# CMSC424: Database Design

Instructor: Amol Deshpande

amol@cs.umd.edu

# Spring 2020 – Online Instruction Plan

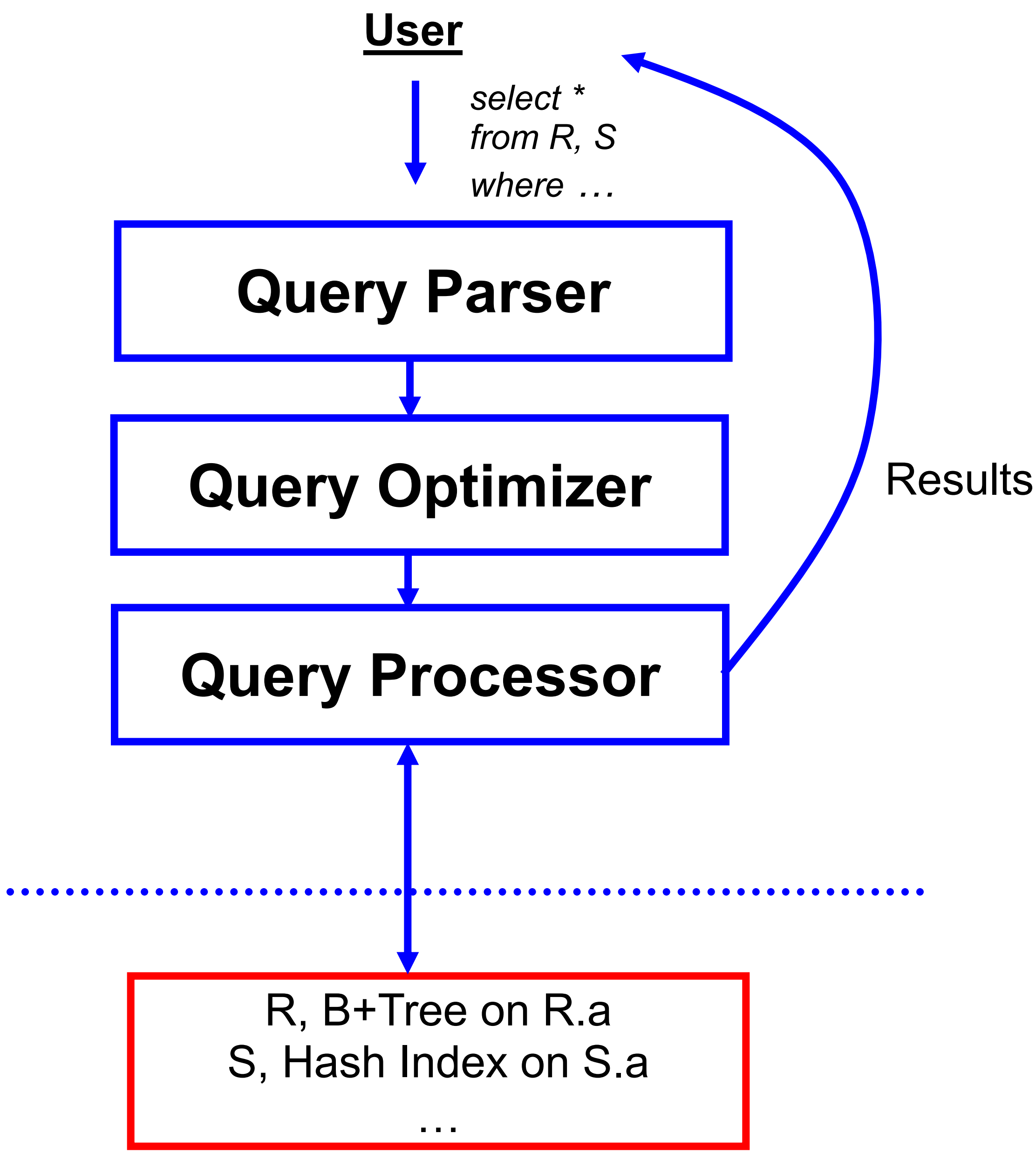
Modified to swap the last two projects

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3: Query Optimization; Architectures/Parallel 1
- Week 4: Parallel Databases + MapReduce; Transactions 1
- Week 5: Transactions 2

