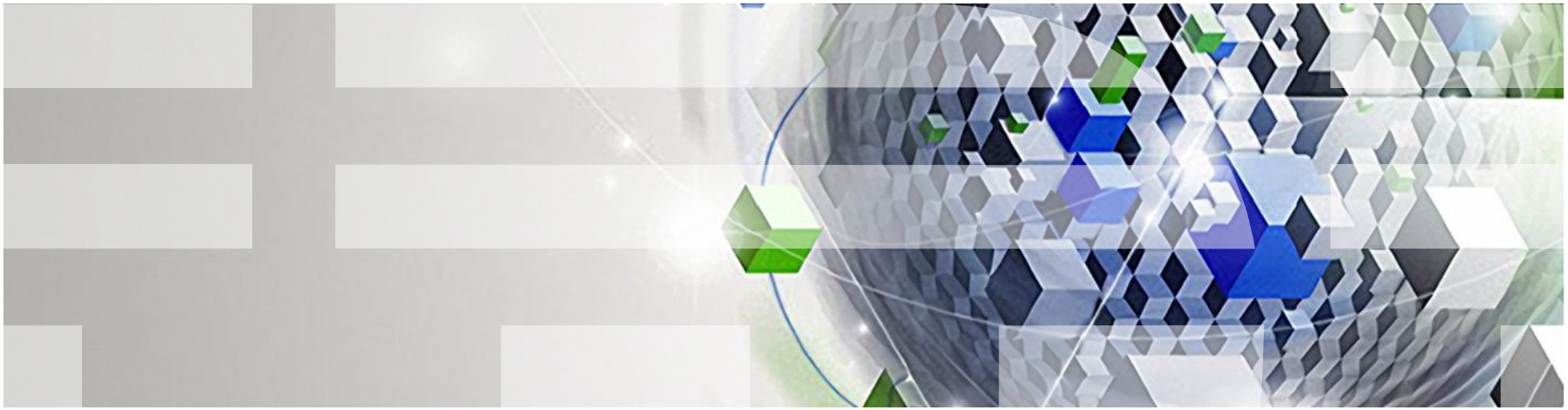
UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

IBM

Graph Databases

Guilherme Fetter Damasio

University of Ontario Institute of Technology and
IBM Centre for Advanced Studies

