Transparent inclusion, validation, and utilization of main memory domain indexes

Thanh Truong C., Tore Risch
Uppsala University
Sweden
“Many scientific applications involving, e.g., data mining, temporal queries, and spatial analyses, require customized indexing to improve performance”
How many index structures are there?

DOI=10.1145/280277.280279
How many index structures are used in DBMSs?

<table>
<thead>
<tr>
<th>Index structure</th>
<th>Oracle</th>
<th>MySQL</th>
<th>SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Btree</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Hash</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>R-tree</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>Trie</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>Bit-map</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Notes:
- Versions
  - Oracle 12c Release 1
  - SQL Server 2014
  - MySQL 5.6
- In MySQL, some storage engines permit some index types, but not all.
- The table does not count “Function based index”.
- In SQL Server, hash index is only available for in-memory tables.
In-memory databases

Hao Zhang, Gang Chen, Kian-Lee Tan, Meihui Zhang
In-Memory Big Data Management and Processing: A Survey,
Knowledge and Data Engineering, IEEE Transactions on
volumn 27, pages 1920 - 1948
DOI: 10.1109/TKDE.2015.2427795
<table>
<thead>
<tr>
<th>Systems</th>
<th>Data Model</th>
<th>Workloads</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-Store</td>
<td>relational (row)</td>
<td>OLTP</td>
<td>hashing, B⁺-tree, binary tree</td>
</tr>
<tr>
<td>Hekaton</td>
<td>relational (row)</td>
<td>OLTP</td>
<td>latch-free hashing, Bw-tree</td>
</tr>
<tr>
<td>HyPer/ScyPer</td>
<td>relational</td>
<td>OLTP, OLAP</td>
<td>hashing, balanced search tree, ART</td>
</tr>
<tr>
<td>SAP HANA</td>
<td>relational, graph, text</td>
<td>OLTP, OLAP</td>
<td>timeline index, CSB⁺-tree, inverted index</td>
</tr>
<tr>
<td>MemepiC</td>
<td>key-value</td>
<td>object operations, analytics</td>
<td>hashing, skip-list</td>
</tr>
<tr>
<td>MongoDB</td>
<td>document (json)</td>
<td>object operations, analytics</td>
<td>B-tree</td>
</tr>
<tr>
<td>RAMCloud</td>
<td>key-value</td>
<td>object operations</td>
<td>hashing</td>
</tr>
<tr>
<td>Redis</td>
<td>key-value</td>
<td>object operations</td>
<td>hashing</td>
</tr>
<tr>
<td>Bitsy</td>
<td>graph</td>
<td>OLTP</td>
<td>N/A</td>
</tr>
<tr>
<td>Trinity</td>
<td>graph</td>
<td>graph operations</td>
<td>N/A</td>
</tr>
<tr>
<td>Memcached</td>
<td>key-value</td>
<td>object operations</td>
<td>N/A</td>
</tr>
<tr>
<td>MemC3</td>
<td>key-value</td>
<td>object operations</td>
<td>N/A</td>
</tr>
<tr>
<td>TsCache</td>
<td>key-value</td>
<td>OLTP</td>
<td>hashing</td>
</tr>
<tr>
<td>M3R</td>
<td>key-value</td>
<td>analytics</td>
<td>N/A</td>
</tr>
<tr>
<td>Piccolo</td>
<td>key-value</td>
<td>analytics</td>
<td>N/A</td>
</tr>
<tr>
<td>Spark/RDD</td>
<td>RDD</td>
<td>analytics</td>
<td>N/A</td>
</tr>
<tr>
<td>Spark Streaming</td>
<td>RDD</td>
<td>streaming</td>
<td>N/A</td>
</tr>
<tr>
<td>Yahoo! S4</td>
<td>Event</td>
<td>streaming</td>
<td>hashing</td>
</tr>
</tbody>
</table>

How many index structures are used in In-memory databases?

