COSC210
Database Mangement Sytems

Lecture 18 - Algorithms for Query Processing and Optimisation

Dr. Mitchell Welch / Dr. Edmund Sadgrove

Reading

- Chapter 19 from ***Fundamentals of Database Systems*** by Elmazri and Navathe

---

Summary

- Query Tree Notation
- Query Tree Optimisation
- Query Transformation Rules
- An Algebraic Query Optimisation Algorithm
- Selectivity and Cost Estimations

Query Tree Notation

- A **Query Tree** is a data structure that corresponds to a relational algebra expression.
    - The inputs to the relation are the leaf nodes.
    - The RA operations are represented by the internal nodes of the tree.
- **Example**:

$$\pi_{pnumber,dnum,lname,address,bdate}(((\sigma_{plocation='Stafford'}(PROJECT))$$
$$\bowtie_{dnum=dnumber}(DEPARTMENT))$$
$$\bowtie_{mgrssn=ssn}(EMPLOYEE))$$

Query Tree Notation

* **Example (a)** - *Query Tree*: * The **leaves** represent the **relations**. * **Operations** are represented by **nodes**. * The **query tree** also represents an order of operations of the RA expression. * Bottom up, the order can be derived. * SQL:

```
SELECT P.pnumber, P.dnum, E.lname, E.address, E.bdate
FROM PROJECT as P, DEPARTMENT as D, EMPLOYEE as E
WHERE P.dnum = D.dnum AND D.mgrssn = E.ssn AND
      P.plocation  = 'Stafford'
```

Query Tree Notation

* **Example (c)** - *Query Graph*: * The **query graph** does not indicate the order of the operations. * **Selection** and **Join** conditions are represented by edges. * **Nodes** represent the **relations** involved. * Constant nodes (**double ovals**/circles): * Represent **constant values** (a result of selection).

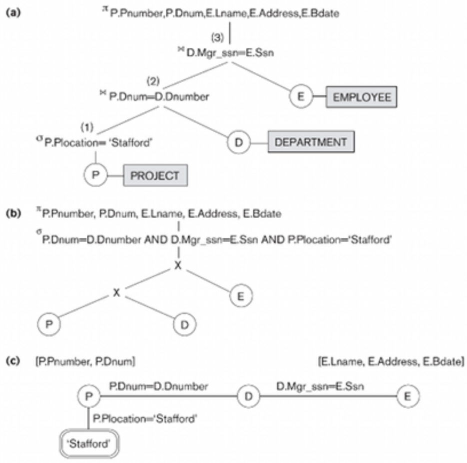


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

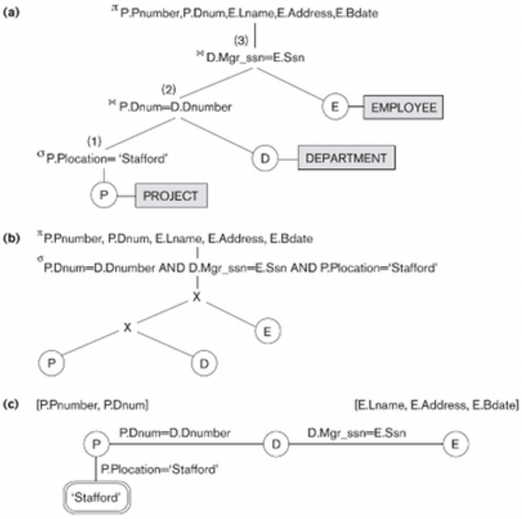


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

- **Query trees** are the preferred representation as the optimiser needs to show the order of operations for query executions.

Query Tree Optimisation

- Multiple different **RA** expressions (and there **query trees**) can be equivalent.
    - Yielding the same results, but some will be more efficient than others.
- The query **parser** will generate an initial tree:
    - This is depicted in Figure 19.4 (b).
    - The 'X' symbols represent **cartesian product**.
        - This is very inefficient.
    - This is a standard form that can be generated from the SQL query.

