VIT Chennai, 29th September 2021

Invited Lecture

as part of the series 'Knowledge Representation and Reasoning'

Auditory Introduction to Query Processing in Semantic Web Databases

Professor Dr. rer. nat. habil. Sven Groppe

https://www.ifis.uni-luebeck.de/index.php?id=groppe
Stations of my **academic life**

- Study of CS ‘96-‘02
- Dr. rer. nat. ‘05

Lübeck
- PD habil. ‘11
- Prof. ‘19

Paderborn

Innsbruck
- Postdoc ‘05-‘07

Three rabbits for illustrating deadlocks instead of dining philosophers problem
Trying to impress (not sure if it's working)...

- > 122 publications
  - 93 co-authors
  - 41 publications (33%) are co-authored by bachelor/master students (being students at time of writing)
- 2 supervised dissertations, 3 current PhD students
- ≈ 82 bachelor/master/diploma thesis/student projects
- ≈ 60 contributors to LUPOSDATE
- > 35 lectures, in the areas of
  - Semantic Web
  - Cloud- and Web-Technologies
  - Mobile and Distributed Databases
  - Databases
  - Algorithms and Datastructures
  - Compiler
Trying to impress (not sure if it's working)...

- 6 third-party fundings (≈ € 1M)
- **Scientific Services**
  - **Workshop Chairs**
    - Semantic Big Data (SBD) @ SIGMOD (2016 - 2020)
    - Big Data in Emergent Distributed Environments (BiDEDE) @ SIGMOD (2021 - 2021)
    - Very Large Internet of Things (VLIoT) @ VLDB (2017 - 2021)
    - Web Data Processing and Reasoning (WDPAR) @ KI 2018
  - **General Chair**
    - International Semantic Intelligence Conference (ISIC) 2021 - 2022
  - many other scientific services
    - ≈ 90 PC memberships
    - reviewer of ≈ 30 journals
    - editor of 3 journals
Trying to impress (not sure if it's working)...

Detailed Information available at
https://www.ifis.uni-luebeck.de/~groppe
Agenda: Introduction to Query Processing in Semantic Web Databases

- **SPARQL** Query Language
  - Transformation to *Relational Algebra* Expression/Operatorgraph
- **Sound of Databases**
- **RDF3X Indices**
- **Merge Join**
- **Sideways Information Passing (SIP) in**
  - Merge Joins
  - RDF3X B⁺-Tree
- **Hash Join**
- **Inspiring Sounds**
Every data model (here Semantic Web) has its own set of languages (data, query, rule, ...)
Recap: SPARQL Query and its Result

**SPARQL Query:**

```
SELECT ?person ?name
WHERE {  
?article rdf:type bench:Article .  
?article dc:creator ?person .  
?person foaf:name ?name .  
}
```

**Query Graph:**

![Query Graph Diagram]

**RDF Graph:**

![RDF Graph Diagram]

**Solutions:**

1. ?article=article1  
   ?person=person1  
   ?name=„Martin“

2. ?article=article1  
   ?person=person2  
   ?name=„Martina“
Transformation to Relational Algebra Expression

```
SELECT ?person ?name
WHERE {
    ?article rdf:type bench:Article.
    ?article dc:creator ?person.
    ?person foaf:name ?name.
}
```

Diagram:
- `π{?person, ?name}`
- `⋈`
Architecture of a Semantic Web Database Management System
Semantic Web DBMS **LUPOSDATE**

**Support of:**
- SPARQL Queries
- RIF Rules
- RDF Schema
- OWL (via OWL2RL in RIF)

**Indexing:**
- Stream Processing
- Main memory for small datasets
- Disk-based for large datasets
  - RDF3X
- Cloud: HBase
- P2P

**Visualizations:**
- Visual Editor
  - Queries (SPARQL)
  - Rules (RIF)
  - Data (RDF) in
    - 2D and
    - 3D
  - Logical Optimization Rules
- Summaries of RDF Data
- Operator graph
- Processing of Queries and Rules
- Optimization Steps

**Extra:**
- Parallel Processing
- Distributed Processing
- Cloud Computing
- Mobile Computing
- P2P for Internet of Things
- Compression of RDF Data
- Embedding of SW Languages in Programming Languages
- Speeding up by FPGAs
LUPOSDATE3000