# Spring 2020 – Online Instruction Plan

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3 (Homework Due April 17, Noon)
  - Query Optimization 1: Overview, Statistics
  - Query Optimization 2: Equivalences, Search Algorithms
  - Architectures/Parallel Databases Introduction
- Week 4: Parallel Databases; Mapreduce; Transactions 1
  - Map-reduce and Apache Spark (will post early for Project 5)
- Week 5: Transactions 2

# Getting Deeper into Query Processing

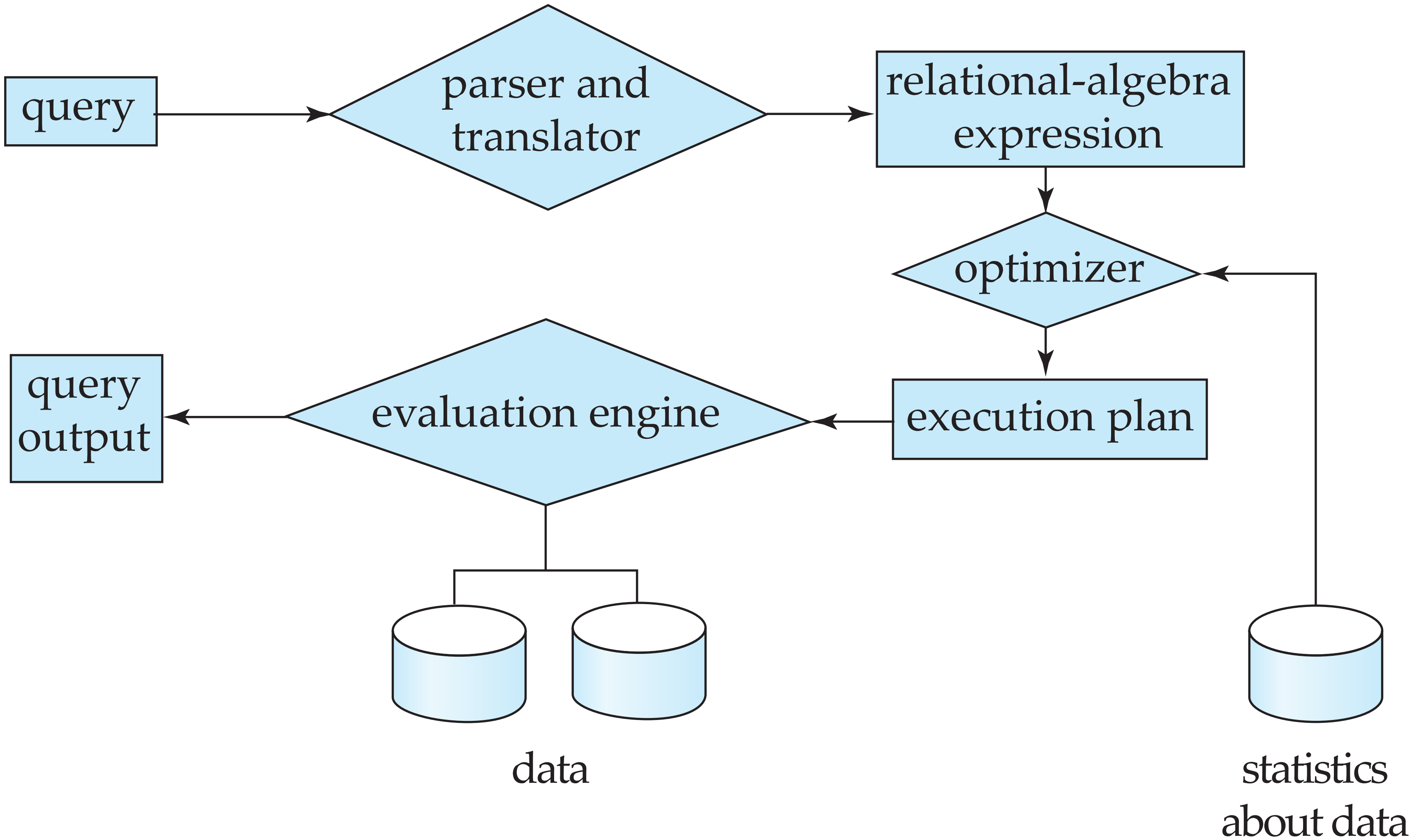



Resolve the references,
Syntax errors etc.
Converts the query to an internal format
- *relational algebra like*

