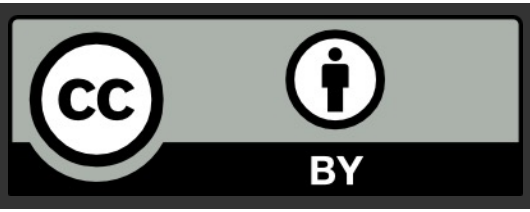
Query Optimization

F28DM Database Management Systems

Matthew P Aylett
m.aylett@hw.ac.uk

Materials from: Alasdair G J Gray

Materials released under CC-BY License



You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

Revision Quiz

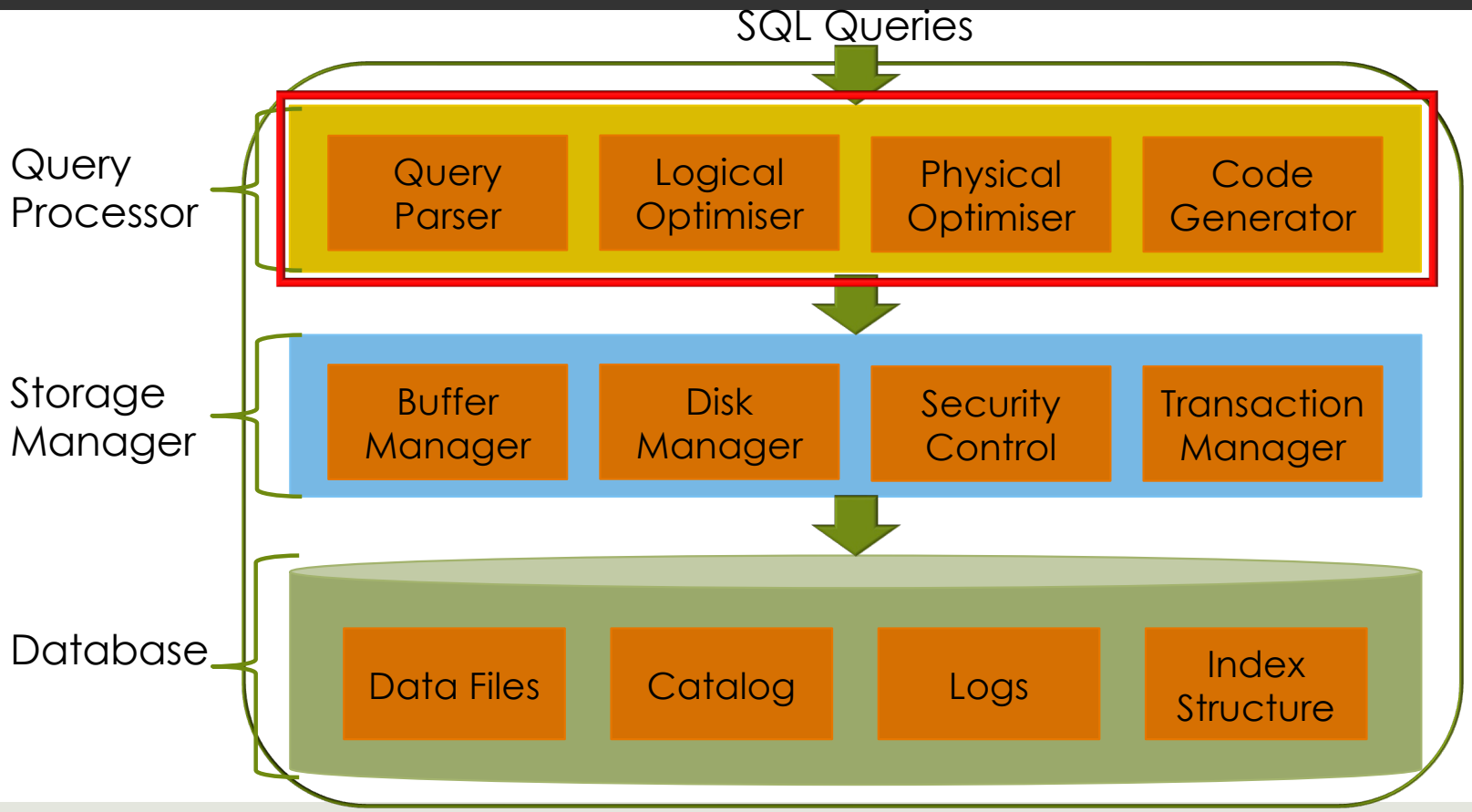
```
SELECT E.lastname, E.firstnames, E.salary
FROM   DBEmployee as E,
       DBDepartment as D,
       DBLocation as L
WHERE  E.empdNum = D.dNum AND
       D.dNum = L.ldNum AND
       L.loc = ‘Riccarton’;
```

- What is the meaning of this query?

Topic: Query execution and optimisation

- Query execution plans
- Query decomposition: Relational algebra
- Query optimisation: Logical and physical plans
- Cost estimation of plans

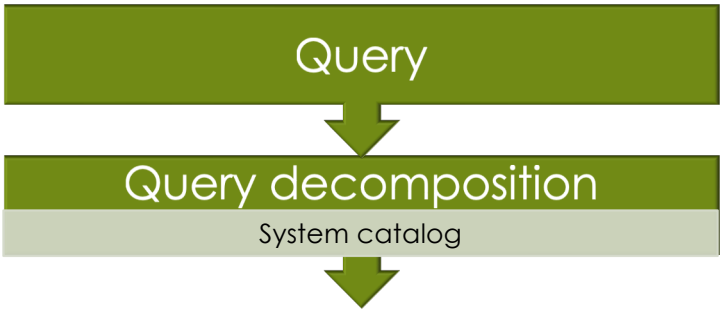
RDBMS Architecture

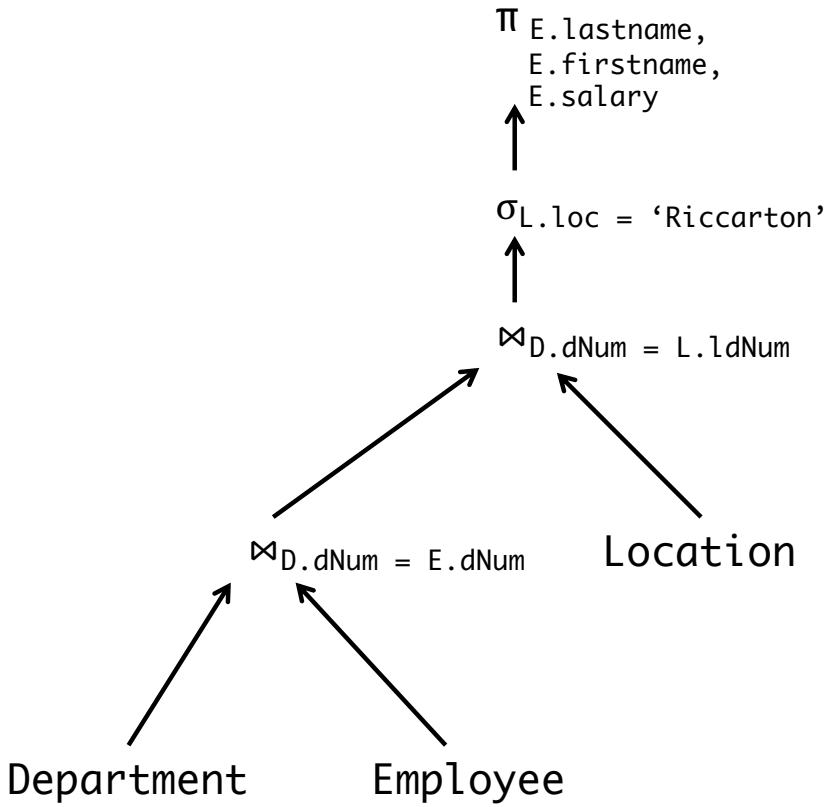


Overview of Query Processing

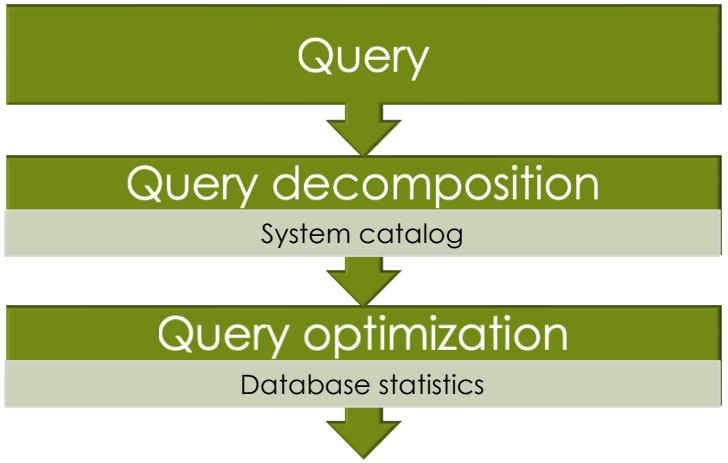

```
SELECT E.lastname, E.firstname,
       E.salary
FROM   DBEmployee as E,
       DBDepartment as D,
       DBLocation as L
WHERE  E.empdNum = D.dNum AND
       D.dNum = L.ldNum AND
       L.loc = 'Riccarton';
```