Hao Zhang, Gang Chen, Kian-Lee Tan, Meihui Zhang
*In-Memory Big Data Management and Processing: A Survey*,
Knowledge and Data Engineering, IEEE Transactions on volume 27, pages 1920 - 1948
DOI: 10.1109/TKDE.2015.2427795
Why ?
Because it is very challenging
Here are some challenges C1,..,C5

- C1. Understanding the DB kernel
- C2. Re-implementing the datastructure
- C3. Integrating with other DB internal components
- C4. Extending query processor
- C5. Validating the index’s functionalities
Only database (kernel) expert can do it!
Solution?
Some extensible indexing frameworks

• GiST

• Extensible Indexing – Orcale 8i

• SP-GisT
Reviews

• These frameworks specifies coding conventions and primitives.

• **Solved** C1 - Understanding DB kernel -

• **Solved** C3 – Integrating with other kernel components
The remaining **unsolved** challenges

- **C2** - Re-implementing the index implementation
  It is not OK if the index implementation
  - has ownership.
  - Is available in binary.
  - or being very complex to re-implement, i.e; Judy-tries
  
- **C4** - Extending query processor

- **C5** - Validating the index’s functionalities
Our motto

Only database (kernel) expert can do it!

“It should not be necessary to be a database kernel expert to introduce a new domain index”
Our solution

• The paper title:
  “Transparent inclusion, utilization, and validation of main memory domain indexes”

• The paper itself
  - Transparent inclusion – to solve C1, C2, C3
    - no index implementation code changed.
  - Transparent utilization – to solve C4
    - automatically transforms queries to utilize the new added index.
  - Transparent validation – to solve C5
    - Automatically generates and executes queries to test the new added index

• The result:
The generalized extensible indexing framework: Main-memory eXternal Index Manager (Mexima).

• Website: [http://www.it.uu.se/research/group/udbl/mexima/](http://www.it.uu.se/research/group/udbl/mexima/)
How to introduce a new index?

- Grab the *index implementation* \((a)\)
- Study the public *index API* \((b)\)
- Write the *index driver* \((c)\)

(glue code) that interfaces Mexima and the index API

➡️ Compiled as dynamic library called as *index extension*
At the end of the day

/* Load main-memory BTREE index*/
load_extension("bt");
At the end of the day (cont.)

/* Create a table to store salaries of people given social security numbers*/
create function salary(Number ssn)->Number sl as stored;

/*create BTREE on sl*/
create_index(”salary”, ”sl”, ”BTREE”, ”);

/*Add data*/
set salary(8301318971) = 2000;
set salary(8501332978) = 3000;
... 
set salary(8001335978) = 4000;

/*Query*/
SELECT ssn, sl
FROM  Number ssn, Number sl
WHERE salary(ssn) = sl AND sl >= 3000;
Mexima

Query
- Query Processing
- Operations
- Mexima core

Mexima core

Extension driver
- Index implementation

Mexima interface = BAOs + SSFs
The index driver code contains

- Basic access operators (**BAOs**)
  - `create()`, `drop()`, `put()`, `delete()`, `get()`,

- and `map()` that scans the index by applying a specified mapper function on each index entry.

- implemented as C functions

(****details in the paper**)
The index driver code contains

- Special search functions (SSFs)
  - Examples:
    - interval search on B-trees: \texttt{bt\_select\_range()}
    - and proximity search on X-trees/R-trees: \texttt{xt\_proximity\_search()}
    - and KNN search on X-trees/R-trees: \texttt{xt\_knn\_search()}
  - Implemented as foreign functions (UDFs)

(*** details in the paper)
But it is not enough ... 

• How new index is utilized in query?
  • Option 1
    End-user can **manually** call a SSF in query by reformulating the query
  • Option 2
    End-user can express query **naturally**, but the query optimizer should be able to utilize the index.

The query processor should **transparently** transform the query to SSF if possible to utilize the index ➔ **SSF translation rules**.
SSF translation rules

• An SSF translation rule describes how query fragments are translated to a new format to expose SSFs.