- Successor of LUPOSDATE
- Multiplatform Database
  - programming language **Kotlin**
    - targets: JVM, JS, native binaries (win/linux/macos/iOS/...)
- Focus on:
  - parallel databases
  - distributed databases for Internet-of-Things
  - Semantic Web: RDF, SPARQL
- Planned:
  - RDFS/OWL Inference
  - multi-model support

Open Source: https://github.com/luposdate3000/luposdate3000
Sonification of Triple Pattern Evaluation

- What do you hear?  
  - An ascending scale (in pitch)
- We learn: Semantic Web data often stored in indices  
  - Output is pre-sorted  
  - Maximize usage of fast merge joins
Sonification

- use of nonspeech audio to convey information [...] for the purposes of facilitating communication or interpretation

- Examples
  - brain activity
  - from deep space to cancer research
  - Sound of Sorting
  - ...

- Motivation
  - reduces barriers for people with poor education
  - presents multidimensional data in an easy-to-understand way
  - additional medium helps with learning new information
  - helps humans with visual impairments
Sound of Databases

- useful mapping for understanding the operator algorithm (or at least its input & output):
  - each operator plays another instrument (easy to distinguish from the other)
  - value of intermediate solution maps to pitch
- mapping for melodies: operator depth maps to pitch
- Online: https://www.ifis.uni-luebeck.de/~groppe/soundofdatabases/
Please **guess** the kind of operator!

- **Solution: Selection/Filter**

- **not all** intermediate results **are passing** the filter operator
# RDF3X Indices for Triple Pattern Evaluation

<table>
<thead>
<tr>
<th>Triple Pattern</th>
<th>1st Alternative</th>
<th>2nd Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prefix</td>
<td>Sort</td>
</tr>
<tr>
<td>?s f:p f:o</td>
<td>POS</td>
<td>?s</td>
</tr>
<tr>
<td>f:s f:p ?o</td>
<td>SPO</td>
<td>?o</td>
</tr>
<tr>
<td>f:s f:p f:o</td>
<td>SPO</td>
<td>-</td>
</tr>
</tbody>
</table>
B+-tree

Search for prefix
rdf:type bench:Article

P
O
S

P
O
S

rdf:type
bench:Article
article2

rdf:type
bench:Article
article4

rdf:type
bench:Article
article1

rdf:type
bench:Article
article2

rdf:type
bench:Article
article3

rdf:type
bench:Article
article4

rdf:type
bench:Article
article5
Joining the Result of 2 Triple Patterns

- What to learn?
  - The result is again sorted if merge join is used
Equi-Join

- Following join algorithms are only suitable for Equi-Joins
- \( R \bowtie_C S = \sigma_C (R \times S) \), where \( C \) is join condition of the form:
  - \( r = s \), where \( r \) is attribute of \( R \) and \( s \) is attribute of \( S \)
  - \( C_1 \land C_2 \), where \( C_1 \) and \( C_2 \) are join conditions

### Set Semantics

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Multi-Set Semantics

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Merge Join

- **Precondition**: Input sorted according to join columns
  - Otherwise a preceding sort phase necessary → "Sort Merge Join"
- **Algorithm**: step-by-step comparison of both sorted (input) relations
  - If it cannot be joined, then it will be read from the relation with the currently smaller value, because this smaller value does not occur any more in the other relation (utilizing the sorting)
**Merge Join as Iterator**

```java
open() {
    R.open(); r = R.next();
    S.open(); s = S.next();
    B = {} × {}; B.open()
}

close() {
    R.close()
    S.close()
}

next() {
    IF(B.hasNext()) RETURN B.next()
    IF(s == null ∨ r == null) RETURN null
    WHILE(\(\pi_{\text{Join Attributes}}(s) \neq \pi_{\text{Join Attributes}}(r)\)){
        IF(\(\pi_{\text{Join Attributes}}(s) < \pi_{\text{Join Attributes}}(r)\)){
            s = S.next()
            IF(s == null) RETURN null
        } ELSE { // r < s (see WHILE- and IF-condition!)
            r = R.next()
            IF(r == null) RETURN null
        }
    }
    cmp = \(\pi_{\text{Join Attributes}}(s)\) // = \(\pi_{\text{Join Attributes}}(r)\) (see WHILE-condition)
    ts = {}; WHILE(\(\pi_{\text{Join Attributes}}(s) == \text{cmp}\)) { ts = ts U s; s = S.next() }
    tr = {}; WHILE(\(\pi_{\text{Join Attributes}}(r) == \text{cmp}\)) { tr = tr U r; r = R.next() }
    B = ts × tr; B.open()
    RETURN B.next()
}
```