Overview of Query Processing

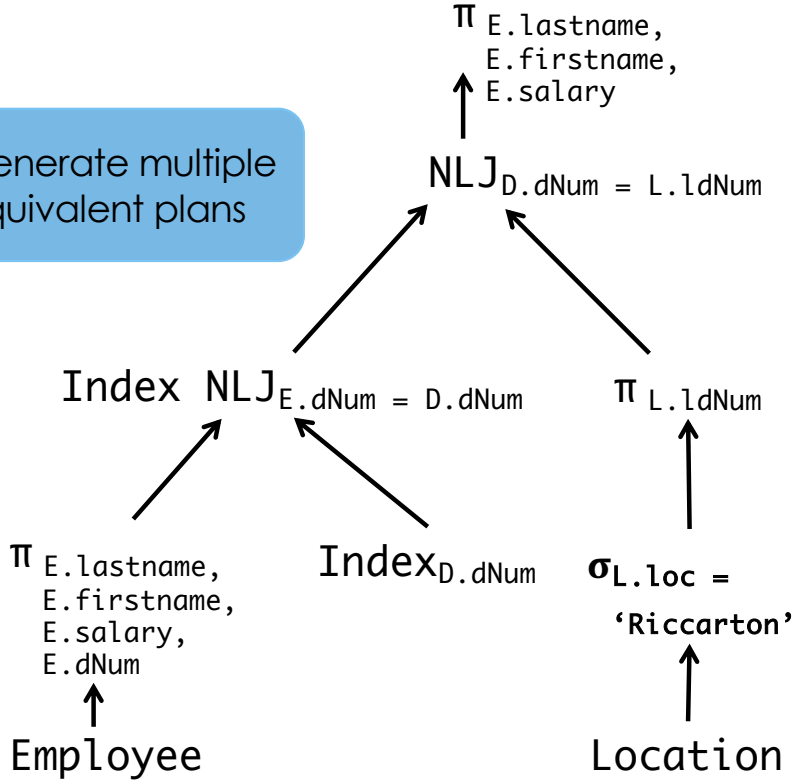




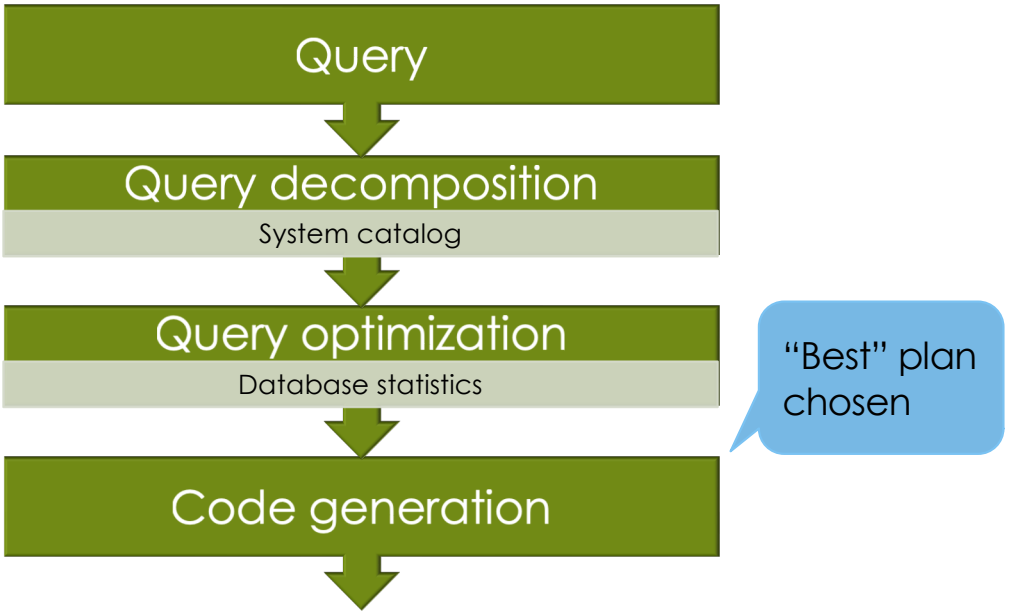
Overview of Query Processing

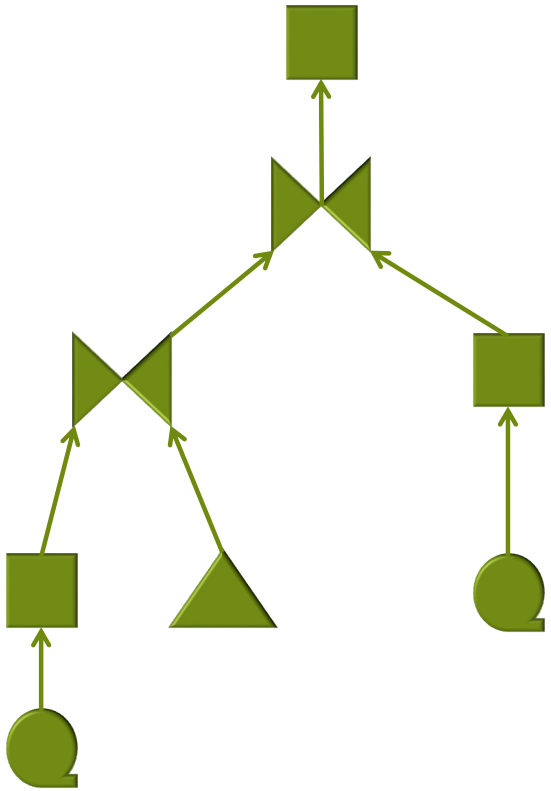


Generate multiple equivalent plans

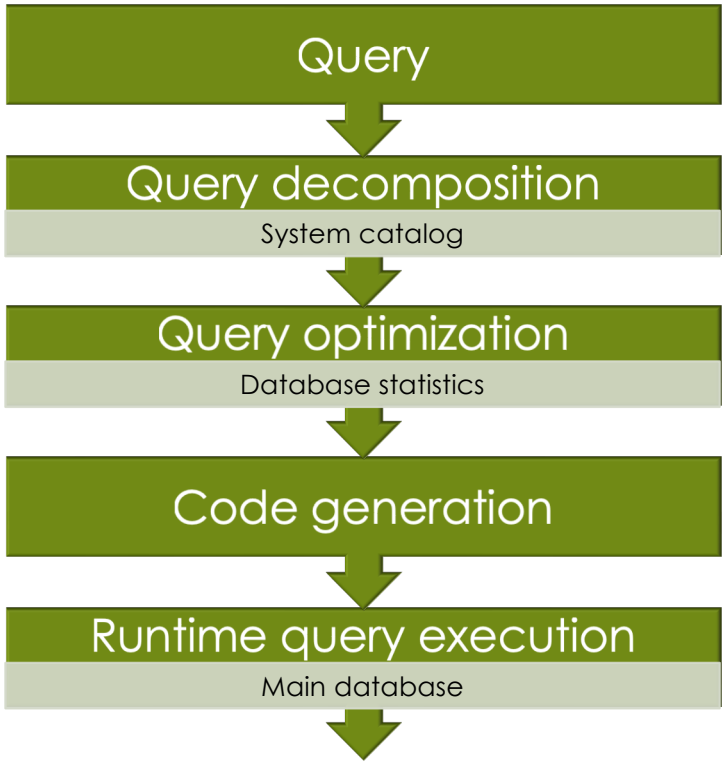


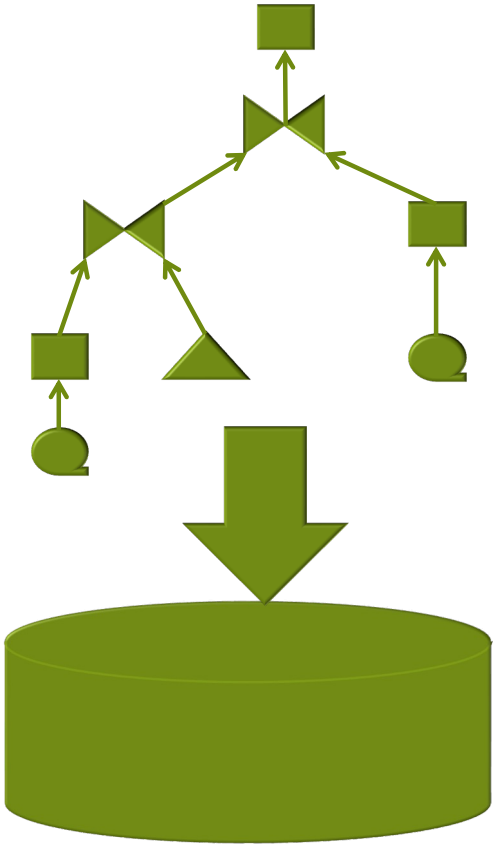
Overview of Query Processing



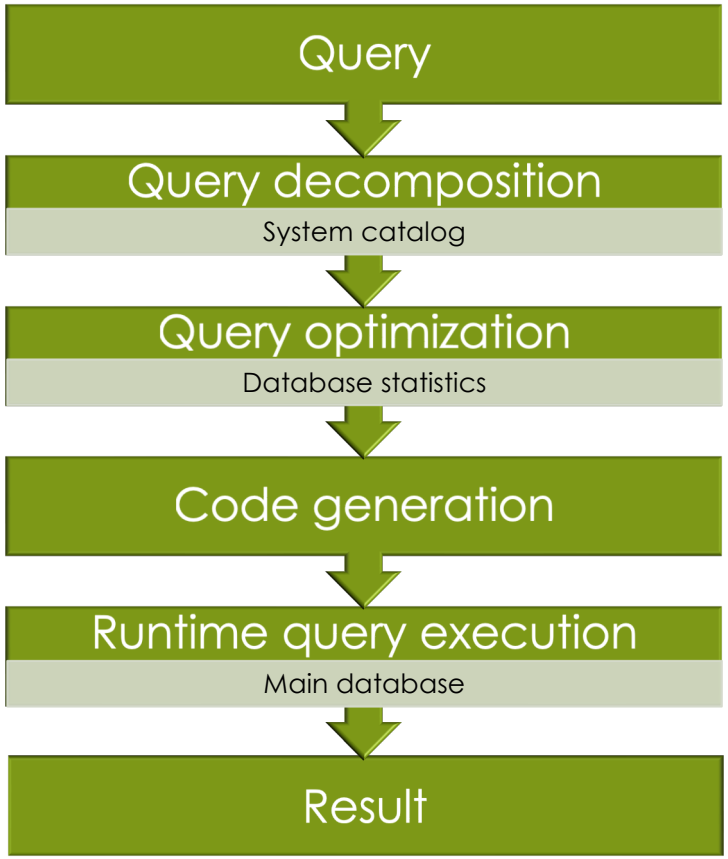


Overview of Query Processing





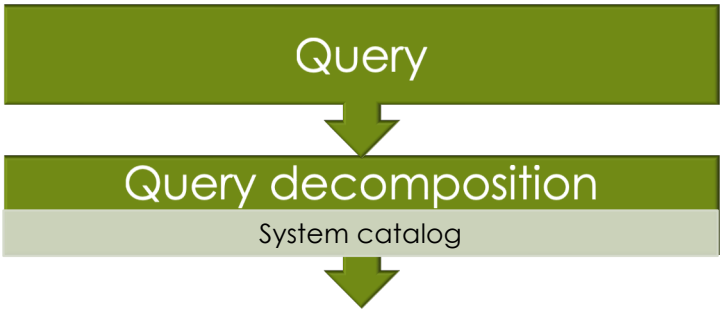
Overview of Query Processing

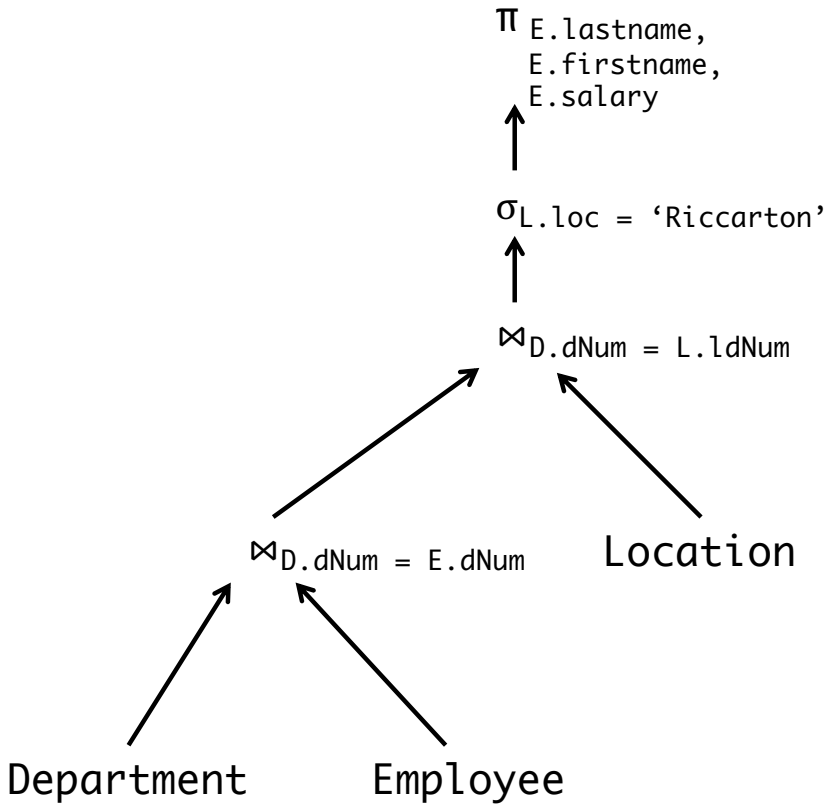


Query Optimisation: "Best" Plan

- Different optimisation goals
  - Execution time:
    how fast can we return the results
  - Initial response time:
    minimise the time taken to return the first result
  - Overall system throughput
  - Minimise resource usage
    reduce financial cost (cloud)
- How hard is the optimisation problem?
  - Intractable
- Search solution space
  - Rule based: Heuristics
  - Cost estimation:
    - Memory usage
    - Disk usage
    - Selectivity

Query Decomposition





SQL and Relational Algebra

- SQL
  - Declarative description of data need
  - Programming language
- Relational Algebra
  - Mathematical operators
  - Theoretical foundation
  - Enable data manipulation

Select operator: σ

- Selects a subset of rows from a relation
  - Expressed as a logical expression
    - AND, OR, NOT
  - SQL WHERE clause

