Lecture 13

Query Processing and Optimization

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Query Processing

- A **database query** expressed in a **high-level query language** such as **SQL** must first be **scanned, parsed**, and **validated.**
- The **scanner** identifies the **query tokens**—such as **SQL keywords, attribute names**, and **relation names**—that appear in the text of the query.
- The **parser checks** the **query syntax** to determine whether it is formulated according to the **syntax rules** (rules of grammar) of the **query language.**
- The **query** must also be **validated** by checking that **all attribute** and **relation names** are **valid** and **semantically meaningful names** in the schema of the particular database being queried.
- An **internal representation** of the **query** is then **created,** usually as a **tree** data structure called a **query tree**.

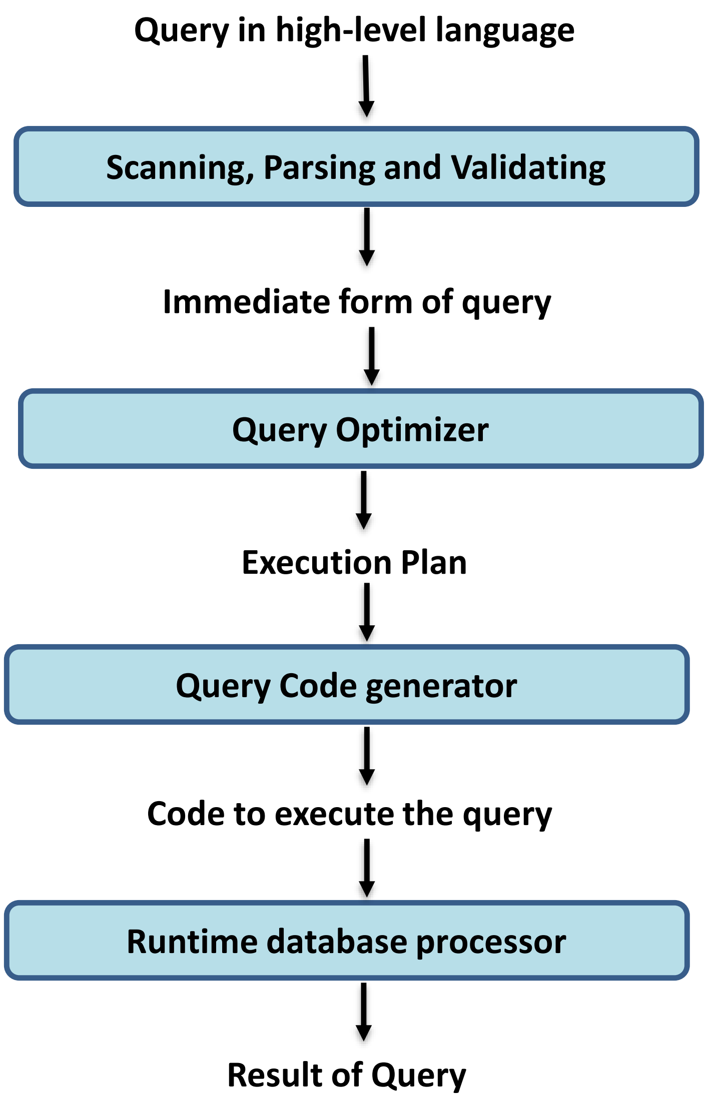
Query Optimization

- **Why optimizing a query?**
- So that it is processed efficiently.
- **What is meant by efficiently?**
- Minimizing the cost of query evaluation.
- **Who will minimize?**
- The system, not the user.
- ***Query optimization*** *is facilitating a system to construct a* ***query-evaluation plan*** *for* ***processing a query efficiently****, without expecting users to write efficient queries*

Query Optimization

- The **DBMS** must then **devise** an **execution strategy** or **query plan** for **retrieving** the **results** of the **query** from the **database files.**
- A **query** typically has **many possible execution strategies**, and the **process** of **choosing** a **suitable one** for processing a query is known as **query optimization**.
- There are **two main techniques** that are employed during **query optimization.**
- The **first technique** is based on **heuristic rules** for **ordering the operations** in a **query execution strategy.**
- **A heuristic** is a **rule** that works well in most cases but is **not guaranteed** to work well in **every case.**
- The **rules** typically **reorder the operations** in a **query tree.**
- The **second technique** involves **systematically estimating** the **cost**(*access cost, storage cost, computation cost, communication cost*) of **different execution strategies** and **choosing the execution plan** with the **lowest cost estimate.**
- These **techniques** are usually **combined** in a **query optimizer.**