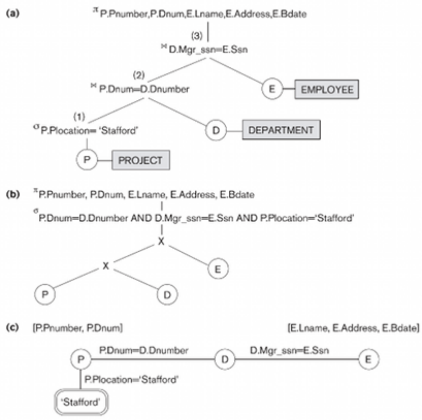


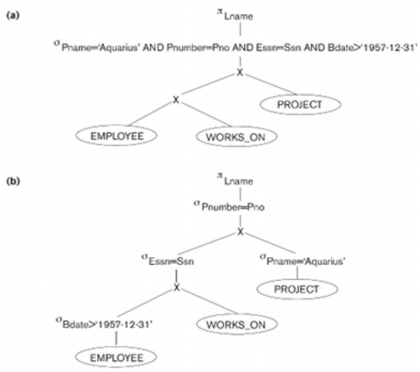
Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Query Tree Optimisation

- Initial canonical query tree (a):
    - **Optimisation** process involves moving the operations down the query tree.
        1. Move the `SELECT` operations down the tree (b).
        2. Re-organise the tree: the most selective operations appear first (c).
        3. **Cartesian product** operations are replaced with **joins** (d).
        4. Then the **projections** are moved down the tree (e).
- A motivating Example:

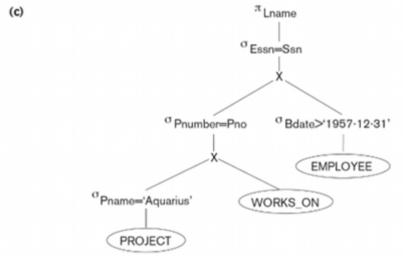
```
SELECT lname
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE pname = 'Aquarius' AND pnumber=pno AND
      essn=ssn AND bdate > '1957-12-31'
```

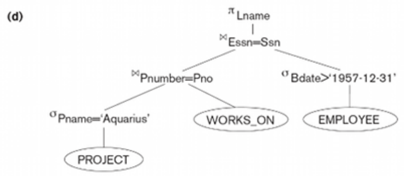
Figure 19.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

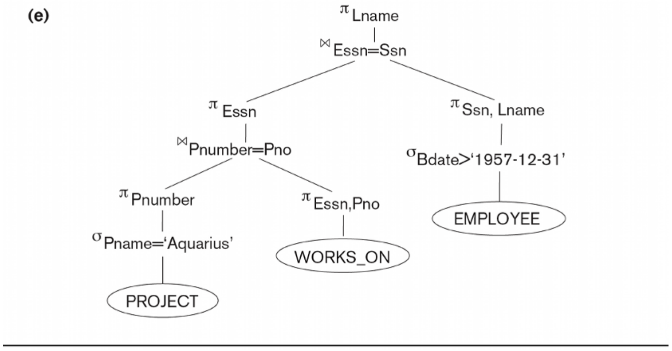


Query Tree Optimisation

- Steps 2, 3 and 4 of optimizing the query tree:







* The query optimiser must know which transformations can be performed while still preserving equivalence.

Query Tree Optimisation.

- Transformation rules can be used to perform optimisations of RA expressions, these include:
  - **Commutativity**:
    - Operations who's order can be changed, in and outside parathesis.
    - E.g. $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$
  - **Associativity**:
    - Where the emphasis on individual operations in an expression can be changed.
    - E.g. $(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$
  - **Cascades**:
    - Conditions that can be broken up into individual operations.
    - E.g. $\sigma_{c_1\ AND\ c_2\ AND..\ AND\ c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$

Query Transformation Rules

- **Cascade** of **σ** A **conjunctive** selection conditions:
  - Can be broken up into a cascade of individual **σ** operations:
  - $\sigma_{c_1\ AND\ c_2\ AND..\ AND\ c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$
- **Commutativity** of σ. The σ operation is commutative:
  - Same result regardless of order:

* $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$

Query Transformation Rules

- **Cascade** of **π**:
  - In a cascade (sequence) of π operations, all but the last one can be ignored:
  - $\pi_{List_1}(\pi_{List_2}(\ldots(\pi_{List_n}(R))\ldots)) \equiv \pi_{List_1}(R)$
- **Commuting σ with π**:
  - If the selection condition inolves Attributes $A_1, \ldots, A_n$ in the projection list,
  - The two operations can be commuted:

- $\pi_{A_1,A_2,\ldots,A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,\ldots,A_n}(R))$

Query Transformation Rules

- **Commutativity** of ⋈ (and ×).
  - The **join** operation is commutative, as is the **×** operation:
  - $R \bowtie_c S \equiv S \bowtie_c R$
  - $R \times S \equiv S \times R$
- Note: the order of the attributes may be different in the result

Query Transformation Rules

- **Commuting σ** with **⋈** (or **×**):
  - Conditions on the attributes of one relation can be commuted.
  - $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$
  - Conditions on individual relations can also be communted.
  - $\sigma_c(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$
- The same rules apply if the is replaced by a **×** operation.