• Examples

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>&gt;=, &lt;=</td>
<td>btree_select_range</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>&lt;=</td>
<td>xt_proximity_search</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>Knn</td>
<td>nil</td>
<td>xt_knn_search</td>
</tr>
</tbody>
</table>
Example - Table

- Table $\text{images}(id, hist)$
  - $id$, image’s identifier
  - $hist$, histogram as image’s feature vector
Example - Query 1

"Query 1 finds images q whose identifiers are between 30 and 100"

Input query

\[ Q1(q) : - \]
\[ \text{images}(q, _) \text{ AND } q \geq 30 \text{ AND } q \leq 100 \]

Intermediate query

\[ TQ1(q) : - \]
\[ (q, _) \text{ in } \text{btree_select_range( '#images', 0, 30, 100)} \]

SSF Btree range search

With Mexima, it is done by the following SSF translation rule

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
<th>pf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>( \geq, \leq )</td>
<td>btree_select_range</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>( \leq )</td>
<td>xt_proximity_search</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>Knn</td>
<td>\text{nil}</td>
<td>xt_knn_search</td>
<td>F</td>
</tr>
</tbody>
</table>
**Example - Query 1**

"Query 1 finds images whose identifiers are between 30 and 100"

**Input query**

\[ Q1(q) :\]
- \( \text{images}(q, _) \) \ AND
- \( q \geq 30 \) \ AND
- \( q \leq 100 \)

**Intermediate query**

\[ TQ1(q) :\]
- \( (q, _) \) \ in \ btree\_select\_range( \#'images', 0, 30, 100) \]

---

### Relation operators and SSF

<table>
<thead>
<tr>
<th>Index type</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>Nil</td>
<td>( \geq, \leq )</td>
<td>btree_select_range</td>
</tr>
</tbody>
</table>

---

**Form (i):**

\[ P(...iv,...) \ AND (iv r1 expression) \ AND (iv r2 expression) \ AND \ldots \ AND (iv rn expression) \]

Here,

- \( iv \) is a variable bound to an indexed column of table \( P(...) \). We say \( iv \) is an indexed variable.

- \( r_i \) are comparison operators in the set \( relop, r_i \in relop, \) where \( relop = \{=, <, >, \geq, \leq\} \).
Example - Query 2

"For a given image \( x \) find the images \( q \) whose feature vectors are closer than epsilon (\( \epsilon = 0.11 \))."

**Input query**

\[
Q2(x, q): \neg
\]

- \( \text{images}(x, \text{hist}_x) \) AND
- \( \text{images}(q, \text{hist}_q) \) AND
- \( \text{distance} (\text{hist}_x, \text{hist}_q) \leq 0.11 \)

**Intermediate query**

\[
TQ2(x, q): \neg
\]

- \( \text{image}(x, \text{hist}_x) \) AND
- \( (q, \text{hist}_q) \) in \( \text{xtree_proximity_search}('#'\text{images'},1, \text{hist}_x, 0.11) \) AND
- \( \text{distance} (\text{hist}_x, \text{hist}_q) \leq 0.11 \)

**MAP**

**SSF X-tree proximity search**

With Mexima, it is done by the following SSF translation rule

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
<th>pf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>( \geq, \leq )</td>
<td>\text{btree_select_range}</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>( \leq )</td>
<td>\text{xt_proximity_search}</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>Knn</td>
<td>\text{nil}</td>
<td>\text{xt_knn_search}</td>
<td>F</td>
</tr>
</tbody>
</table>
"For a given image \( x \) find the images \( q \) whose feature vectors are closer than epsilon (\( \epsilon = 0.11 \))."
Example - Query 3

”Find the $k = 10$ closest images compared to a given image $x$”

**Input query**

$$Q3(x, \text{hist}_x):-$$

$$\text{images}(x, \text{hist}_x) \text{ AND } \text{images}(q, \text{hist}_q) \text{ AND } (q, \text{hist}_q) \text{ in knn}($$

$\text{hist}_x, 10, \#'\text{images}')$$

**Intermediate query**

$$TQ3(x, \text{hist}_x):-$$

$$\text{image}(x, \text{hist}_x) \text{ AND } (q, \text{hist}_q) \text{ in } \text{xt_knn_search}($$