**Runtime complexity for**
- Input without duplicates in join columns: \(O(R + S)\)
- Input with duplicates in join columns: \(O(R × S)\)
Merge Join 1/11

- Result $R \bowtie_{R.lang = S.lang} S$:

- Buffer:
- Input:

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>lang</td>
</tr>
<tr>
<td>hallo</td>
<td>de</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
</tr>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
</tbody>
</table>

Precondition: Inputs must be sorted according to join columns
Merge Join 2/11

- **Result** \( R \bowtie_{R.lang=S.lang} S \):

- **Buffer:**

  \[ \begin{array}{|c|c|} \hline
  \text{first} & \text{lang} \\
  \hline
  \text{hallo} & \text{de} \\
  \text{de} & \text{Student} \\
  \hline
  \end{array} \]

  Load rows with smallest values in join columns into buffer

- **Input:**

  \[ \begin{array}{|c|c|} \hline
  \text{first} & \text{lang} \\
  \hline
  \text{hello} & \text{en} \\
  \text{hi} & \text{en} \\
  \text{hola} & \text{sp} \\
  \hline
  \end{array} \quad \begin{array}{|c|c|} \hline
  \text{lang} & \text{second} \\
  \hline
  \text{en} & \text{student} \\
  \text{en} & \text{collegian} \\
  \text{cn} & \text{daxuesheng} \\
  \hline
  \end{array} \]
Merge Join 3/11

- **Result** \( R \bowtie_{R.lang=S.lang} S \):

- **Buffer:**

\[
R \bowtie_{R.lang=S.lang} S
\]

<table>
<thead>
<tr>
<th>first</th>
<th>R.lang</th>
<th>S.lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
</tbody>
</table>

Compute **join result** for current content of buffer

- **Input:**

\[
R \\
S
\]

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hola</td>
<td>sp</td>
<td>daxuesheng</td>
</tr>
</tbody>
</table>
Merge Join 4/11

- Result $R \bowtie_{R.lang = S.lang} S$:

<table>
<thead>
<tr>
<th>first</th>
<th>$R$.lang</th>
<th>$S$.lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
</tbody>
</table>

- Emit join row

- Buffer:

- Input:

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>lang</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
</tr>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
<tr>
<td>lang</td>
<td>second</td>
</tr>
<tr>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>cn</td>
<td>daxuesheng</td>
</tr>
</tbody>
</table>
### Merge Join 5/11

- **Result** $R \bowtie_{R.lang=S.lang} S$:

<table>
<thead>
<tr>
<th>first</th>
<th>$R$.lang</th>
<th>$S$.lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
</tbody>
</table>

- **Buffer:**

$R$

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>en</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
</tr>
</tbody>
</table>

$S$

<table>
<thead>
<tr>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

Load rows with smallest values in join columns into buffer

- **Input:**

$R$

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
</tbody>
</table>

$S$

<table>
<thead>
<tr>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>daxuesheng</td>
</tr>
</tbody>
</table>
Merge Join 6/11

- Result $R \bowtie_{R\text{.lang}=S\text{.lang}} S$:

<table>
<thead>
<tr>
<th>first</th>
<th>$R$\text{.lang}</th>
<th>$S$\text{.lang}</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
</tbody>
</table>

- Buffer:

<table>
<thead>
<tr>
<th>first</th>
<th>$R$\text{.lang}</th>
<th>$S$\text{.lang}</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

Compute join result as cartesian product $\times$ of the buffer

- Input:

$R$

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
</tbody>
</table>

$S$

<table>
<thead>
<tr>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>daxuesheng</td>
</tr>
</tbody>
</table>
Merge Join 7/11

- Result \( R \bowtie_{R.lang = S.lang} S \): 

<table>
<thead>
<tr>
<th>first</th>
<th>R.lang</th>
<th>S.lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

Emit join rows

- Buffer:
- Input:

\[
R \quad S
\]