Query Transformation Rules

- **Commuting π** with **⋈** (or **×**):
  - Projections on attributes of individual relations can be communited.
  - $\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\ldots,A_n}(R)) \bowtie_c (\pi B_1, \ldots, B_m(S))$
- If the join condition (**c**) includes additional attributes not in *L*:
  - These must be added to the projection list.

Query Transformation Rules

- **Set operations** (excluding MINUS) are **commutative**.
- JOIN, CARTESIAN PRODUCT, UNION and INTERSECTION are all individually **associative**:
  - $(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$
- Theta represents any of these operations.

Query Transformation Rules

- **Commuting σ** with set **operations**.
- The σ operation commutes with ∪, ∩ and −:
  - $\sigma_c(R\ \theta\ S) \equiv (\sigma_c(R))\ \theta\ (\sigma_c(S))$

Query Transformation Rules

- The **π operation commutes** with **∪**:
    - $\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$
- Finally we can Convert a **(σ, ×)** sequence into ⋈.
    - $(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

An Algebraic Query Optimisation Algorithm

- We can now apply these transformations to our queries to optimise for execution.
- This **algorithm** outline 6 Broad Stages:
    - **Step 1**:
        - We use rule (1) to break up the **conjunctive** conditions within any select operations.
    - **Step 2**:
        - The next step is to apply the **commutativity** of the **SELECT operation** (rule 2).
        - The select operations can be moved down the query tree as far as possible.
            - SELECT operations with attributes from a single table can be moved to a **leaf node**.
            - SELECT operations that involve attributes from multiple tables represent a **join condition**.
                - These can only be placed after the tables have been combined.

An Algebraic Query Optimisation Algorithm

- Algorithm continued:
    - **Step 3**:
        - Next we use the rules for **commutativity** and **associativity** of the binary operations.
        - The SELECT operations with the **lowest selectivity** are moved so they **executed first**.
        - This should be done so that the select operation is only carried out on already joined relations.
            - If the SELECT involves attributes from multiple tables.
    - **Step 4**:

* CARTESIAN PRODUCT operations are combined SELECT operations to produce a JOIN (rule 12).

An Algebraic Query Optimisation Algorithm

- Algorithm continued:
    - **Step 5**:
        - Using the rules regarding the PROJECT operation can be applied.
        - The **projection** operations should be pushed down the tree as far possible.
        - Only the attributes required in the final result or subsequent operations should be retained after each project.
        - This may require the creation of new project operations

---

An Algebraic Query Optimisation Algorithm

- Algorithm continued:
    - **Step 6**:

- Identify subtrees that represent groups that can be executed by a single algorithm.
- Figure 19.5 uses this approach to optimise the example shown.

- The main idea behind this algorithm is reduce the size of the intermediate results as early as possible.

Selectivity and Cost Estimations

- The **query optimiser**:
  - Does not solely rely on the heuristic rules for improving performance.
  - The query optimiser compares different strategies and chooses the strategy with the *lowest* cost.
- The optimiser must also limit the number of possibilities tested.
  - This process is best suited to compiled queries.
  - Optimisation is not carried out at runtime.

Selectivity and Cost Estimations

- The cost components that contribute to the total cost of query execution include:
  - Access to **secondary storage**: Hard drive speed.
  - **Intermediate disk usage**: Cache.
  - **Computation Costs**: Sorting/searching/merging.
  - **Memory Usage** Cost: Block reads.
  - **Communication**: Between the database and application.

Selectivity and Cost Estimations

- In order to calculate the costs the execution strategies:
  - The **DBMS** must maintain a **catalogue** of information.
  - Some if the information stored includes:
    - File information such as:
      - **Tuple counts**.
      - **Record size**.
      - Number of **blocks**.
      - **Blocking factor**.
      - **Index levels** and number of first level indexes.