Query Processing & Optimization



Query Tree

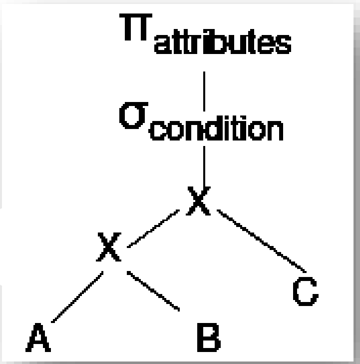
- **Tree** that represents a **relational algebra expression** corresponding to a **SQL statement** is referred as **Query Tree.**
- **Leave nodes** represent the base tables.
- **Internal nodes** represent the relational algebra operators applied to the child nodes.
- The **query tree** is **executed** from **leaves to root.**
- An **internal node** can be **executed** when its **operands** are **available** (children relation has been computed)
- **Internal Node** is **replaced** by the **result of the operation** it represents.
- **Root node** is **executed last**, and is **replaced** by the **result** of the **entire tree.**

Canonial query tree

- Construct the **canonical(initial) query tree** as follows:
  - **Cartesian product** of the FROM-tables
  - **Select** with WHERE-condition
  - **Project** to the SELECT-attributes

- **Example:**

  SELECT attributes
  FROM A, B, C
  WHERE condition

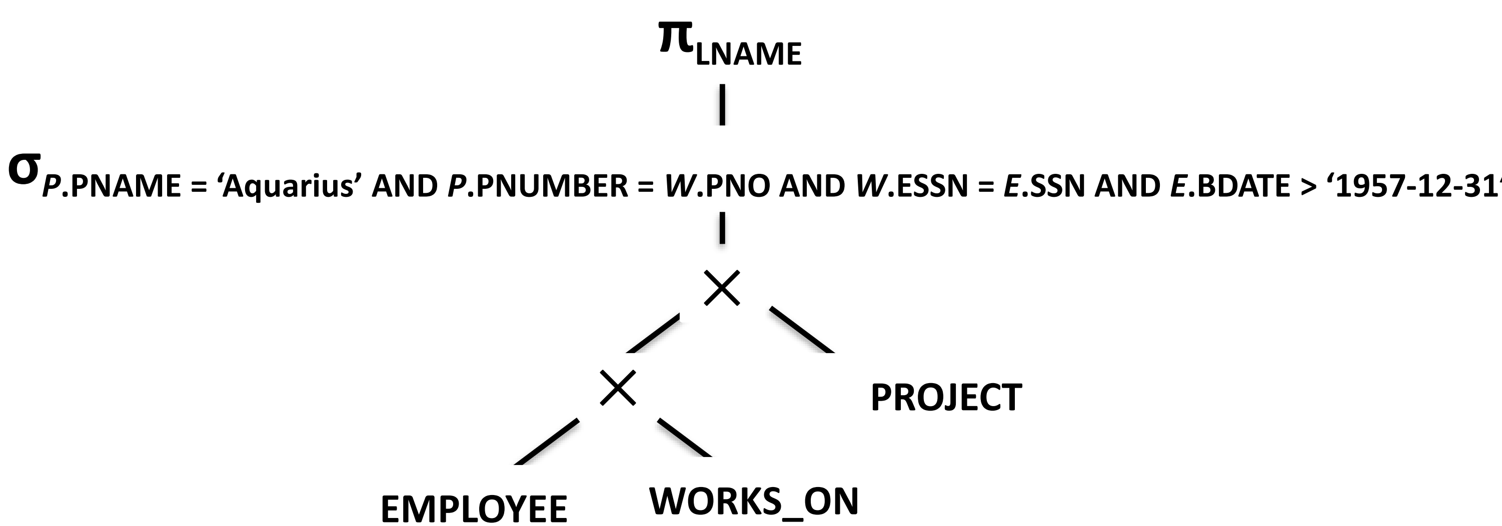


Canonical Query Tree

- **Example:** List the last name of the employees born after 1957 who work on a project named ”Aquarius”.