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>sp</td>
<td></td>
</tr>
<tr>
<td>cn</td>
<td>daxuesheng</td>
<td></td>
</tr>
</tbody>
</table>
### Merge Join 8/11

- **Result** $R \bowtie_{R.lang = S.lang} S$:

<table>
<thead>
<tr>
<th>first</th>
<th>$R$.lang</th>
<th>$S$.lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

- **Buffer:**

  $S$

<table>
<thead>
<tr>
<th>lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>daxuesheng</td>
</tr>
</tbody>
</table>

  Load rows with smallest values in join columns into buffer

- **Input:**

  $R$

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
</tbody>
</table>
Merge Join 9/11

- **Result** \( R \bowtie_{R\.lang = S\.lang} S: \)
  
<table>
<thead>
<tr>
<th>first</th>
<th>( R.lang )</th>
<th>( S.lang )</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

- **Buffer:** Discard buffer

- **Input:**
  
<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>hola</td>
<td>sp</td>
</tr>
</tbody>
</table>
## Merge Join 10/11

### Result \( R \bowtie_{R.lang = S.lang} S \):

<table>
<thead>
<tr>
<th>first</th>
<th>( R ).lang</th>
<th>( S ).lang</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

### Buffer:

- \( R \) to buffer
- Load rows with smallest values in join columns into buffer

### Input:
**Merge Join 11/11**

- **Result** \( \mathbf{R} \bowtie_{\mathbf{R}.\text{lang}=\mathbf{S}.\text{lang}} \mathbf{S} \):  

<table>
<thead>
<tr>
<th>first</th>
<th>( \mathbf{R}.\text{lang} )</th>
<th>( \mathbf{S}.\text{lang} )</th>
<th>second</th>
</tr>
</thead>
<tbody>
<tr>
<td>hallo</td>
<td>de</td>
<td>de</td>
<td>Student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hello</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>student</td>
</tr>
<tr>
<td>hi</td>
<td>en</td>
<td>en</td>
<td>collegian</td>
</tr>
</tbody>
</table>

- **Buffer**: Discard buffer
- **Input**:
Merge Join

\[ \pi_{\text{?person, ?name}} \]

\( \bowtie \) Hash

\( \bowtie \) Merge

?article rdf:type bench:Article

?article dc:creator ?person

?person foaf:name ?name

Search for prefix dc:creator in index sorted according to P S O:

?article = article1 = ?article = article1 ?person=p1
?article = article2 = ?article = article2 ?person=p2
?article = article3
?article = article4
?article = article5
... ?article = article5 ?person=p1
...
Sideways Information Passing (SIP) in RDF3X B⁺-Tree
SIP in $B^+$-Tree

- Too much overhead if the inner nodes are searched every time and the value to search for is close to the current one
- **Heuristic:** If the SIP value is not in the current and in the next leaf node, then search via the inner nodes (starting from the leaf node to the root via the previous search path)
Sideways Information Passing (SIP)

```java
open() {
    R.open(); r = R.next(); R.close()
    S.open(); s = S.next(); S.close()
    B = {} × {}; B.open()
}

close() {
    R.close()
    S.close()
}

next(sip) {
    // modified for SIP! Return result ≥ sip (not guaranteed)!
    IF(B.hasNext(sip)) RETURN B.next(sip)  // SIP!
    IF(s == null ∨ r == null) RETURN null
    WHILE(πJoin Attributes(s) ≠ πJoin Attributes(r) ∨
        πJoin Attributes(r) < sip ∨ πJoin Attributes(s) < sip){  // SIP!
        IF(πJoin Attributes(s) < πJoin Attributes(r)){
            s = S.next( max(πJoin Attributes(r), sip) )  // SIP!
            IF(s == null) RETURN null
        } ELSE {
            r = R.next( max(πJoin Attributes(s), sip) )  // SIP!
            IF(r == null) RETURN null
        }
    }
    cmp = πJoin Attributes(s)  // = πJoin Attributes(r) (see WHILE-condition)
    ts = {}; WHILE(πJoin Attributes(s) == cmp) { ts = ts U s; s = S.next() }
    tr = {}; WHILE(πJoin Attributes(r) == cmp) { tr = tr U r; r = R.next() }
    B = ts × tr; B.open()
    RETURN B.next()  
```
Sideways Information Passing (SIP) with several Merge Joins 1/2

1. 