Find the *best* way to evaluate the query
- Which index to use ?
- What join method to use ?
- …

Read the data from the files
Do the query processing
- *joins, selections, aggregates*
- …

# Getting Deeper into Query Processing



# Query Optimization

- Book Chapters
  - 13.1, 13.2, 13.3, 13.4
- Key topics:
  - Why query optimization is so important?
  - How to enumerate different query plans for a single SQL query
  - How to estimate the sizes of "intermediate results"
  - How to "search" the space of all query plans efficiently

# Query Optimization

- Overview
- Statistics Estimation
- Transformation of Relational Expressions
- Optimization Algorithms

# Equivalence of Expressions

- Two relational expressions equivalent iff:
  - Their result is identical on all legal databases
- Equivalence rules:
  - Allow replacing one expression with another
- Examples:

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

2. Selections are commutative

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

# Equivalence Rules

- Examples:

  3. $\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$

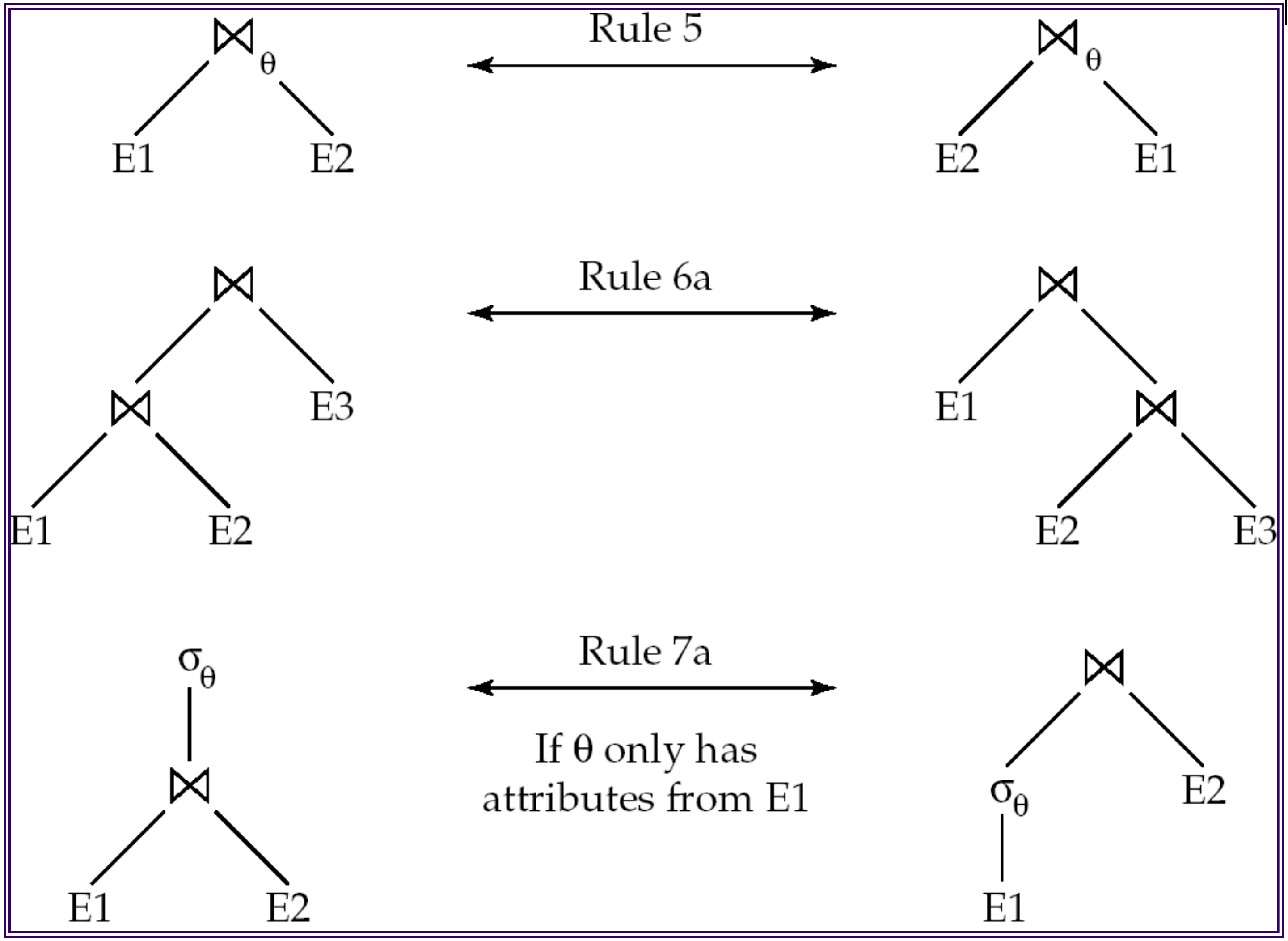
  5. $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$