  SELECT *E*.LNAME **FROM** EMPLOYEE *E*, WORKS_ON *W*, PROJECT *P*

  WHERE *P*.PNAME = ‘Aquarius’ AND *P*.PNUMBER = *W*.PNO AND *W*.ESSN = *E*.SSN AND *E*.BDATE > ‘1957-12-31’



Heuristics based Query Optimization

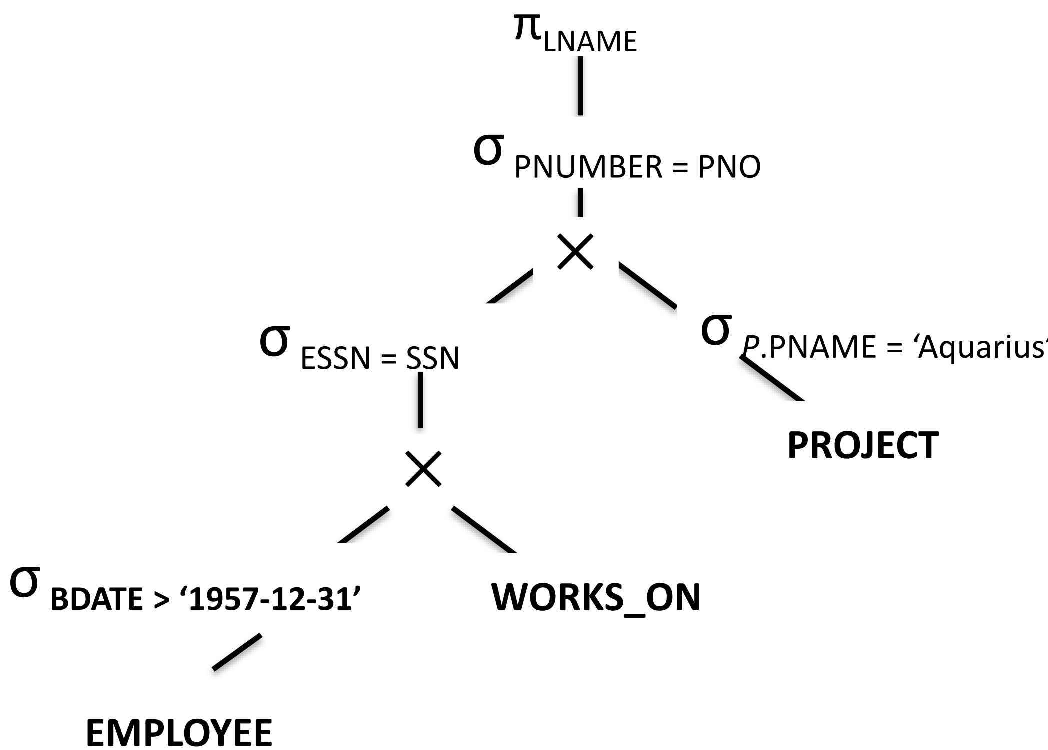
- **Heuristic rules** are used to **modify the internal representation** of a **query**—which is usually in the form of a **query tree** or a **query graph** data structure—to **improve** its **expected performance.**
- The **scanner** and **parser** of an **SQL query** first generate a data structure that corresponds to an ***initial query representation,*** which is then **optimized** according to **heuristic rules**.
- This leads to an ***optimized query representation***, which corresponds to the **query execution strategy.**
- Following that, a **query execution plan** is generated to execute groups of operations based on the access paths available on the files involved in the query.

Heuristics based Query Optimization

- One of the **main heuristic rules** is to *apply SELECT and PROJECT operations before applying the JOIN* or other binary operations, because the size of the file resulting from a binary operation—such as JOIN—is usually a multiplicative function of the sizes of the input files.
- The **SELECT** and **PROJECT** operations **reduce the size** of a file and hence should be applied before a join or other binary operation.
- Thus in a **Query tree** we should **try to move SELECT and PROJECT operations down**.