```
SELECT *
FROM Staff
WHERE position = 'Manager'
  AND branchNo <> 2
```

Example

- *Select the managers not in branch 2*

$\sigma_{\text{position = 'Manager' AND branchNo} \neq 2}(\text{Staff})$

Staff				
staffNum	**name**	**salary**	**position**	**branchNo**
22	James	22,000	Manager	1
31	Mary	35,000	Manager	2
42	Jack	18,000	Cleaner	3
58	Susan	40,000	IT staff	1

Project operator: π

- Removes unwanted columns
  - Comma separate list of columns to keep
  - SQL SELECT clause

Example

- *Return the names and positions of staff members*

$\Pi_{name,\ position}$ (Staff)

```
SELECT name, position
FROM Staff
```

Staff				
staffNum	**name**	**salary**	**position**	**branchNo**
22	James	22,000	Manager	1
31	Mary	35,000	Manager	2
42	Jack	18,000	Cleaner	3
58	Susan	40,000	IT staff	1

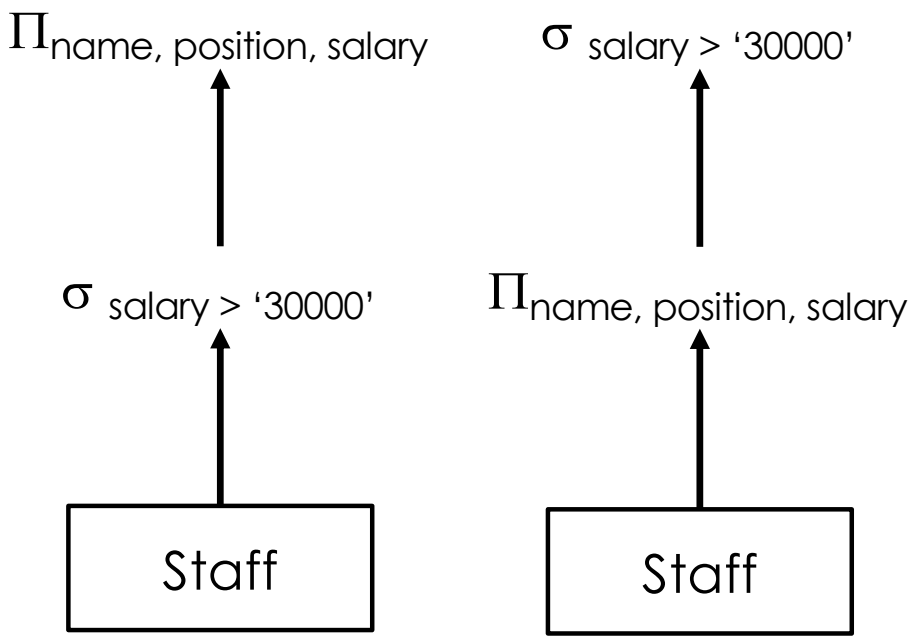
Composing operations/Operator Order

- Operators are **composable**
- Drawn as trees

Example

- *Find the name, position, and salary of staff earning more than 30,000*
  - $\Pi_{\text{name, position, salary}}(\sigma_{\text{salary > '30000'}}(\text{Staff}))$

Equivalent

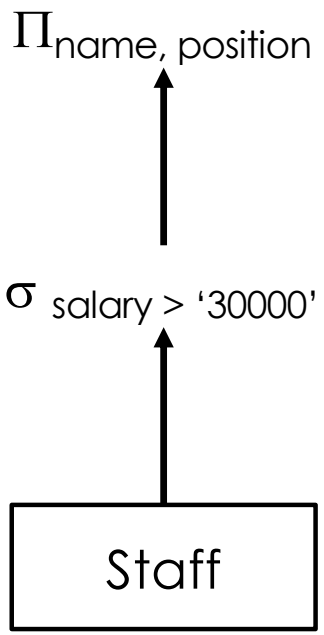


Composing operations/Not reorderable

- Operators **composable**
- Drawn as trees

Example

- *Find the names and positions of staff earning more than 30,000*
  - $\Pi_{name,\ position}\ (\sigma_{\ salary\ >\ '30000'}\ (Staff))$

