# Using Indexes for Queries

Database Instance and Available Indexes:

<table>
<thead>
<tr>
<th>Table</th>
<th>Entries</th>
<th>Number of Entries per Block</th>
<th>Number of Blocks</th>
<th>Index Fields</th>
<th>Index Type</th>
<th>Keys per Node</th>
<th>Depth of B+ Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>10,000</td>
<td>10</td>
<td>1,000</td>
<td>accountId</td>
<td>B+ tree and ordered sequential file</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>Rental</td>
<td>1,000,000</td>
<td>100</td>
<td>10,000</td>
<td>lastName, zipcode, accountId, movieId, date, title, genre</td>
<td>B+ tree, hash, B+ tree, B+ tree, B+ tree, B tree, Hash</td>
<td>50, 100, 100, 100, 20, 100</td>
<td>3, 3, 3, 2, 4</td>
</tr>
<tr>
<td>Movie</td>
<td>10,000</td>
<td>20</td>
<td>500</td>
<td>movieId</td>
<td>B+ tree and ordered sequential file</td>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>

select * from Customer
where accountId = 101
select * from Customer
where accountld >= 101 and accountld < 300
select * from Customer where lastName < 'D'
Query Processing

- SQL Query
- Scanning, Parsing, Validating
- Expression Tree (algebra)
- Optimizer
- Execution Plan
- Query Code Generator
- Query Code
- DB Runtime Processor
- Result
Query Optimization

- Query optimization determines the most efficient (or sufficiently efficient) process for executing the query.

- Optimization reorganizes an expression tree for a query using algebraic transformation rules.

- Information used:
  - implementation techniques for algebra operators
  - algebraic transformation rules
  - heuristic rules
  - cost estimates
Algebraic Transformations

- Commutative: $\sigma \bowtie x \cup \cap$
- Associative: $\bowtie x \cup \cap$

• Cascades of select:
  - $\sigma_{c_1 \text{ AND } c_2 \text{ AND } \ldots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots \sigma_{c_n}(R))\ldots))$
  - selects can then be commuted

• Cascades of project:
  - $\pi_{l_1}(\pi_{l_2}(\ldots \pi_{l_n}(R))\ldots)) \equiv \pi_{l_1}(R)$
  - projects cannot be commuted
Algebraic Transformations

- Commuting select and project:

  \[ \pi_{A_1,A_2,\ldots,A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,\ldots,A_n}(R)) \]

  valid when selection condition only involves attributes in projection list

- Converting select/cross-product into join:

  \[ \sigma_c(R \times S) \equiv R \bowtie_c S \]
Algebraic Transformations

• Commuting select and join or cross-product:
  \[ \sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S \]
  valid when selection condition involves only attributes in R

• Commuting project with join or cross-product:
  \[ \pi_L(R \bowtie_c S) \equiv (\pi_{A_1,A_2,\ldots,A_n}(R)) \bowtie_c (\pi_{B_1,B_2,\ldots,B_n}(S)) \]
  where \(A_1,\ldots,A_n \subseteq L\) are attributes of \(R\),
  and \(B_1,\ldots,B_n \subseteq L\) are attributes of \(S\),
  and \(A_1,\ldots,A_n, B_1,\ldots,B_n\) are involved in \(C\)
  (slightly more complicated if \(C\) contains attributes not in \(L\))
Optimization Heuristics

- Break up conjunctive select conditions into a cascade of selects
  - more flexibility in moving selects

- Push selects as early as possible, using commutativity of select with other operators

- Reorder sub-expressions such that most restrictive selects are done first

- Combine select/cross-product into join

- Push projects as early as possible
  - keep only necessary attributes
Example Query Plan

Query 1: Query cost (relative to the batch): 100.00%
Query text: SELECT CompanyName, orders.OrderId, ProductName, SUM(order_details.unitprice * order_details.quantity) FROM order_details, orders, customers, products WHERE...
Optimization Processor

- Enumerate the query plans
  - Apply algebraic transformations & heuristics to get all reasonable trees

- Estimate the cost of each plan
  - Account for size of relations, available indexes, information about file layout, info about value distributions …

- Choose the best (fastest) plan
Example Query

```
select e.lname, e.fname, w.pno, w.hours
from employee e, works_on w
where e.ssn = w.essn and w.hours > 20;
```
Example Query

\[ \pi_{e.\text{name}, e.\text{fname}, w.\text{pno}, w.\text{hours}} \]

\[ \sigma_{w.\text{hours} > 20} \]

\[ \sigma_{e.\text{ssn}=w.\text{ssn}} \]

Heuristic: Conjunctive select \rightarrow cascade of selects

Note: selects could commute
Example Query

\[ \pi_{e.lname, e.fname, w.pno, w.hours} \]

Heuristic:
Combine select and cross \( \rightarrow \) join

\[ \sigma_{w.hours > 20} \]

\[ \bowtie_{e.essn = w.essn} \]

Employee \( e \)

\( \text{works\_on } w \)
Example Query

\[ \pi_{\text{e.lname, e.fname, w.pno, w.hours}} \]

Heuristic: Push selects as early as possible

\[ \bowtie_{\text{e.essn} = \text{w.essn}} \]

Employee e

\[ \sigma_{\text{w.hours} > 20} \]

works_on w
Example Query

\[ \pi_{e.lname, e.fname, w.pno, w.hours} \]

\[ \pi_{e.ssn, e.lname, e.fname, w.essn, w.pno, w.hours} \]

First cascade projects to get attributes needed to push through join

Heuristic: Push projects as early as possible

Employee e

\[ \bowtie_{e.ssn=w.essn} \]

\[ \sigma_{w.hours > 20} \]

works_on w
Example Query

Heuristic: Push projects as early as possible

\( \Pi_{e.lname, e.fname, w.pno, w.hours} \)

\( \bowtie e.ssn = w.essn \)

\( \Pi_{w.essn, w.pno, w.hours} \)

\( \sigma_{w.hours > 20} \)

Employee e

Push project through join

works_on w
Example Query

Heuristic:
Push projects as early as possible

$\pi_{e.lname, e.fname, w.pno, w.hours}$

$\exists e.essn=w.essn$

Push project through select

$\pi_{e.essn, e.lname, e.fname}$

$\sigma_{w.hours > 20}$

Employee e

$\pi_{w.essn, w.pno, w.hours}$

Works_on w
Example Query

- At this point we’ve generated nine different query plans for the same query.
- Adding strategies/algorithms for implementing individual operations would add even more potential plans.
- Next Step: Estimate the cost of executing each plan.