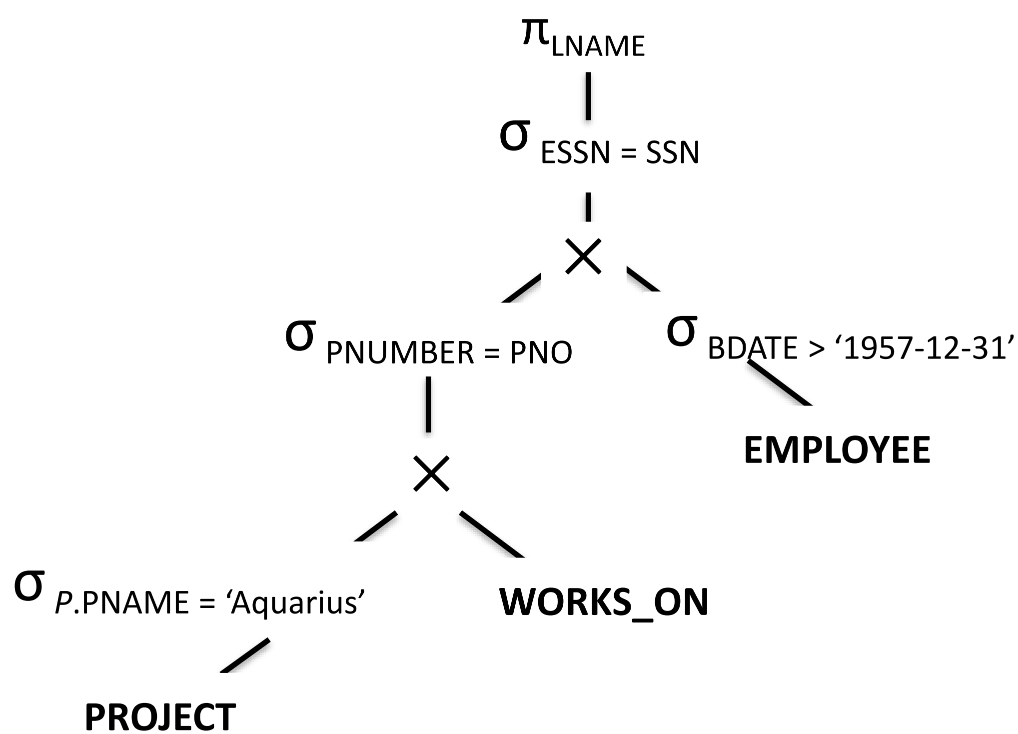
Heuristics based Query Optimization

- **Example:** Move **SELECT** down:
-



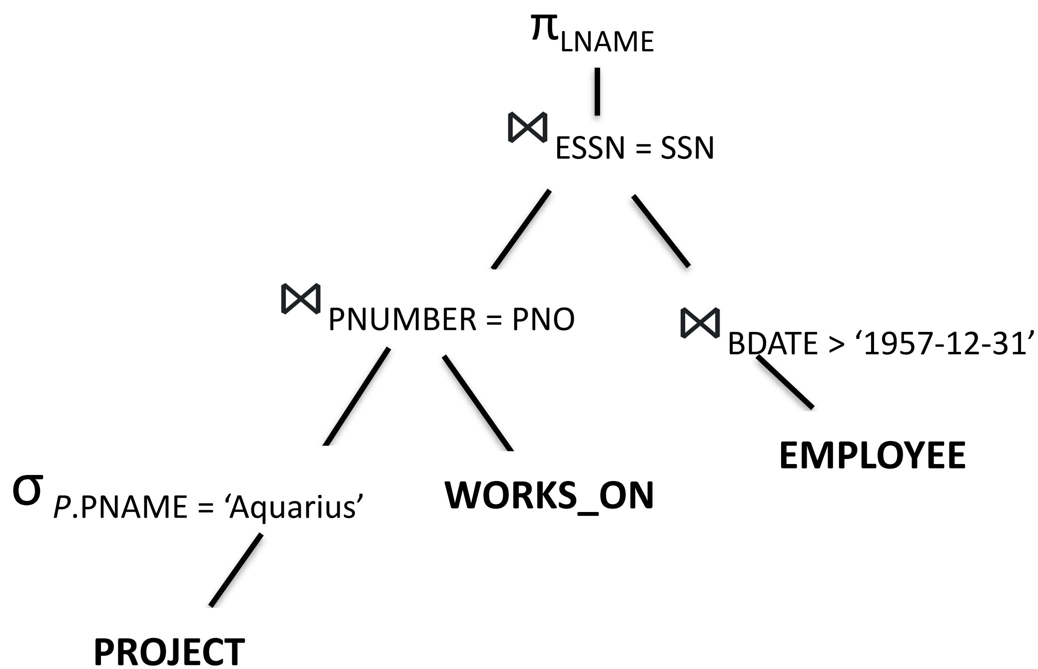
Heuristics based Query Optimization

- Apply most **restrictive Select first.**



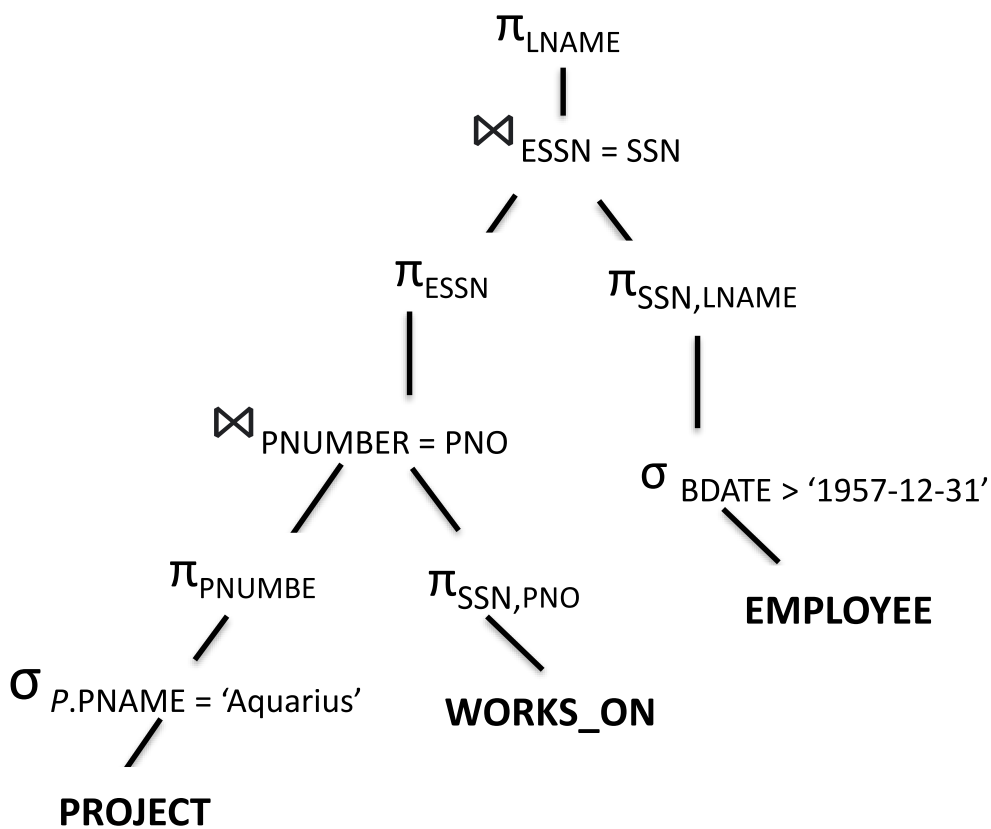
Heuristics based Query Optimization

- Replace **Cartesian product** with **Join.**



Heuristics based Query Optimization

- **Restrict selection** results with **projection.**



Rules for Transformation of Relational Expressions

- A query can be expressed in several different ways, with different costs of evaluation.
- Two **relational-algebra expressions** are said to be **equivalent** if, on every legal database instance, the two expressions generate the same set of tuples.
- **Equivalence Rules**
- An **equivalence rule** says that expressions of two forms are equivalent.
- Let $\boldsymbol{\theta_1}$ $\boldsymbol{\theta_2}$ $\boldsymbol{\theta_3}$…………………… Are conditions(predicates).
- **Rule 1:** Conjunctive selection operations can be deconstructed into a sequence of individual selections. This transformation is referred to as a cascade of $\boldsymbol{\sigma}$.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