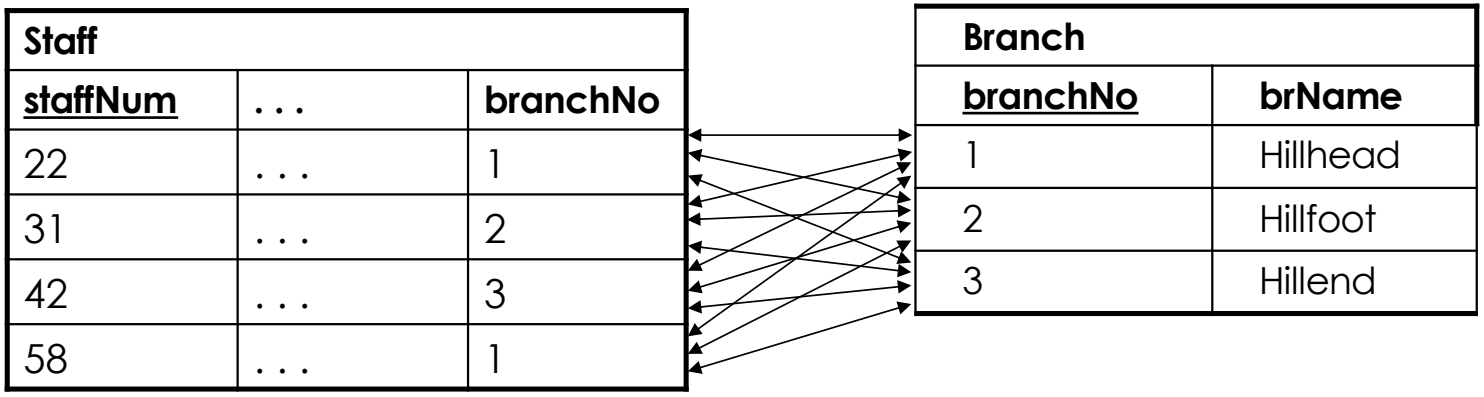


Cartesian Product: X

- Creates one table consisting of **each** row from one table **linked with each row** from the other table.

```
SELECT *
FROM Staff, Branch
```

Staff X Branch



Staff		
staffNum	. . .	**branchNo**
22	. . .	1
31	. . .	2
42	. . .	3
58	. . .	1

Branch	
branchNo	**brName**
1	Hillhead
2	Hillfoot
3	Hillend

Cartesian Product: Staff X Branch

staffNum	. . .	branchNo	branchNo	brName
22	. . .	1	1	Hillhead
22	. . .	1	2	Hillfoot
22	. . .	1	3	Hillend
31	. . .	2	1	Hillhead
31	. . .	2	2	Hillfoot
31	. . .	2	3	Hillend
42	. . .	3	1	Hillhead
42	. . .	3	2	Hillfoot
42	. . .	3	3	Hillend
58	. . .	1	1	Hillhead
58	. . .	1	2	Hillfoot
58	. . .	1	3	Hillend

Join ⋈

- Relates data from two relations
  - Foreign key relationship *(not necessarily)*
  - Eliminates duplicate columns

```
SELECT *
FROM Staff AS S, Branch AS B
WHERE S.branchNo = B.branchNo
```

Example

- *Join the staff and branch relations on the branch number*

Staff ⋈ $_{\text{Staff.branchNo = Branch.branchNo}}$ Branch

staffNum	**. . .**	**branchNo**	**brName**
22	. . .	1	Hillhead
31	. . .	2	Hillfoot
42	. . .	3	Hillend
58	. . .	1	Hillhead

S-P-J Example

- Find the name of the manager of the Hillhead branch

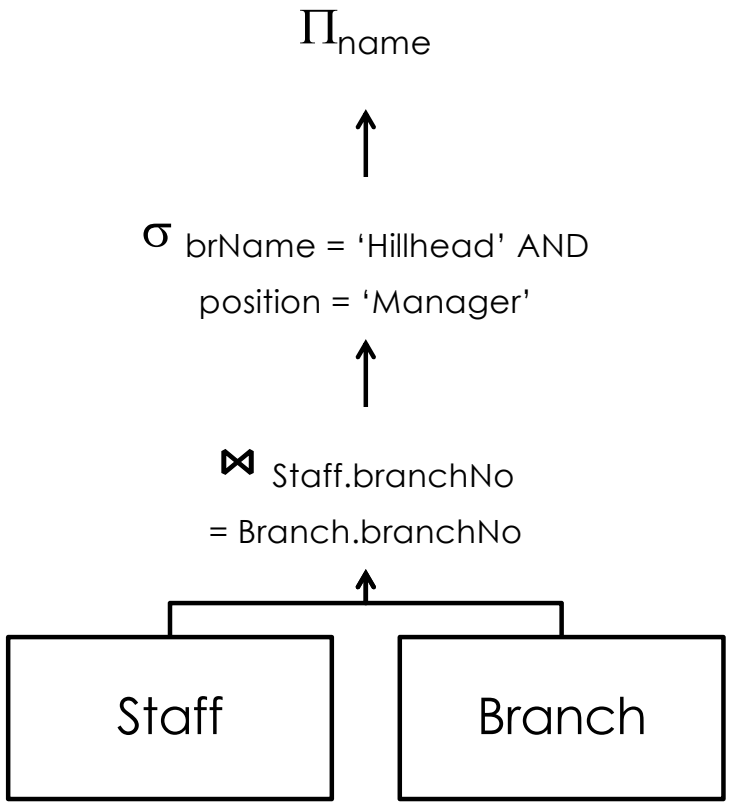
```
SELECT name
FROM Staff AS S, Branch AS B
WHERE S.branchNo = B.branchNo
  AND position = 'Manager'
  AND brName = `Hillhead'
```

Π_{name} (σ brName = 'Hillhead' AND position = 'Manager'
(Staff $\bowtie$ Staff.branchNo = Branch.branchNo Branch))

staffNum	name	salary	position	branchNo	brName
22	James	22,000	Manager	1	Hillhead
31	Mary	35,000	Manager	2	Hillfoot
42	Jack	18,000	Cleaner	3	Hillend
58	Susan	40,000	IT staff	1	Hillhead

Transform to query tree

- Leaf node created for each base relation.
- Non-leaf node created for each intermediate relation produced by a relational algebra operation.
  - Only Join and Cartesian Product nodes can have 2 inputs
- Root of tree represents query result.
- Sequence is directed from leaves to root



Equivalence Relationships
(Show for background knowledge)

- Basics on selection
  - $\sigma_A(R) = \sigma_A \sigma_A(R)$
  - $\sigma_A \sigma_B(R) = \sigma_B \sigma_A(R)$
- Breaking up conditions
  - $\sigma_{A \vee B}(R) = \sigma_A(R) \cup \sigma_B(R)$
  - $\sigma_{A \wedge B}(R) = \sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$
- Selection and Cross Product
  - $\sigma_{A \wedge B \wedge C}(R \times S) = \sigma_A(\sigma_B(R) \times \sigma_C(S))$
    where B only contains attributes of R and
    C only contains attributes of S

Equivalence Relationships
(Show for background knowledge)

- Basics on projection
  - $\Pi_{a1,\ldots,an}(\Pi_{b1,\ldots,bn}(R)) = \Pi_{a1,\ldots,an}(R)$ where $\{a1,\ldots,an\} \subseteq \{b1,\ldots,bn\}$
- Selection and Projection
  - $\Pi_{a1,\ldots,an}(\sigma_A(R)) = \sigma_A(\Pi_{a1,\ldots,an}(R))$ where $A \subseteq \{a1,\ldots,an\}$

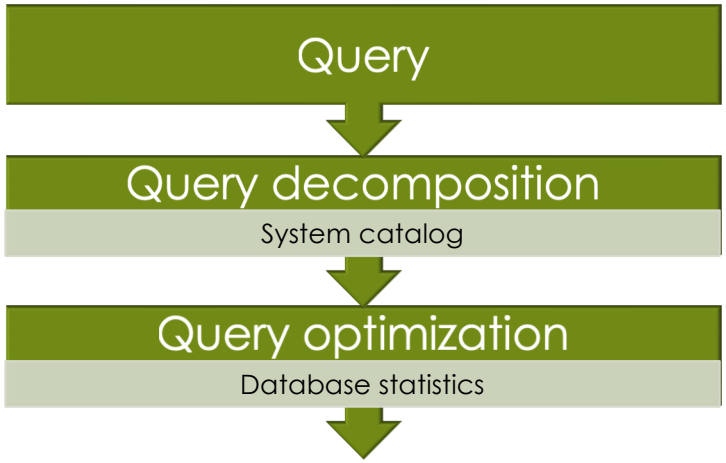
Exercises (for home)

Write SQL queries and then draw a query tree for:

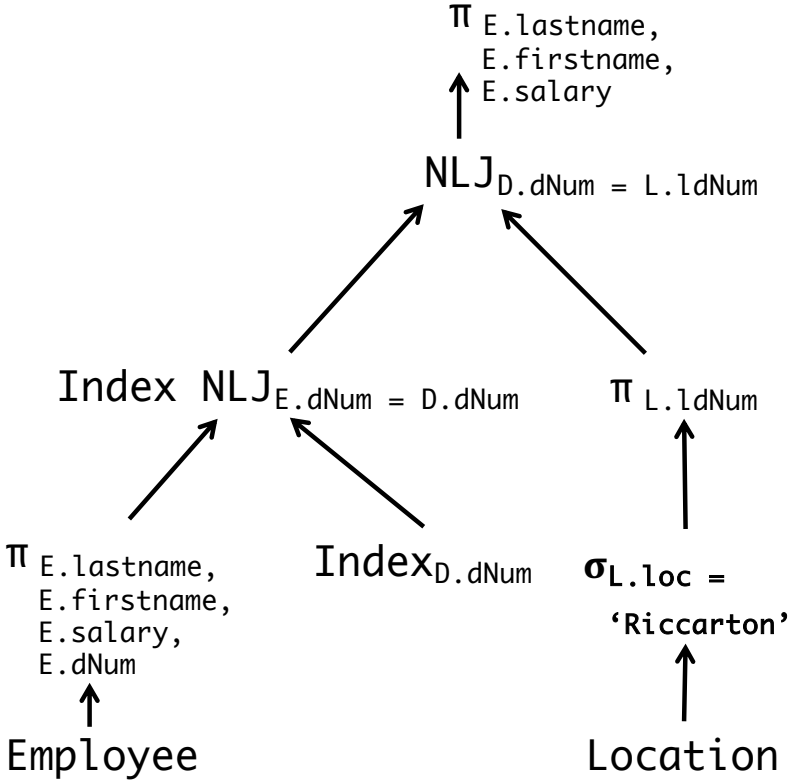
1. Find the name of the branch that Mary works at