  7(a). If $\theta_0$ *only involves attributes from* $E_1$

  $$\sigma_{\theta 0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_\theta E_2$$

- And so on…
  - Many rules of this type

# Pictorial Depiction



# Example

- Find the names of all customers with an account at a Brooklyn branch whose account balance is over $1000.

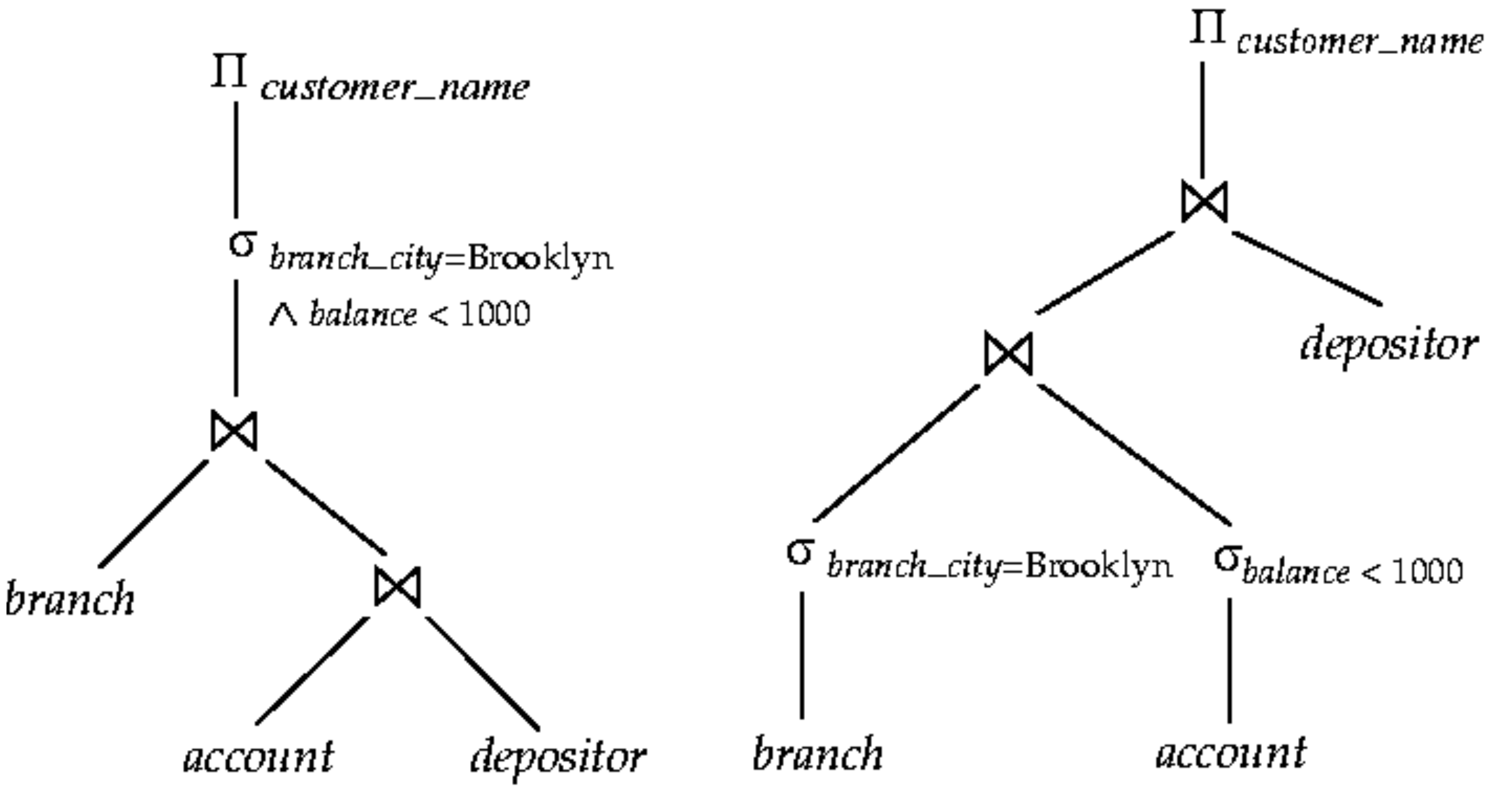
$$\Pi_{customer\_name}(\sigma_{branch\_city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie (account \bowtie depositor)))$$