Rules for Transformation of Relational Expressions

- **Rule 2: Selection** operations are **commutative**.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

  - In practice, it is better and more optimal to apply that selection first which yields a fewer number of tuples.
  - This saves time on our outer selection.
- **Rule 3: Selections** on **Cartesian Products** can be re-written as **Theta Joins(Natural Join)**.

$$\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$$

- The **cross product operation** is known to be **very expensive.**
- This is because it matches each tuple of E1 (total m tuples) with each tuple of E2 (total n tuples).
- This **yields m*n entries**.

Rules for Transformation of Relational Expressions

- **Rule 4:** Theta Joins are commutative.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

- **Rule 5:** Join operations are associative.

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

- Joins are all commutative as well as associative, so one must join those two tables first which yield less number of entries, and then apply the other join.

Rules for Transformation of Relational Expressions

- **Rule 6:** **Selection** operation can be **distributed**.

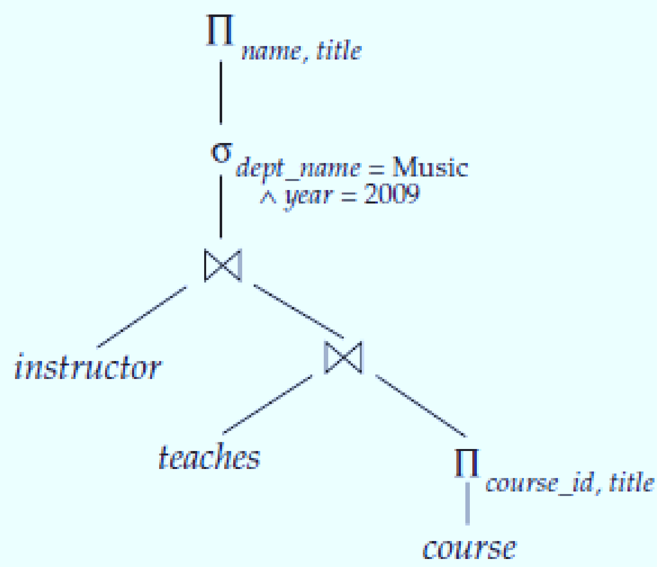
$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

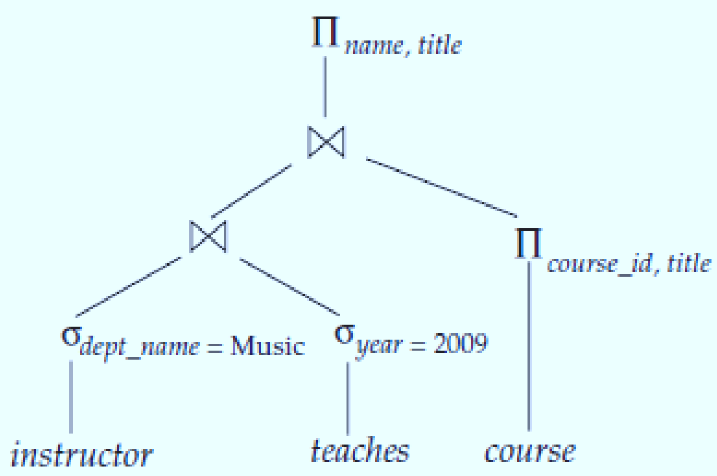
- **Rule 7:** **Projection distributes** over the **Theta Join.**

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

Example



(a) Initial expression tree



(b) Tree after multiple transformations

Heuristics for Optimization

- *Do operations that generate smaller tables first.*
- Do **selection** as **early** as possible.
- Use **cascading, commutativity, and distributivity** to **move selection** as far **down** the **query tree** as possible.
- Use **associativity** to **rearrange relations** so the **selection operation** that will produce the **smallest table** will be executed first.
- Do **projection early.**
- Use **cascading, distributivity and commutativity** to **move the projection** as far **down** the **query tree** as possible.