2. Find the staff number, name and gender of the Hillend cleaner

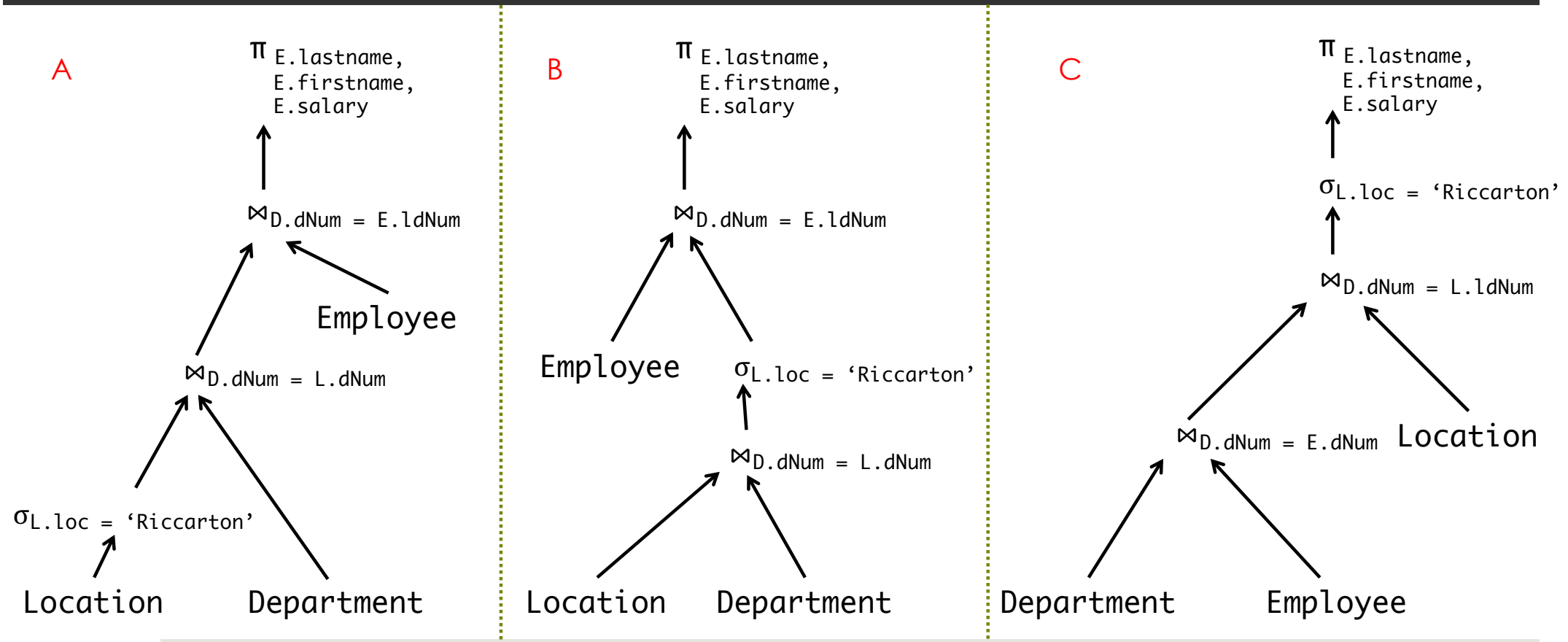
Query Optimisation



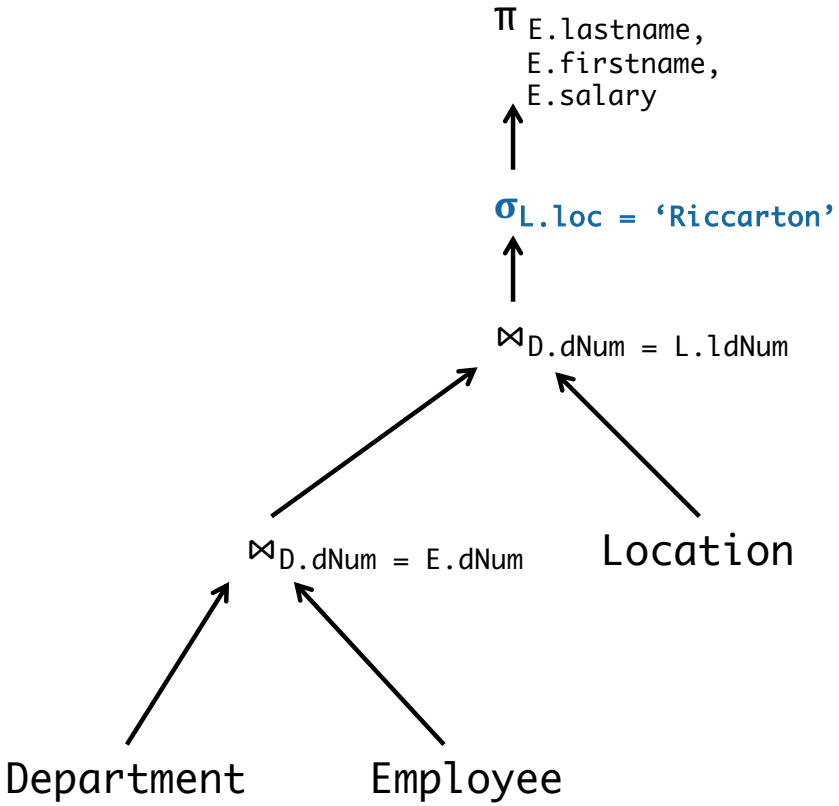
- Generate many alternative plans
- Cost up execution of plans
- Decide on one plan to use

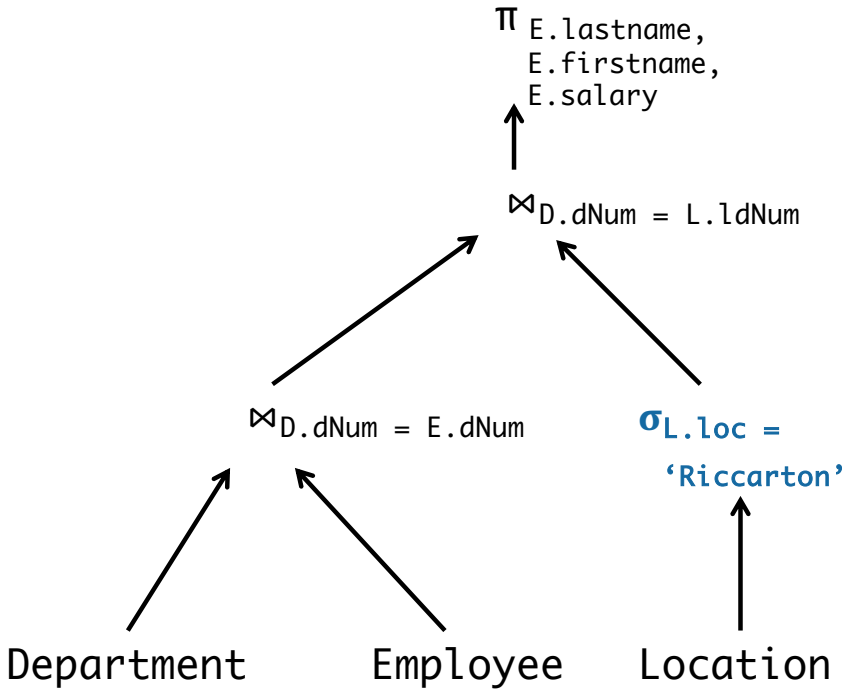


Are these query plans equivalent?



