Query Execution

SQL Statement

Compiler

Query Execution Plan

Runtime System

Result (Relation)
• SQL is declarative, for runtime system it has to be translated into something procedural
• DBMS first translates SQL into an internal representation
• Common approach is translation into the relational algebra
Canonical Translation

- Standard translation of SQL into relational algebra
- Algebra expression are often represented graphically
- Example

```sql
select A1, ..., An
from R1, ..., Rm
where p
```
Optimizer

- Canonical plan not efficient, e.g. cartesian product

- DBMS has optimizer to transform the plan into an efficient form

- Finding the / one optimal plan very difficult: still research topic
Optimizer (2)

• User ↔ Query optimization?

• Users can
  o see the generated plan
  o analyse the generated plan
  o (where appropriate) reformulate queries or give DBMS hints for query execution
Visualization of plans

e.g. PostgreSQL:
Visualization of plans
e.g. Hyper:

Query Plan
Show Information: All / None

- Attributes
- Cardinalities
- Criteria
- Predicates
- Restrictions
- Residuals
- Outputs

```
sort
  v desc
  v2 desc
  cardinality: 135000

  Γ
  v3
  cardinality: 135000

  groupjoin
  cardinality: 150000

  customer
  cardinality: 150000

  orders
  cardinality: 135000
```

Legend
Query Optimization Basics

- Query optimization is cost based
- Execution time estimation with the help of cost models and statistics
- Application of heuristics far too expensive to look at all possible plans
- Two layers of optimization:
  - Logical layer
  - Physical layer
Logical Layer

• **Starting point**: expression of the relational algebra (result of the canonical translation)

• **Optimization**: transformation into equivalent expressions (with faster execution time)

• **Goal of the transformations**: Output (result) of the single (algebra) operations preferably small
Logical Layer (2)

Basic techniques - rules:

• Break selections
• Shift selections 'down' in the plan
• Combine selections and cartesian products to join
• Determine the join order
• Insert projections
• Shift projektions 'down' in the plan
Example Query

```sql
select S.Name, P.Name
from Students S, attend a, Lectures L, Professors P
where S.StudNr = a.StudNr
and a.LectNr = L.LectNr
and L.Given_by = P.PersNr
and S.Semester > 4
and P.Name = 'Sokrates';
```
Query Plan

\[ \pi_{S.Name, P.Name} \]
\[ \sigma_{P.Name = 'Socrates'} \]
\[ \sigma_{S.Semester > 4} \]
\[ L.Given\_by = P.PersNr \]
\[ a.LectNr = L.LectNr \]
\[ a.StudNr = S.StudNr \]
\[ P \]
\[ L \]
\[ S \]

25-Jan-18
Optimized Query Plan

\[ \pi_{S.Name, P.Name} \]
\[ \Box_{a.StudNr=S.StudNr} \]
\[ \Box_{a.LectNr=L.LectNr} \]
\[ \sigma_{S.Semester > 4} \]
\[ \Box_{L.Given\_by = P.PersNr} \]
\[ \sigma_{P.Name = 'Sokrates'} \]
Physical Optimization

• Distinction between logical and physical algebra operators
• Physical algebra operators are the implementation of the logical ones
• Several physical operators for one logical operator available
• Optimization on the physical layer means:
  o Pick one of the physical operators
  o Decide whether to use indexes
  o Decide whether to materialize intermediate results
  o …
Implementation of Operators

• Selection:
  o Scan
  o Indexscan

• Join:
  o Nested-Loop-Join
  o (Sort-)Merge-Join
  o Index-Join
  o Hash-Join
Cost model

Algebra expression \rightarrow \text{Cost model} \rightarrow \text{Execution costs}

Index information \rightarrow \text{Cost model} \rightarrow \text{Cluster information}

DB cardinalities \rightarrow \text{Cost model} \rightarrow \text{attribute distribution}
Cost estimation

• Selectivity
  fraction of qualifying tuples of an operation high selectivity means small fraction
• Estimate of selectivity through
  o Formula
  o Sampling
• Selectivity cost
• Join cost
• Join order
  \( \rightarrow \) therefore you need statistics
When to gather statistics?

When to update statistics?

Excerpt of IBM DB2 Manual:

• after data has been loaded into a table and appropriate indexes have been built
• after a new index for a table has been built
• after a table has been reorganized with REORG
• after a table and the corresponding indexes have been changed considerably via UPDATE-, INSERT- or DELETE-operations
• after executing the command REDISTRIBUTE DATABASE PARTITION GROUP
Which kind of statistics are there?