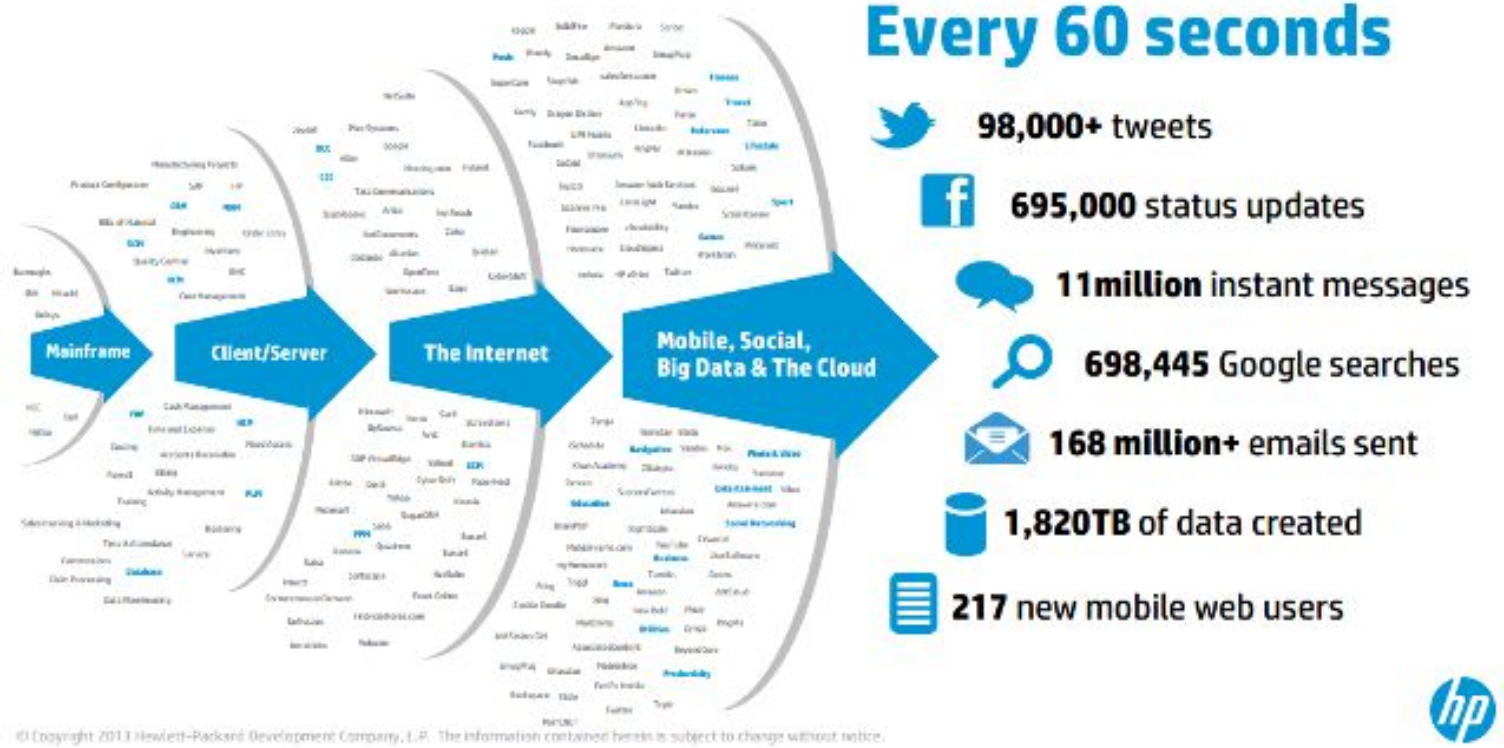


Outline

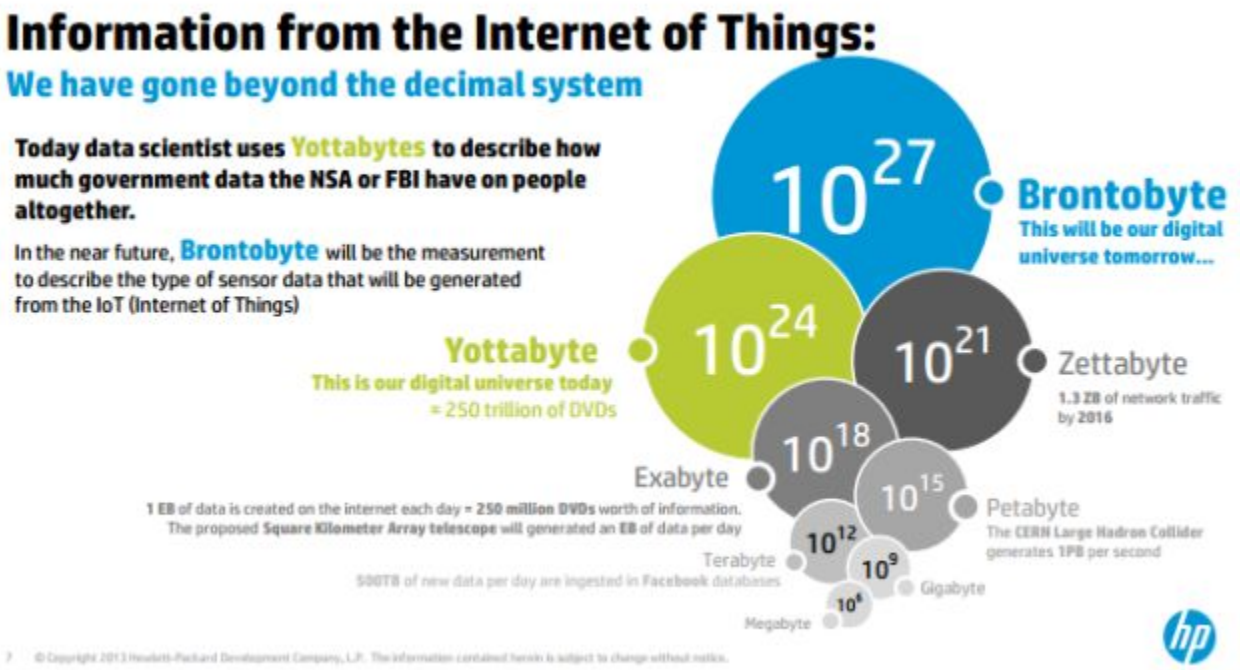
- Introduction
- Relational Database
- Graph Database
- Our Research

Introduction

- Data is everywhere
  - And it keeps increasing

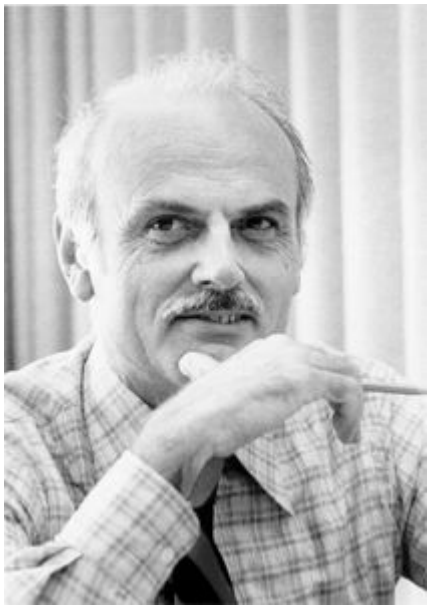


Introduction



Introduction

- How can we store all this data?
  - Database



Edgar Codd invents the Relational Data Model, and its first order theory. IBM team implements System R. 1970

Relational Database

- Relational Database
  - Highly structured data
  - Represents information in tables with rows (tuples) and columns (attributes)

- References to tuples in other table by referring to their primary-key (PK) attributes via foreign-key (FK) columns