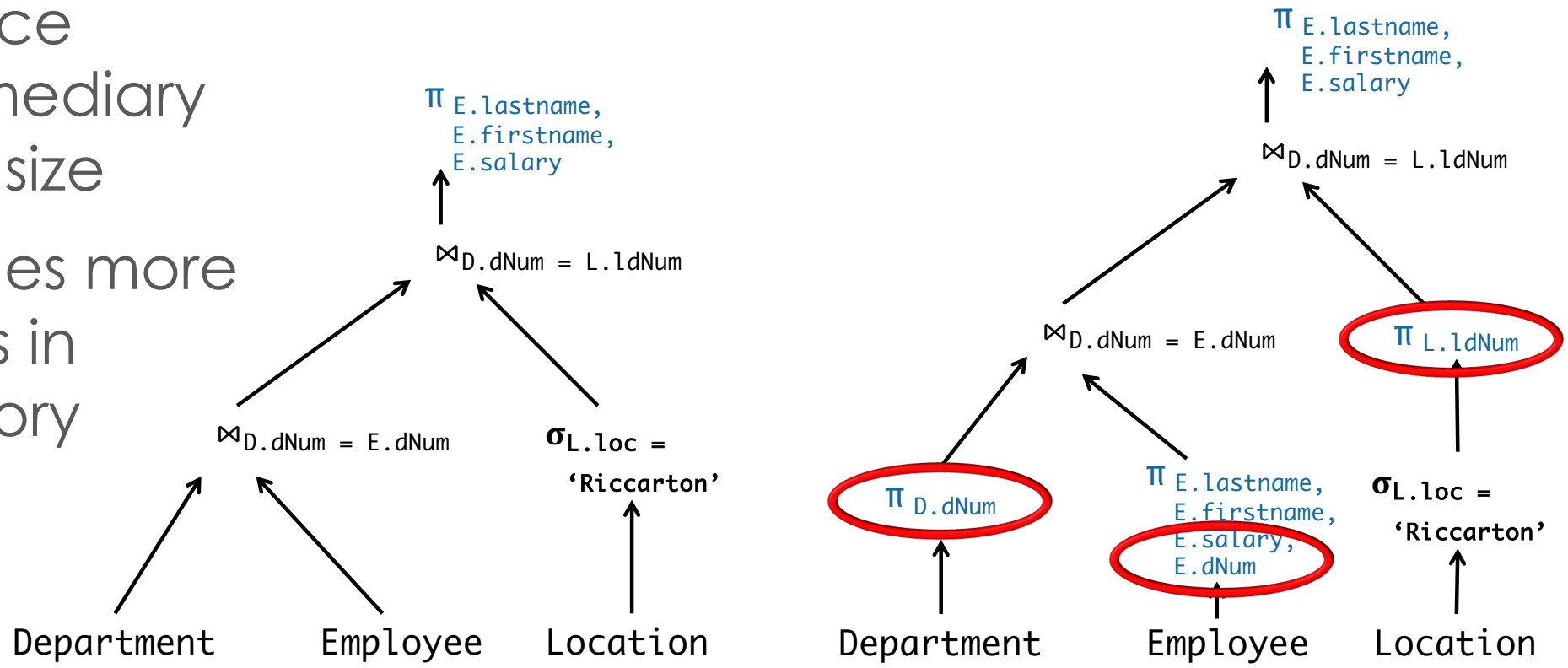
Push σ: Reduce intermediary result size





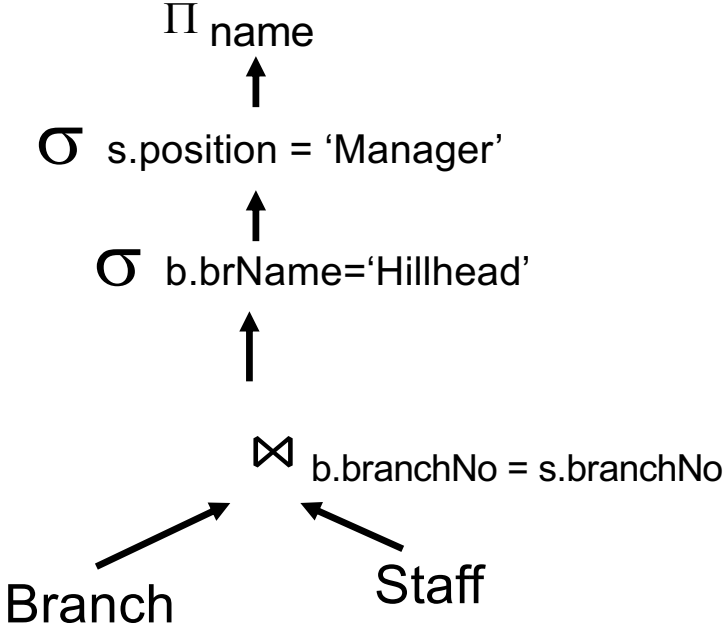
Push/Introduce π

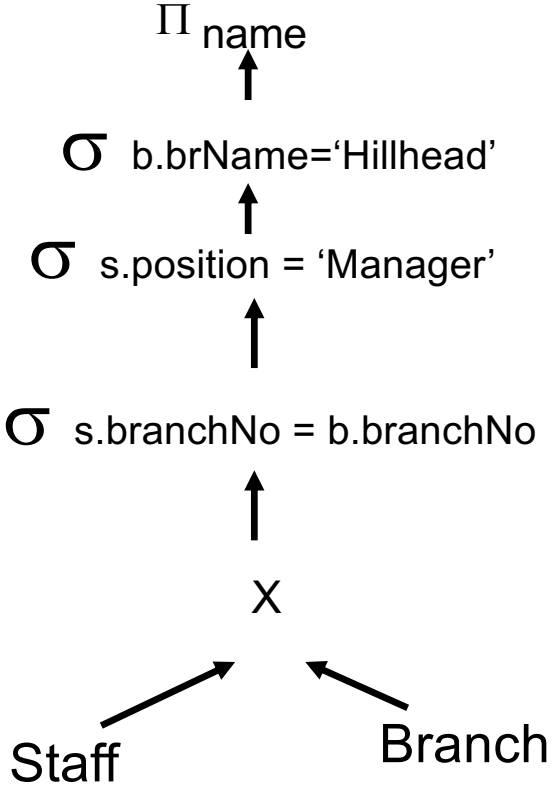
- Reduce intermediary result size
- Enables more results in memory



Equivalent trees

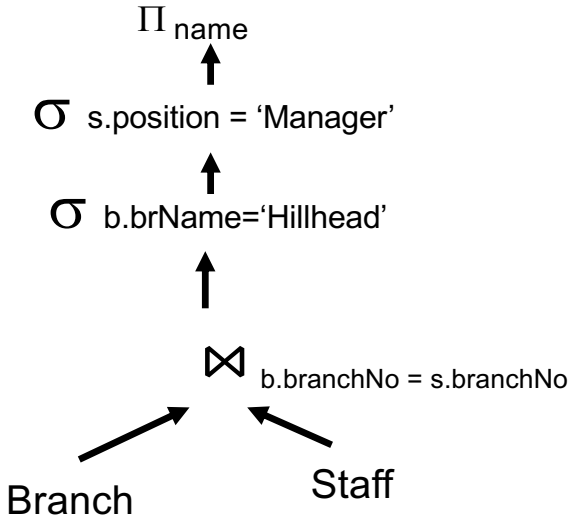
The left tree is more efficient, because the join result is smaller

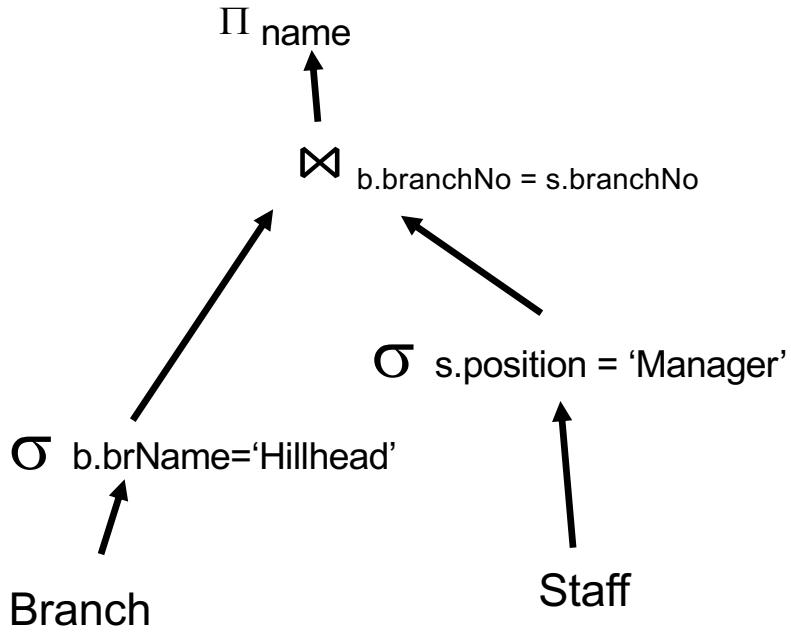




Equivalent trees

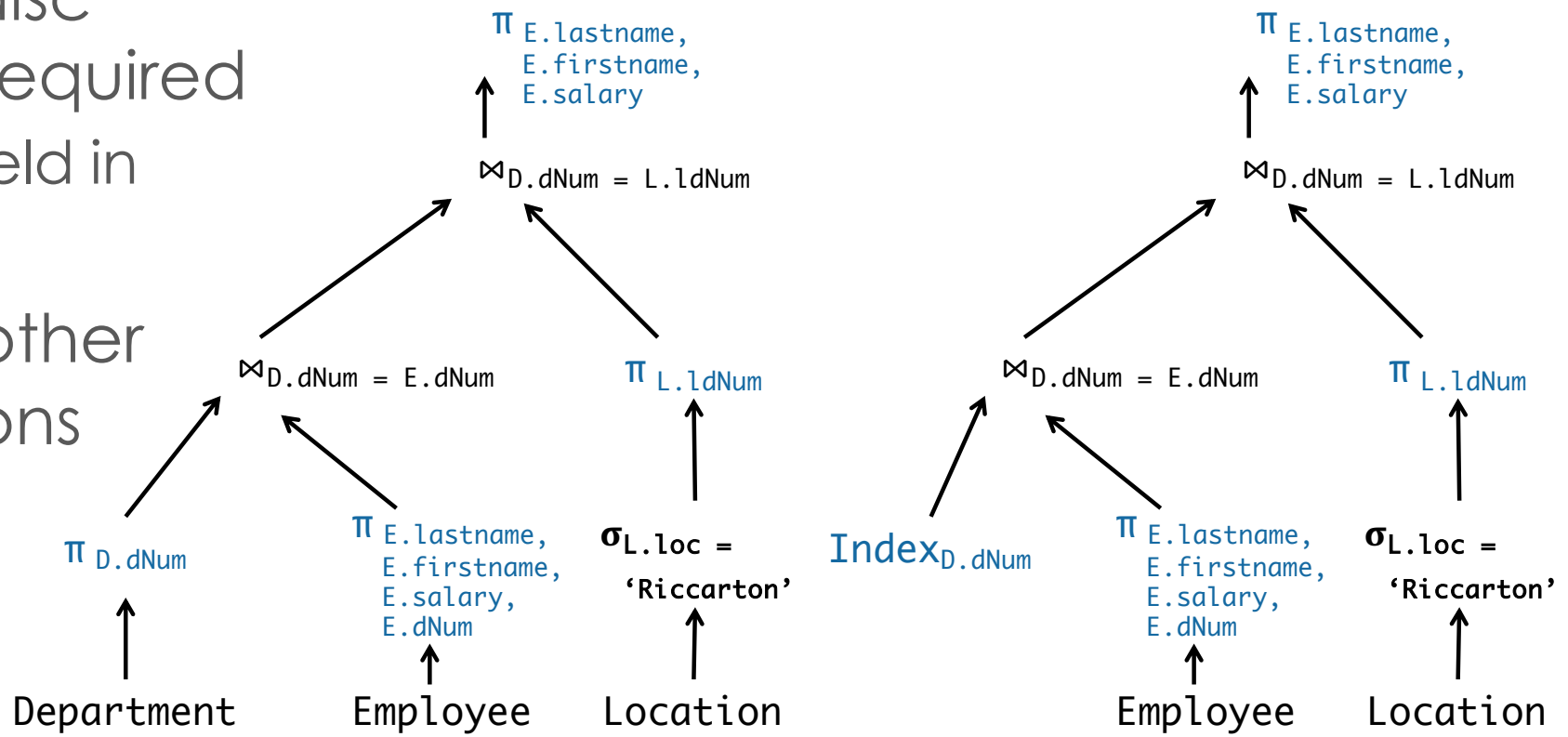
The right tree is more efficient,
because the join doesn't have so many tuples entering it