$\#'\text{images}', 1, \text{hist}_x, 10)$

**SSF X-tree KNN search**

With Mexima, it is done by the following SSF translation rule

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
<th>pf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>$\geq, \leq$</td>
<td>$\text{btree_select_range}$</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>$\leq$</td>
<td>$\text{xt_proximity_search}$</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>Knn</td>
<td>nil</td>
<td>$\text{xt_knn_search}$</td>
<td>F</td>
</tr>
</tbody>
</table>
**Example Query 3**

"Find the $k = 10$ closest images compared to a given image $x$"

**Intermediate query**

$TQ3(x, \ hist_x): -$

- $image(x, \ hist_x)$
- $(q, \ hist_q) \ in \ xt_knn_search \ (#'images', 1, \ hist_x, 10)$

**Input query**

$Q3(x, \ hist_x): -$

- $images(x, \ hist_x)$
- $images(q, \ hist_q)$
- $(q, \ hist_q) \ in \ knn(\ hist_x, 10, #'images')$

**Form (iii):**

$$P(...) AND (..iv..) in isf(......P...)$$

**SSF X-tree KNN search**

With Mexima, it is done by the following SSF translation rule

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>$\geq$, $\leq$</td>
<td>$btree_select_range$</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>$\leq$</td>
<td>$xt_proximity_search$</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>Knn</td>
<td>nil</td>
<td>$xt_knn_search$</td>
</tr>
</tbody>
</table>

**Table:**

- **Index type:**
  - B-tree
  - X-tree

- **Priority:**
  - 1
  - 2

- **Index sensitive function:**
  - Nil
  - distance
  - Knn

- **Relation operators:**
  - $\geq$, $\leq$
  - $\leq$
  - nil

- **SSF:**
  - $btree\_select\_range$
  - $xt\_proximity\_search$
  - $xt\_knn\_search$
Reviews of query fragment forms

**Form (i):**

\[ P(...iv,..) \land (iv \ r_1 \ expression) \land (iv \ r_2 \ expression) \land \ldots \land (iv \ r_n \ expression) \]

**Form (ii):**

\[ P(...iv,..) \land isf(...,iv,..) \ r_1 \ expression \land isf(...,iv,..) \ r_2 \ expression \land \ldots \land isf(...,iv,..) \ r_n \ expression \]

**Form (iii):**

\[ P(...,iv,..) \land (..,iv,..) \ in \ isf(.....,P,..) \]

**Form (iv):**

\[ P(...iv,..) \land F(iv) \ \text{relop expression} \]

**Form (v):**

\[ P(...,iv,..) \land F(isf(...,iv,..)) \ \text{relop expression} \]

**isf(...) relop expression**

**isf(...) LIKE expression**

---

**Oracle**

Advisor tools to suggest on reformulating the query to utilize indexing


---

**Mexima**

Transparently transformation to utilize indexing

Our solution

• The paper title:
  “Transparent inclusion, utilization, and validation of main memory domain indexes”

• The paper itself
   Transparent inclusion
    o no index implementation code changed.
   Transparent utilization
    o automatically transforms queries to utilize the new added index.
   Transparent validation
    o Automatically generates and executes queries to test the new added index
What to test?

• BAOs: correctness of BAOs

• SSFs
  — Correctness of SSFs
  — Correctness of SSF translation rules
BAO tester

- Automatically tests correctness of `put()`, `get()`, `delete()`, `map()`, and `drop()`.

- Index key generator as queries

<table>
<thead>
<tr>
<th>#</th>
<th>Idxtype</th>
<th>Index key type</th>
<th>Index KeyGenerators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>Number</td>
<td><code>select uniform_int(1000,0,10000)</code></td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>Vector-Number</td>
<td><code>select uniform_vec_real(1000,5,0,1)</code></td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>Vector-Number</td>
<td><code>select CSV_file_rows(“colorhistogram.csv”)</code></td>
</tr>
</tbody>
</table>
BAO tester (cont.)

