# Query Optimization: Comprehensive Review

DSC 208R - Data Management for Analytics

## Introduction to Query Optimization

Query optimization is a fundamental aspect of Relational Database Management Systems (RDBMS).

### Goal of Query Optimization

- **Observation**: A given Logical Query Plan (LQP) can have several possible Physical Query Plans (PQPs) with vastly different runtime performances.
- **Ideal Goal**: To automatically find the optimal (fastest) PQP for a given LQP, without user involvement.
- **Realistic Goal**: It is often infeasible to guarantee the absolute optimal plan, but it is feasible to avoid many of the obviously bad plans.
- Optimized queries can run hundreds of thousands of times faster.

## Typical Query Optimizer Architecture

A typical query optimizer is composed of several key components:

- **Parser**: Takes the SQL Query as input and produces a Logical Query Plan (LQP).
- **Plan Enumerator**: This component explores alternative PQPs given an LQP. It finds alternate plans through:
  - Algebraic rewrite rules to modify one LQP into another logically equivalent LQP.
  - Physical operator selection choices for a fixed LQP.
- **Plan Cost Estimator**: Calculates an analytically determined runtime cost for a given PQP.
  - Physical operator selection and plan costing depend on hardware and require calibration runs when setting up the RDBMS.
- **Catalog**: Provides metadata (statistics, schema information) to the optimizer for cost estimation and plan generation.
- The Optimizer then passes the (optimized) Physical Query Plan to the Scheduler/Executor.

## Algebraic Rewrites

Modern RDBMSs implement numerous algebraic rewrites, which are sophisticated optimizations that RDBMS users are often unaware of. These rules transform one relational algebra expression into an equivalent, but potentially more efficient, one.

## Examples of Algebraic Rewrites

### 1. Selection Pushdown Through Join

This rule states that a selection operation that applies only to attributes of one relation in a join can be performed *before* the join. This reduces the size of the relation involved in the join, potentially speeding up the join operation.

- **Rule**: $\sigma_{p(A)}(R \bowtie S) = \sigma_{p(A)}(R) \bowtie S$
- **Condition**: $A \subseteq R.*$ (meaning attribute $A$ must be part of relation $R$).

### 2. Commuting of Select and Project

This rule allows the reordering of selection and projection operations. If a selection predicate only depends on attributes that are retained by a projection, the selection can be moved before the projection.

- **Rule**: $\sigma_{p(A)}(\pi_B(R)) = \pi_B(\sigma_{p(A)}(R))$
- **Condition**: $A \subseteq B$ (meaning attributes in predicate $A$ must be a subset of projected attributes $B$).

### 3. Algebraic Properties of the Join Operator

- **Commutativity**: The order of relations in a join does not affect the result.
  - **Rule**: $R \bowtie S = S \bowtie R$.
- **Associativity**: The grouping of relations in multiple joins does not affect the final result.
  - **Rule**: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.

## Join Order Optimization

- The ordering of joins in a query can dramatically alter the sizes of intermediate results and overall query runtimes.
- Join order optimization is a significant part of RDBMS query optimizers.
- The number of possible join orderings is exponential in the number of joins, making it nearly impossible for RDBMS users to hand-optimize.