Excerpt of the IBM DB2 Manual:

SYSCAT: read-only view – System catalog
SYSSTAT: updatable view – Statistic data

• Table statistics (SYSCAT/SYSSTAT.TABLES)
• Column statistics (SYSCAT/SYSSTAT.COLUMNS)
• Statistics for groups of columns (SYSCAT/SYSSTAT.COLGROUPS)
• distribution statistics for columns (SYSCAT/SYSSTAT.COLDIST)
• distribution statistics for groups of columns (SYSCAT/SYSSTAT.COLGROUPDIST / COLGROUPDISTCOUNTS)
• Index statistics (SYSCAT/SYSSTAT.INDEXES)
Summary

- Query execution and optimization are important tasks of a database system.
- Also, even users should know about it as design decisions and query formulation influence the performance of a DBMS.
New Developments

• Main Memory Database Systems, (https://en.wikipedia.org/wiki/List_of_in-memory_databases), e.g. Times Ten (Oracle) VoltDB (some database scientists, open source) Monet DB (CWI, Amsterdam, open source) SAP HANA HYPER (Informatics, TUM)

• Column Store Database Systems, (https://en.wikipedia.org/wiki/List_of_column-oriented_DBMSes), e.g. C-Store / Vertica (HP) Monet DB (CWI, Amsterdam, open source) SAP HANA HYPER (Informatics, TUM)
Column Stores

What is a column-store?

row-store

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Product</th>
<th>Customer</th>
<th>Price</th>
</tr>
</thead>
</table>

+ easy to add/modify a record
- might read in unnecessary data

= suitable for read-mostly, read-intensive, large data repositories

column-store

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Product</th>
<th>Customer</th>
<th>Price</th>
</tr>
</thead>
</table>

+ only need to read in relevant data
- tuple writes require multiple accesses
## Row Store versus Column Store

<table>
<thead>
<tr>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product</strong></td>
</tr>
<tr>
<td><strong>Customer</strong></td>
</tr>
<tr>
<td><strong>Price</strong></td>
</tr>
<tr>
<td><strong>Branch</strong></td>
</tr>
<tr>
<td><strong>…”</strong></td>
</tr>
<tr>
<td>Mobile</td>
</tr>
<tr>
<td>Kemper</td>
</tr>
<tr>
<td>345</td>
</tr>
<tr>
<td>Schwabing</td>
</tr>
<tr>
<td>“…”</td>
</tr>
<tr>
<td>Radio</td>
</tr>
<tr>
<td>Mickey</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>Bogenhausen</td>
</tr>
<tr>
<td>“…”</td>
</tr>
<tr>
<td>Mobile</td>
</tr>
<tr>
<td>Minnie</td>
</tr>
<tr>
<td>233</td>
</tr>
<tr>
<td>Schwabing</td>
</tr>
<tr>
<td>“…”</td>
</tr>
<tr>
<td>Fridge</td>
</tr>
<tr>
<td>Urmel</td>
</tr>
<tr>
<td>240</td>
</tr>
<tr>
<td>Augsburg</td>
</tr>
<tr>
<td>“…”</td>
</tr>
<tr>
<td>Beamer</td>
</tr>
<tr>
<td>Bond</td>
</tr>
<tr>
<td>740</td>
</tr>
<tr>
<td>London</td>
</tr>
<tr>
<td>“…”</td>
</tr>
<tr>
<td>Mobile</td>
</tr>
<tr>
<td>Lucie</td>
</tr>
<tr>
<td>321</td>
</tr>
<tr>
<td>Bogenhausen</td>
</tr>
<tr>
<td>“…”</td>
</tr>
</tbody>
</table>
## Row Store versus Column Store

<table>
<thead>
<tr>
<th>Product</th>
<th>Customer</th>
<th>Price</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Product</td>
<td>ID</td>
<td>Customer</td>
</tr>
<tr>
<td>0</td>
<td>Mobile</td>
<td>0</td>
<td>Kemper</td>
</tr>
<tr>
<td>1</td>
<td>Radio</td>
<td>1</td>
<td>Mickey</td>
</tr>
<tr>
<td>2</td>
<td>Mobile</td>
<td>2</td>
<td>Minnie</td>
</tr>
<tr>
<td>3</td>
<td>Fridge</td>
<td>3</td>
<td>Urmel</td>
</tr>
<tr>
<td>4</td>
<td>Beamer</td>
<td>4</td>
<td>Bond</td>
</tr>
<tr>
<td>5</td>
<td>Mobile</td>
<td>5</td>
<td>Lucie</td>
</tr>
</tbody>
</table>
Compression

In particular possibilities of compression are interesting:

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Product</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Word</td>
<td>ID</td>
</tr>
<tr>
<td>0</td>
<td>Augsburg</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Beamer</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Bogenhausen</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Mobile</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Fridge</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>London</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Radio</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Schwabing</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NoSQL

No SQL - Not only SQL

Characteristics: Schema-free, web scale (but sacrificing ACID), distributed (scale-out), usually key-value store (hash tables: key and pointer to the value), specific data, e.g. graphs

CAP Theorem:
• Consistency
• Availability
• Partition Tolerance

Only two of the three goals can be achieved
NoSQL cont.

Short explanation:
http://www.youtube.com/watch?v=pHAItWE7QMU&list=PLB9uLawXQoggpG9MGz5v9wDodr9f_p4lg

'Debate‘ RDBMS – NoSQL (Mongo DB):
http://www.youtube.com/watch?v=b2F-DltXtZs