• Populate generated data into
  – Table $I_{Table}(k, v)$, having index to test at column $k$
  – Table $R_{Table}(k,v)$, having Hash index at $k$

• Execute BAO tester algorithms (***)

• Validate $I_{Table}$ against $R_{Table}$

*** details in the paper
SSF tester - Ideas

• Create sample tables with and without the index

• Auto-generate validation queries

• Recall, SSF translation rules transform these queries

⇒ Same value returned if there is no index, or no matching SSF translation rules
SSF tester – Ideas (cont.)

- SSF parameter generators

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>SSF name</th>
<th>SSF parameter generator</th>
<th>SSF Parameter types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>btree_select_range</td>
<td><code>select l, u from Number l, Number u where l in uniform_int(100, 0.10000) and u in uniform_int(100,0,10000)</code></td>
<td>(Number, Number)</td>
</tr>
<tr>
<td>2</td>
<td>B-tree</td>
<td>btree_select_open</td>
<td><code>select u from Number u where u in uniform_int(100, 0.10000)</code></td>
<td>(Number)</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>xtree-proximity-search.</td>
<td><code>select x, d from Vector of Number x, Number d where x in uniform_vec_real(100.5,0,1) and d in uniform_real(100.0, 1.4)</code></td>
<td>(Vector of Number, Number)</td>
</tr>
<tr>
<td>4</td>
<td>X-tree</td>
<td>xtree_knn-search</td>
<td><code>select x, k from Vector of Number x, Number k where x in uniform_vec_real(100.5,0.1) and k in uniform_int(0.5)</code></td>
<td>(Vector of Number, Number)</td>
</tr>
</tbody>
</table>
It is getting more complicated!
SSF tester – Ideas (cont.)

• Join three index property tables
  – SSF translation rule table

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>priority</th>
<th>Index sensitive function</th>
<th>Relation operators</th>
<th>SSF parameter generator</th>
<th>pf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>1</td>
<td>Nil</td>
<td>&gt;=, &lt;=</td>
<td>btree select range</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>1</td>
<td>distance</td>
<td>&lt;=</td>
<td>xt proximity search</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>2</td>
<td>nil</td>
<td></td>
<td>xt knn search</td>
<td>F</td>
</tr>
</tbody>
</table>

  – Index key generator table

<table>
<thead>
<tr>
<th>#</th>
<th>Idxtype</th>
<th>Index key type</th>
<th>Index KeyGenerators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>Number</td>
<td>select uniform int(1000,0,10000)</td>
</tr>
<tr>
<td>2</td>
<td>X-tree</td>
<td>Vector-Number</td>
<td>select uniform_vec_real(1000,5,0,1)</td>
</tr>
<tr>
<td>3</td>
<td>X-tree</td>
<td>Vector-Number</td>
<td>select CSV_file_rows(&quot;colorhistogram.csv&quot;)</td>
</tr>
</tbody>
</table>

  – SSF parameter generator table

<table>
<thead>
<tr>
<th>#</th>
<th>Index type</th>
<th>SSF name</th>
<th>SSF parameter generator</th>
<th>SSF Parameter types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B-tree</td>
<td>btree_select_range</td>
<td>select l, u from Number l, Number u where l in uniform_int(100, 0,10000) and u in uniform_int(100,0,10000)</td>
<td>(Number, Number)</td>
</tr>
</tbody>
</table>
SSF tester – Validation queries

- Validation query matching Form (I)

<table>
<thead>
<tr>
<th>Formula</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select iv, v</code>&lt;br&gt;from <code>IT iv, Number v, T1 p1, T2 p2,..., Tm pm</code>&lt;br&gt;where <code>I_Table(iv, v)</code> and&lt;br&gt;(p1, p2, ...,pm) in (SPG) and&lt;br&gt;(iv r1 p1) and&lt;br&gt;(iv r2 p2) and&lt;br&gt;...&lt;br&gt;(iv rm pm);`</td>
<td>Here,&lt;br&gt;• SSF parameters types in the SSF parameter generator table are <code>T1,..., Tm</code> (Table 3)&lt;br&gt;• <code>IT</code> is the index key type in the index key generator table (Table 2)&lt;br&gt;• <code>SPG</code> is the SSF parameter generator (Table 3) for parameters <code>p1,...,pm</code>, and that <code>ri</code> are the <code>relops</code> in Form (i).</td>
</tr>
</tbody>
</table>