Index Access

- Reduces disc accesses required
  - Indexes held in memory
- Useful for other optimizations



JOIN Implementations

- Many alternatives
  - Generate the same answer
- Different runtime characteristics
  - Speed
  - Use of indexes
  - Use of resources: number of disk accesses

- Join algorithms:
  - Nested Loops
  - Index nested loops
  - Hash join
  - Merge/sort join

https://en.wikipedia.org/wiki/Category:Join_algorithms

Nested Loops Join

Department

dNum	…
1	…
2	…
3	…
…	…

Employee

ssn	dNum	…
122	1	…
123	1	…
124	2	…
…	…	…

Result

dNum	…	ssn	…
1	…	122	…
1	…	123	…
1	…	…	…
2	…	124	…
2	…	…	…

Index Nested Loops

Department

dNum	…
1	…
2	…
3	…
…	…

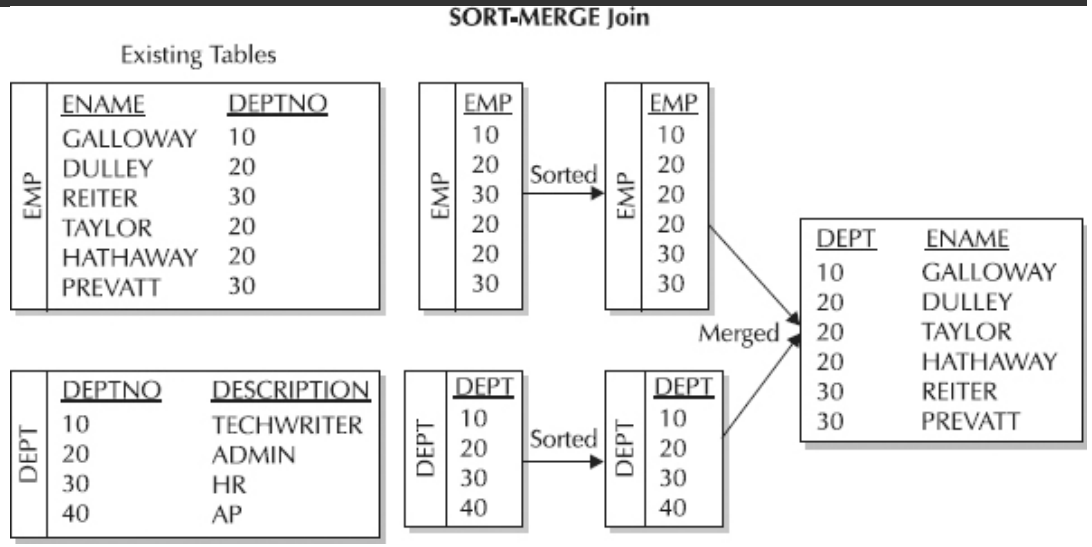
Employee

ssn	dNum	…
122	1	…
123	1	…
124	2	…
…	…	…

- Use index on Employee.dNum

More Joins

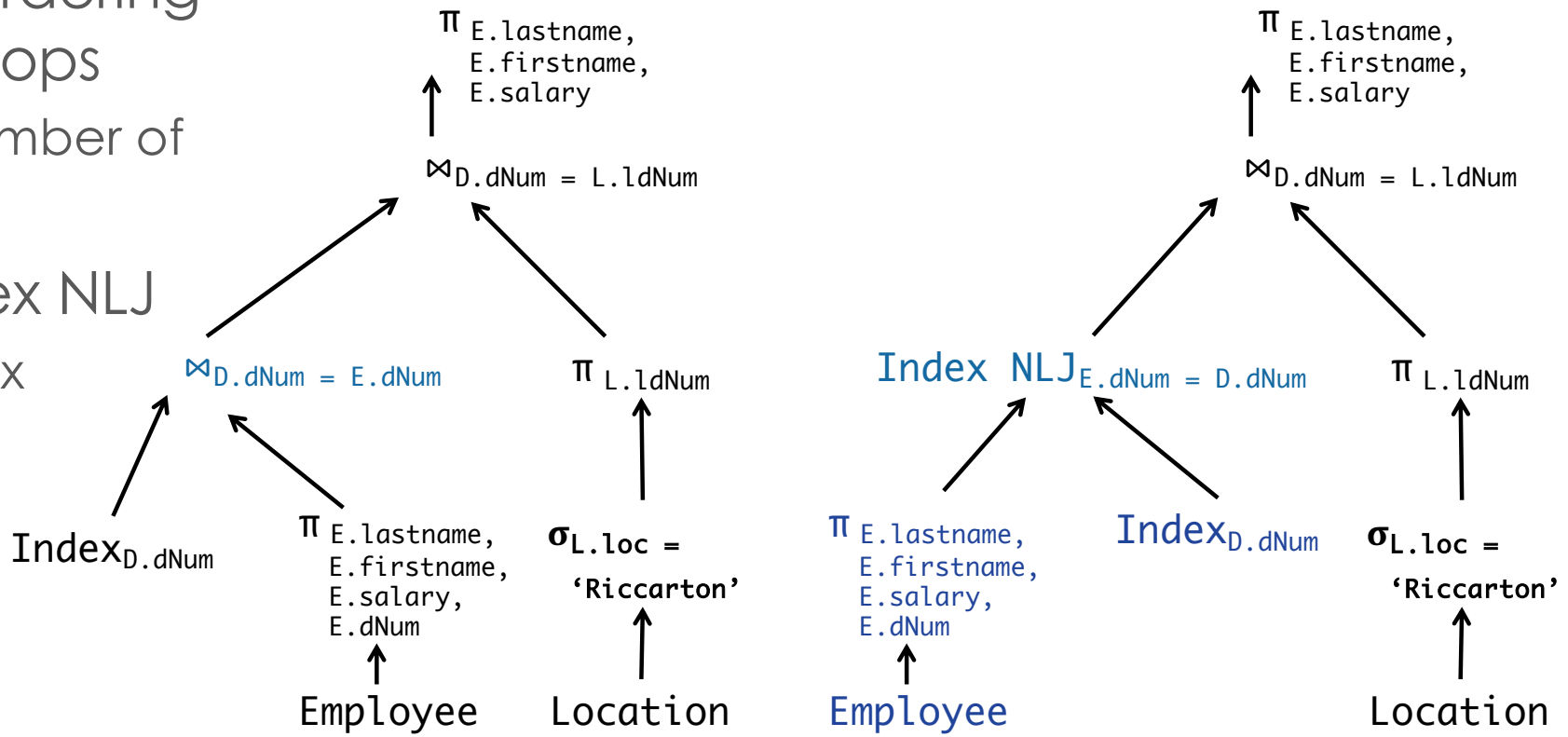
- Sort-Merge join
  - Sort both inputs
  - Merge
- Hash join
  - Hash one of the inputs
  - Join same as index nested loops



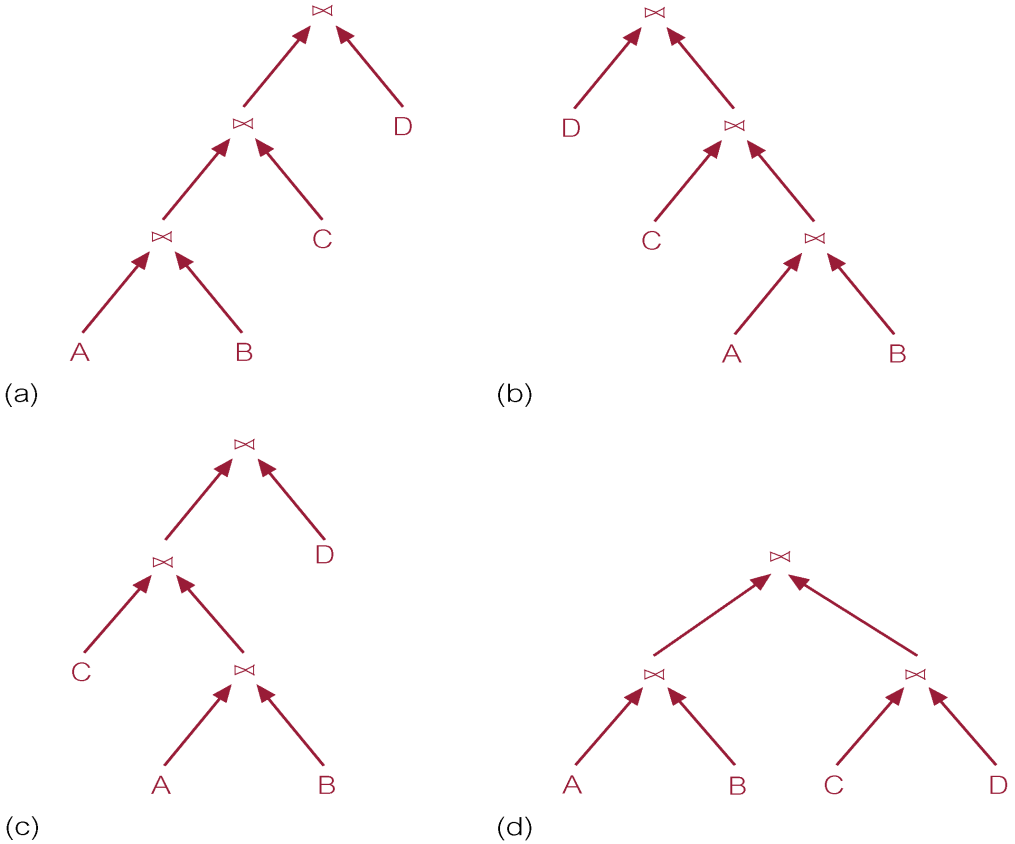
https://logicalread.com/oracle-11g-sort-merge-joins-mc02