- Apply the rules one by one

$$\Pi_{customer\_name}((\sigma_{branch\_city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie account)) \bowtie depositor)$$

$$\Pi_{customer\_name}(((\sigma_{branch\_city = \text{“Brooklyn”}}(branch)) \bowtie (\sigma_{balance > 1000}(account))) \bowtie depositor)$$

# Example




(a) Initial expression tree (b) Tree after multiple transformations

# Equivalence of Expressions

- The rules give us a way to enumerate all equivalent expressions
  - Note that the expressions don't contain physical access methods, join methods etc…
- Simple Algorithm:
  - Start with the original expression
  - Apply all possible applicable rules to get a new set of expressions
  - Repeat with this new set of expressions
  - Till no new expressions are generated

# Equivalence of Expressions

- Works, but is not feasible
- Consider a simple case:
  - $R1 \bowtie (R2 \bowtie (R3 \bowtie (\ldots \bowtie Rn)))\ldots.)$
- Just join commutativity and associativity will give us:
  - At least:
    - n^2 * 2^n
  - At worst:
    - n! * 2^n
- Typically the process of enumeration is combined with the search process

# Evaluation Plans

- We still need to choose the join methods etc..
  - Option 1: Choose for each operation separately
    - Usually okay, but sometimes the operators interact
    - Consider joining three relations on the same attribute:
      - $R1 \bowtie_a (R2 \bowtie_a R3)$
    - Best option for R2 join R3 might be hash-join
      - But if *R1* is sorted on *a,* then *sort-merge join* is preferable
      - Because it produces the result in sorted order by *a*
- Also, we need to decide whether to use pipelining or materialization
- Such issues are typically taken into account when doing the optimization

# Query Optimization

- Introduction
- Statistics Estimation
- Transformation of Relational Expressions
- Optimization Algorithms

# Optimization Algorithms

- Two types:
  - Exhaustive: That attempt to find the best plan
  - Heuristical: That are simpler, but are not guaranteed to find the optimal plan

- Consider a simple case
  - Join of the relations *R1, …, Rn*
  - No selections, no projections
- Still very large plan space

# Searching for the best plan

- Option 1:
  - Enumerate all equivalent expressions for the original query expression
    - Using the rules outlined earlier
  - Estimate cost for each and choose the lowest

- Too expensive !
  - Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie \ldots r_n$.
  - There are $(2(n - 1))!/(n - 1)!$ different join orders for above expression. With $n = 7$, the number is 665280, with $n = 10$, the number is greater than 176 billion!

# Searching for the best plan

- <u>Option 2:</u>
  - Dynamic programming
    - There is too much commonality between the plans
    - Also, costs are additive
      - Caveat: Sort orders (also called “interesting orders”)
  - Reduces the cost down to O(n3^n) or O(n2^n) in most cases
    - Interesting orders increase this a little bit
  - Considered acceptable
    - Typically n < 10.
  - Switch to heuristic if not acceptable

# Heuristic Optimization

- Dynamic programming is expensive
- Use *heuristics* to reduce the number of choices
- Typically rule-based:
  - Perform selection early (reduces the number of tuples)
  - Perform projection early (reduces the number of attributes)
  - Perform most restrictive selection and join operations before other similar operations.
- Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

# Query Optimization

- Introduction
- Transformation of Relational Expressions
- Optimization Algorithms
- Statistics Estimation
- Summary

# Query Optimization

- Integral component of query processing
  - Why ?
- One of the most complex pieces of code in a database system
- Active area of research
  - E.g. XML Query Optimization ?
  - What if you don't know anything about the statistics
  - Better statistics
  - Etc …