**Example**

```sql
select iv, v
from Number iv, Number v, Number p1, Number p2
where I_Table(iv, v) and
    (p1, p2) in (select l, u from Number l, Number u
        where l in uniform_int(100, 0,10000) and
        u in uniform_int(100,0,10000)) and
    iv >= p1 and iv <= p2;
```
SSF tester – Validation queries

- Validation queries matching Form (II)

**Formula**

```
select iv, v
from IT iv, Number v,
    T_1 p_1, T_2 p_2, ..., T_m p_m, T_j res
where I_table(iv, v) and
    (p_1, p_2, ..., p_m) in (SPG) and
    res = ISF (iv, p_1, ..., p_{j-1}) and
    (res r_1 p_j) and
    ... and
    (res r_m p_m);
```

**Descriptions**

Here,
- \( j \) is the arity of ISF()

**Example**

```
select iv, v
from Vector of Number iv, Number v, Vector of Number p1,
    Number p2, Number res
where I_Table(iv, v) and
    (p1, p2) in (select x, d from Number x, Number d
    where x in uniform_vec_real(100,5,0,1) and
def in uniform_real(100,0, 1.4)) and
    res = distance(iv, p1) and res <= p2;
```
SSF tester – Validation queries

• Validation queries matching Form (III)

### Formula

```
select iv, v
from IT iv, Number v,
    T_1p_1, T_2p_2,..., T_mp_m,
where I_table(iv, v)
    and
    (p_1, p_2, ...,p_m) in (SPG) and
    (iv, v) in ISF (p_1,...,p_m, I_Table)
```

### Example

```
select iv, v
from Vector of Number iv, Number v, Vector of Number p1,
    Number p2
where I_Table(iv, v) and
    (p_1, p_2) in (select x, k from Number x, Number k
        where x in uniform_vec_real(100,5,0,1) and k in
            uniform_int(0,5)) and
    (iv,v) in knn(p_1, p_2, '#Images');
```
Experiments
Experiment - Purposes

• Code size
  – Compare coding size to introduce some indexes between Mexima vs other extensible indexing frameworks
  – To show Mexima requires no code change, driver code (glue code) is small

• BAO overhead
  – Time to run a stand-alone index implementation
  – Time to run it when plugging into Mexima
  – To investigate the overhead = Penalty of using Mexima

• Impact of SSF translation rules
  – Time to run queries with/without SSF translation rules
  – To show the importance of query rewrite to utilize indexes
Experiment - Settings

• All performance experiments were repeated 10 times, from which the average figures were calculated after removing outlier results if any.

• The experiments were run under Windows 7 on an Intel (R) Core(TM) i5 760 @2.80GHz 2.93 GHz CPU with 8GB RAM, using the Visual Studio 10 32 bits C compiler.
Experiment – Code size

- Count number of code C/C++ lines of glue code vs other extensible indexing systems

<table>
<thead>
<tr>
<th></th>
<th>GiST</th>
<th>SP-GiST</th>
<th>Mexima</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>5031</td>
<td>--</td>
<td>116</td>
<td>43</td>
</tr>
<tr>
<td>KD-tree</td>
<td>--</td>
<td>572</td>
<td>118</td>
<td>5</td>
</tr>
<tr>
<td>R-tree</td>
<td>1133</td>
<td>--</td>
<td>120</td>
<td>9.5</td>
</tr>
<tr>
<td>Trie</td>
<td>--</td>
<td>580</td>
<td>120</td>
<td>5</td>
</tr>
</tbody>
</table>

- PostgreSQL version 9.3.5, [http://www.postgresql.org/ftp/source/v9.3.5/](http://www.postgresql.org/ftp/source/v9.3.5/)
- SP-GiST version 0.0.1, [https://www.cs.purdue.edu/spgist/](https://www.cs.purdue.edu/spgist/)
- Mexima, [http://www.it.uu.se/research/group/udbl/mexima](http://www.it.uu.se/research/group/udbl/mexima)

- Mexima requires
  - no code change to the index implementation
  - little coding effort for the driver (interface)
Experiment – BAO overhead

• The total time
\[ \text{tot} = \text{op} + \text{mc} + \text{ed} + \text{st} \]

• The Mexima overhead,
\[ o = \text{op} + \text{mc} + \text{ed} \]

• Breaking down the overhead
  – %op ?
  – %mc?
  – %ed?
## Experiment – BAO overhead (cont.)