Selectivity and Cost Estimations

- Other information stored includes:
  - The number of distinct values.
  - The **selectivity** for each attribute.
- Some of this information is relatively static, other items change constantly.
- The optimiser needs reasonably up-to-date information for optimisation.

Selectivity and Cost Estimations

- Some examples of Cost estimations on the SELECT operation:
  - **Linear search**:
    - Complexity: linear: *O(N)* worst case.
  - **Binary Search**:
    - Complexity: logarithmic: $O(log_2 b)$ file blocks accessed.
  - **Primary key index**:
    - Complexity: constant: O(1) - one access for each level of the index.
    - Plus one access for the actual record.
  - **B-Tree Index**:
    - Complexity: O(log(nodes)) - one block access for each level of the B-Tree.
    - Plus one access for the data.

Selectivity and Cost Estimations

- Some examples of the cost estimations on the Join operation:
  - Where relation R has b_r blocks and S has b_s blocks:
  - Nested-Loop Join: If R is in the outer loop:
    - $C_{J1} = b_R + (b_R * b_S) + ((j_s * |R| * |S|)/bfr_{RS})$

Selectivity and Cost Estimations

* Example query: * Suppose it has the following meta data in the catalogue.

```
SELECT P.pnumber, P.dnum, E.lname, E.address, E.bdat
FROM PROJECT as P, DEPARTMENT as D, EMPLOYEE as E
WHERE P.dnum = D.dnum AND D.mgrssn = E.ssn AND
      P.plocation  = 'Stafford'
```

Figure 19.8
Sample statistical information for relations in Q2. (a) Column information. (b) Table information. (c) Index information.

(a)

Table_name	Column_name	Num_distinct	Low_value	High_value
PROJECT	Plocation	200	1	200
PROJECT	Pnumber	2000	1	2000
PROJECT	Dnum	50	1	50
DEPARTMENT	Dnumber	50	1	50
DEPARTMENT	Mgr_ssn	50	1	50
EMPLOYEE	Ssn	10000	1	10000
EMPLOYEE	Dno	50	1	50
EMPLOYEE	Salary	500	1	500

(b)

Table_name	Num_rows	Blocks
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

(c)

Index_name	Uniqueness	Blevel*	Leaf_blocks	Distinct_keys
PROJ_PLOC	NONUNIQUE	1	4	200
EMP_SSN	UNIQUE	1	50	10000
EMP_SAL	NONUNIQUE	1	50	500

*Blevel is the number of levels without the leaf level.

Selectivity and Cost Estimations

- First we examine the possible join ordering.
  - The potential **join orders** without considering the CARTESIAN PRODUCT:

$$PROJECT \bowtie DEPARTMENT \bowtie EMPLOYEE$$

$$DEPARTMENT \bowtie PROJECT \bowtie EMPLOYEE$$

$$DEPARTMENT \bowtie EMPLOYEE \bowtie PROJECT$$

$$EMPLOYEE \bowtie DEPARTMENT \bowtie PROJECT$$

Selectivity and Cost Estimations

- Starting with the join between `PROJECT` and `DEPARTMENT`:
  - `DEPARTMENT` doesn't have any indexing structures (based upon the info in table 19.8).
    - As a result, a linear search is the only option.

- The `PROJECT` table has the `PROJ_PLOC` **index** on the project name attribute.
  - This index is non-unique, so the optimiser will assume a uniform distribution.
  - From this the number of corresponding rows for each key value can be estimated.

- Using this information, the optimiser can estimate the total **block accesses** for the operation.

Selectivity and Cost Estimations

- We then calculate the cost of the second join:
  - This operation can make use of the single-loop join.
    - As there is an index on the EMPLOYEE relation.
  - The block accesses for each item is given by (x+1) where x is the level.
  - For PROJECT x=2, therefore 10 lookups will result in 30 block accesses.
    - $Selectivity \times Num\_rows = 10$
    - Where, $selectivity = 1/Num\_distinct$
- The optimiser can than add the total cost of the operation up.

Selectivity and Cost Estimations

- The optimiser can then perform similar calculations on the other potential join combinations.
- The cheapest approach can then be selected.

Summary

- Query Tree Notation
- Query Tree Optimisation
- Query Transformation Rules
- An Algebraic Query Optimisation Algorithm
- Selectivity and Cost Estimations

Reading

- Chapter 19 from ***Fundamentals of Database Systems*** by Elmazri and Navathe

---