2. 

- Merge $R.j = T.j$
- Merge $R.j = S.j$
- Merge $T.j = U.j$

<table>
<thead>
<tr>
<th>$R.$</th>
<th>$S.$</th>
<th>$T.$</th>
<th>$U.$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>$j$</td>
<td>$j$</td>
<td>$j$</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>1001</td>
<td>2000</td>
<td>50</td>
</tr>
<tr>
<td>1000</td>
<td>1002</td>
<td>4000</td>
<td>3000</td>
</tr>
</tbody>
</table>

- $j = 1$
- $j = 1000$
- $j = 50$
- $j = 3$
- Next(1000)
- Next(50)
Sideways Information Passing (SIP) with several Merge Joins 2/2

3. Information next(1000) passes sideways from $S$ to $R$ and also over the root node to $T$ and $U$
RDF3X - Indexing Scheme for large-scale RDF triple stores

Prefix-Search in **Index** with (Prefix-)**Key**:

- **PSO**
  - rdf:type ont:Car

Result of Triple Pattern:

- ?car
  - HL 100
  - HL 101
  - ... HL 999
  - ...

Search in **PSO** – **B**°-tree:

- search(rdf:type ont:Car*)

- next(\text{O} \geq \text{HL 999})

- Heuristics: continue search in inner nodes if not found in next leaf node!

**Complexity of Merge Join** $\Join_{\text{Merge}}$:

- Worst Case (duplicates): $O(|R| \times |S|)$
- without duplicates: $O(|R| + |S|)$
- with **sideways information passing**: $O(|R \bowtie S|)$

(assuming quasi-constant access in **B**°-tree)

Maximise usage of **merge joins** ⇒ 3!=6 indices for all possible sort orders of triples **SPO**: **SPO, SOP, PSO, POS, OSP, OPS**
(In-Memory) Hash Join/Index Join with Hash Table
Auditory Introduction to Query Processing in Semantic Web Databases

Institut für Informationssysteme | Prof. Dr. habil. S. Groppe

Index Join 1/2

<table>
<thead>
<tr>
<th>first</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>„hallo“</td>
<td>de</td>
</tr>
<tr>
<td>„hello“</td>
<td>en</td>
</tr>
<tr>
<td>„hi“</td>
<td>en</td>
</tr>
<tr>
<td>„salut“</td>
<td>fr</td>
</tr>
<tr>
<td>„hola“</td>
<td>sp</td>
</tr>
<tr>
<td>„ciao“</td>
<td>it</td>
</tr>
<tr>
<td>„nihao“</td>
<td>cn</td>
</tr>
</tbody>
</table>

Index for $S$
(e.g. B*-tree or hash table)

lang: de

second:

daxuesheng

lang: en
collegian

lang: fr
eduitant

lang: it
universitario

lang: it
studente
Index Join 2/2

- Runtime $O(\text{Indexing}(S) + |R| \times \text{Lookup(}\text{Index}(S)\text{)))$
  - $\text{Indexing}(S)$ is omitted in case of existing (primary/secondary) indices

<table>
<thead>
<tr>
<th></th>
<th>$O(\text{Indexing}(S))$</th>
<th>$O(\text{Lookup(}\text{Index}(S)\text{)))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasTable</td>
<td>$O(</td>
<td>S</td>
</tr>
<tr>
<td>$B^+$-Tree</td>
<td>$O(</td>
<td>S</td>
</tr>
</tbody>
</table>

With long keys, the time for hashing, comparing, etc. is significant, otherwise $O(1)$

$h$ Runtime for hash function (often linear to key length), $k$ Out degree $B^+$-tree node

$B^+$-tree can be created after sorting by $1 \times$ traversing the sorted entries
Summary and Conclusions

- **SPARQL Query Language**
  - Transformation to **Relational Algebra** Expression/Operatorgraph
- **Sound of Databases**
- **RDF3X Indices**
- **Merge Join**
- **Sideways Information Passing (SIP) in**
  - Merge Joins
  - **RDF3X B**⁺**-Tree**
- **Hash Join**
- **Inspiring Sounds**
Whole Pieces of Music from your Database

Start Sonification