<table>
<thead>
<tr>
<th>BAO</th>
<th>Index</th>
<th>op</th>
<th>%op</th>
<th>%mc</th>
<th>%ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>LH</td>
<td>0.56</td>
<td>51.7%</td>
<td>36.2%</td>
<td>12.1%</td>
</tr>
<tr>
<td></td>
<td>B-tree</td>
<td>0.53</td>
<td>52.3%</td>
<td>35.8%</td>
<td>11.9%</td>
</tr>
<tr>
<td></td>
<td>Judy-trie</td>
<td>0.54</td>
<td>52%</td>
<td>35.3%</td>
<td>11.7%</td>
</tr>
<tr>
<td>Get</td>
<td>LH</td>
<td>0.26</td>
<td>37.2%</td>
<td>47.1%</td>
<td>15.7%</td>
</tr>
<tr>
<td></td>
<td>B-tree</td>
<td>0.23</td>
<td>36.6%</td>
<td>47.6%</td>
<td>15.7%</td>
</tr>
<tr>
<td></td>
<td>Judy-trie</td>
<td>0.22</td>
<td>36%</td>
<td>48%</td>
<td>16%</td>
</tr>
<tr>
<td>Map</td>
<td>LH</td>
<td>0.07</td>
<td>32.1%</td>
<td>50.9%</td>
<td>17%</td>
</tr>
<tr>
<td></td>
<td>B-tree</td>
<td>0.07</td>
<td>34.4%</td>
<td>49.2%</td>
<td>16.4</td>
</tr>
<tr>
<td></td>
<td>Judy-trie</td>
<td>0.07</td>
<td>33.7%</td>
<td>49.7%</td>
<td>16.6%</td>
</tr>
<tr>
<td>Delete</td>
<td>LH</td>
<td>0.42</td>
<td>45%</td>
<td>41.3%</td>
<td>13.7%</td>
</tr>
<tr>
<td></td>
<td>B-tree</td>
<td>0.42</td>
<td>43.3%</td>
<td>42.5%</td>
<td>14.2%</td>
</tr>
<tr>
<td></td>
<td>Judy-trie</td>
<td>0.41</td>
<td>43.4%</td>
<td>42.5%</td>
<td>14.1%</td>
</tr>
</tbody>
</table>

- Data size = 5 million key/value pairs
- 1000 random inserts, lookups, deletes.
- The average overhead in microseconds per call

\[ \text{Overhead} < 0.6 \text{ microsecond} \]
Experiment – Overhead w.r.t data size

The stand-alone index implementations are always faster.

The overhead is independent of the database size.
Experiment – Impact of SSF translation rules

With SSF translation rules, queries run faster. It made indexes utilized.

Xt-trees on Proximity search

X-trees and R*-trees on KNN search
Experiment – Side notes

- Bugs found in the following used index implementations using Mexima tester
  - X-trees [1]
  - R*-trees [2]
  - B-trees [1]

- Comparisons
  - Judy-trie [ref] outperformed B-trees in get(), insert(), delete(), but not map()
  - For 2D – 4D, X-trees is as good as R*-trees
  - For higher dimension (9D), X-trees is applicable and scale

Conclusions & Future work

• Conclusions
  – The Mexima framework allows plugging-in of main-memory domain index implementations with ease
    • without code changes
    • a simple Mexima driver for BAOs and SSFs
    • declare index properties as queries
    • transparently, Mexima makes new indexes utilized
    • automatically generating and executing validation queries, Mexima validates correctness of BAOs and SSFs
  – Tool for testing and comparing indexes

• Future work
  – More indexes will be plugged-in
  – It might put additional requirement to Mexima
Thank you!