Join Optimisation

- Switch join ordering for nested loops
  - Reduces number of scans
- Choose Index NLJ
  - Exploits index



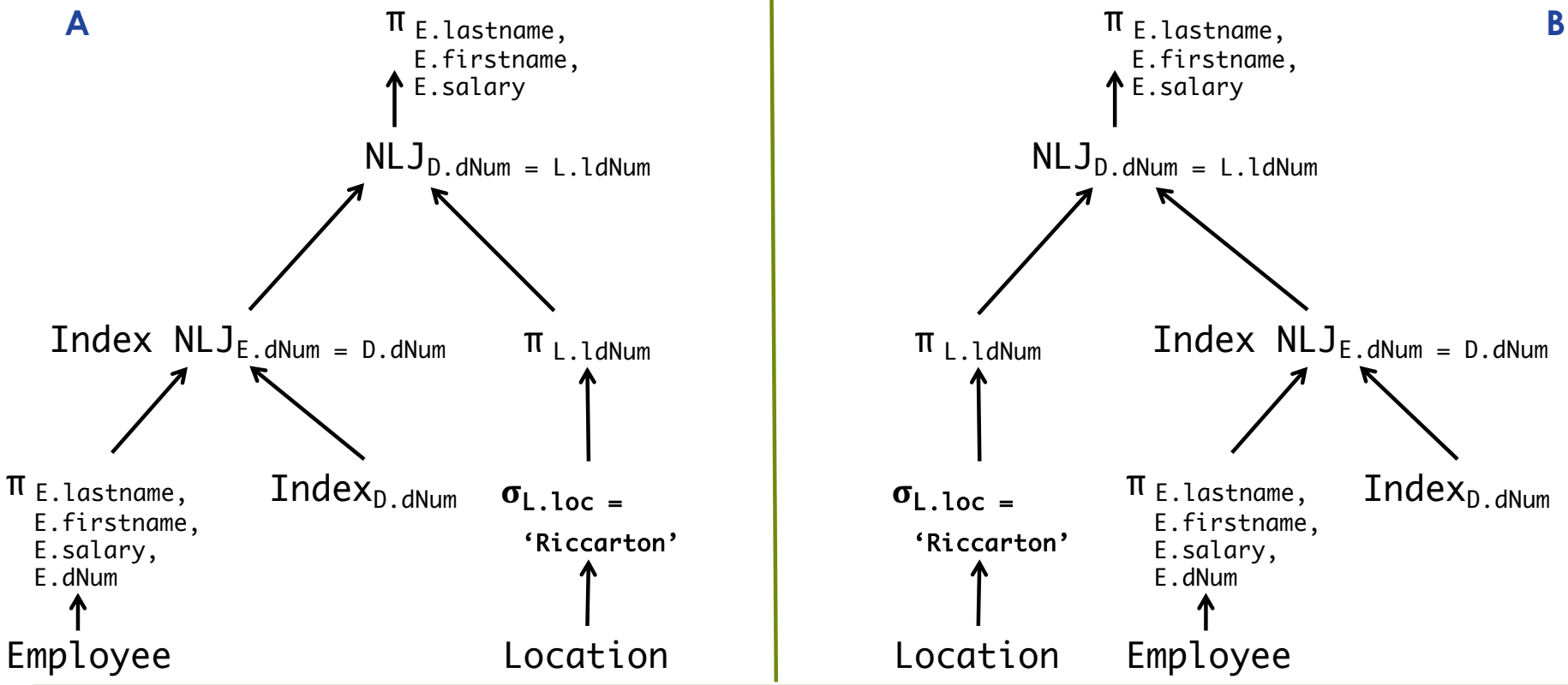
Join ordering



Pipelining

- Results not materialised
- Data flows from one operator to the next
- Implementations favour left-deep trees (a)

Join Ordering: which is more efficient?



Cost-based Optimisation

- Estimate execution cost of each plan
  - Dominated by disk access cost
    - Join selection: different joins have different costs
    - Index access
  - Estimate selectivity of conditions
- Based on data statistics
  - Statistics need to be fresh
  - Cost in maintaining stats
- Choose plan with lowest cost for optimization goal

Database Statistics

- Relation
  - Cardinality: number of rows
  - Number of disc blocks
  - Number of tuples per block
- Attribute: calculate selectivity
  - Number of distinct values
  - Minimum value
  - Maximum value
  - Data distribution
- Index
  - Number of levels
  - Number of leaf blocks

Summary

Summary: Query Processing

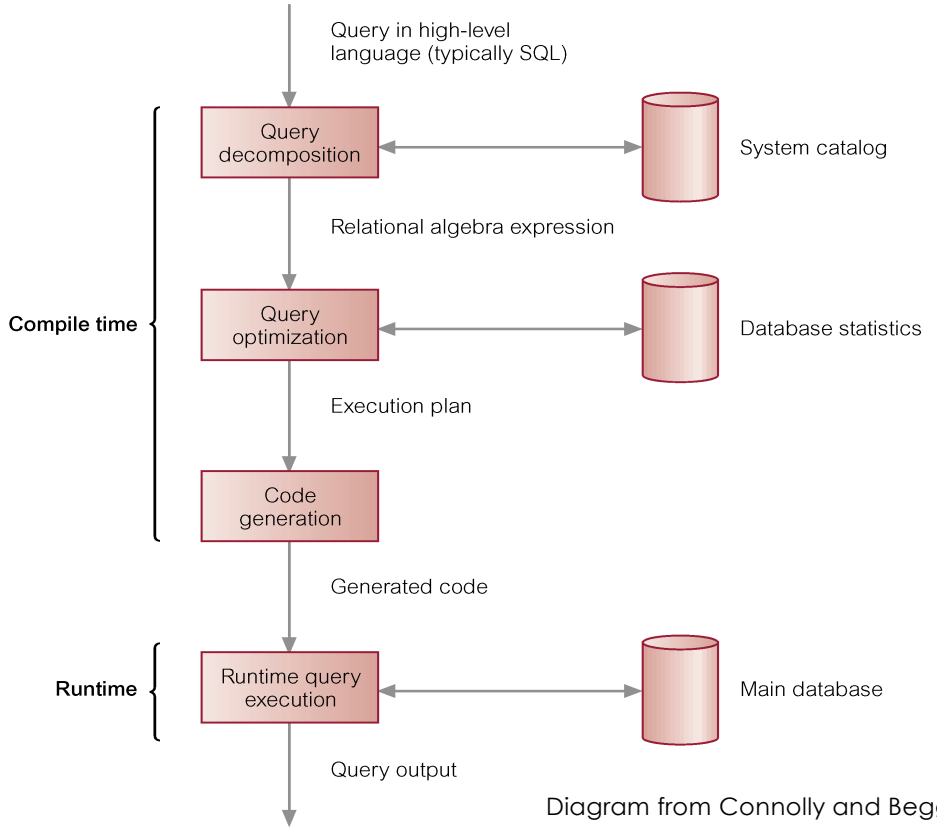


Diagram from Connolly and Begg. Database Systems. Addison Wesley

Note the use of the system catalog and database statistics to inform query planning.

Summary: Query Optimisation

- Query processing: parse, optimise, execute
- Optimisation:
  - Logical based on relational algebra
    - Push selection and projection down – reduces intermediary result sizes
    - Reorder joins so smallest tables considered first
  - Physical: choose join implementation and table access
    - Exploit indexes
- Cost plans based on data statistics and selectivity estimates
- Explain plan shows the DBMS choice

```
EXPLAIN SELECT …
```

Summary: Relational Algebra

- Provides a mathematical formalism for querying
  - Selection σ_A: applies condition A to rows
  - Projection $\pi_{a1,\ldots,an}$: returns columns a1,…,an
  - Cartesian product $R \times S$: returns a table consisting of each row from R linked with each row from S
  - Join $R \bowtie_A S$: returns the combined rows of R and S joined on condition A
- Operators are composable to form expressions
- Equivalence rules: enable manipulation of algebra expressions

References

Connolly, T., & Begg, C. (2005). *Database Systems: A Practical Approach to Design, Implementation, and Management* (4th ed.). Addison Wesley. Chapters 4 & 21

Ward, P. (2008). *Database Management Systems* (2nd ed.). Middlesex University Press. Chapter 7