PERSON		
ID	NAME	CAR_ID
1	John	1
...		
5	Brad	5

PK FK

CAR		
ID	TYPE	MAX_NUM_OF_PAS
1	Mercedes	2
...		
3	BMW	5

PK

Relational Database

- Structured Query Language (SQL)
  - Standard language for storing, manipulating and retrieving data in databases
  - Very-high-level language
  - Query optimization

- SQL example: find everyone who has a BMW car with max number of passengers = 5

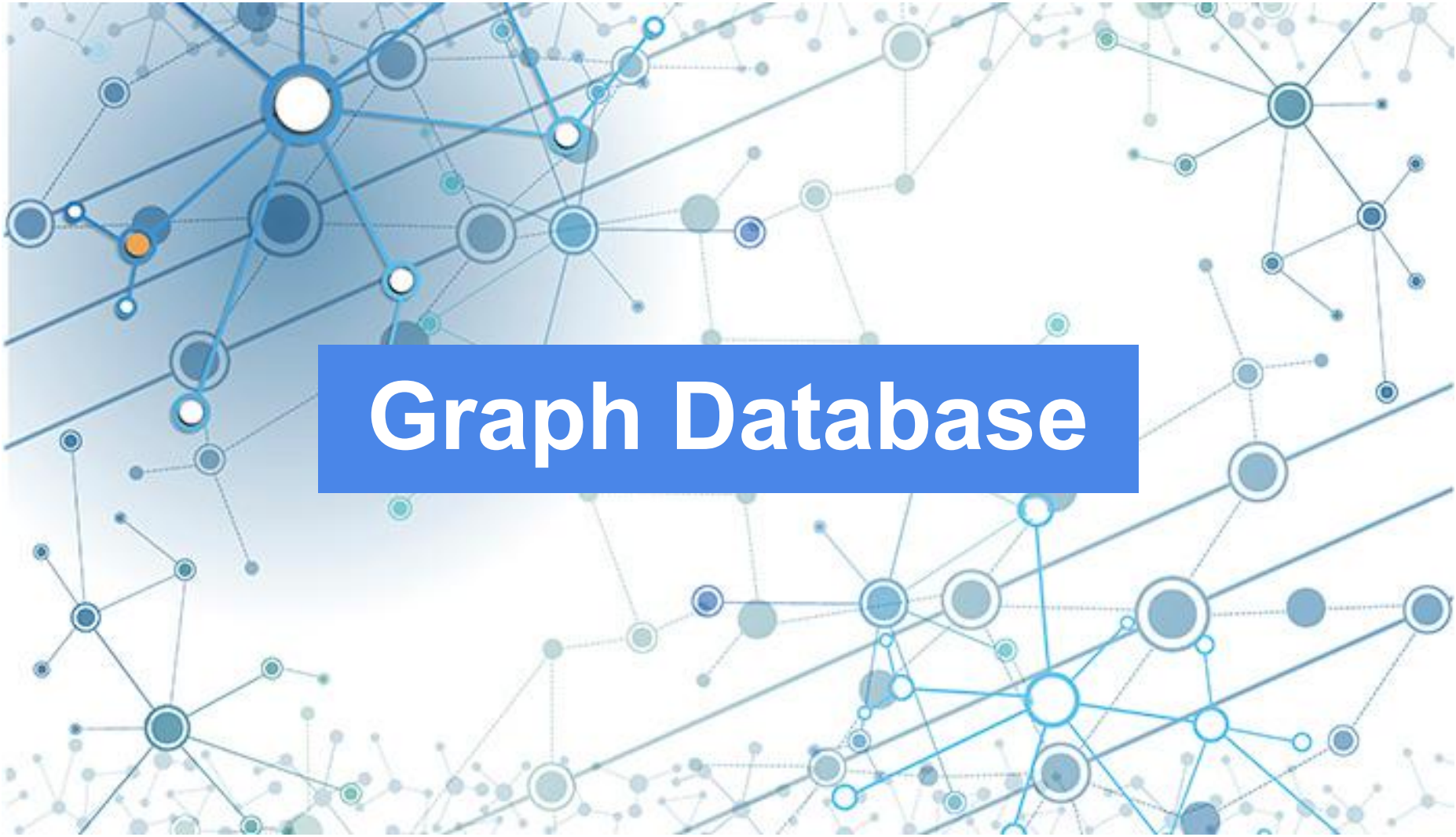
```
SELECT NAME FROM "PERSON"
    WHERE (
        PERSON."CAR_ID" IN (
            SELECT ID FROM CAR
            WHERE (
                CAR."TYPE" = 'BMW' AND
                CAR."MAX_NUM_OF_PAS" = 5)
            )
        )
    );
```

Relational Database

- Joins are computed at query time by matching primary- and foreign-keys of the many rows of the to-be-joined tables
- These operations are compute- and memory-intensive and have an exponential cost
- A lot of the data available is naturally represented as graphs
  - Data is related to resources
  - Example: Social network data

Relational Database

How can we improve the performance for this type of data?

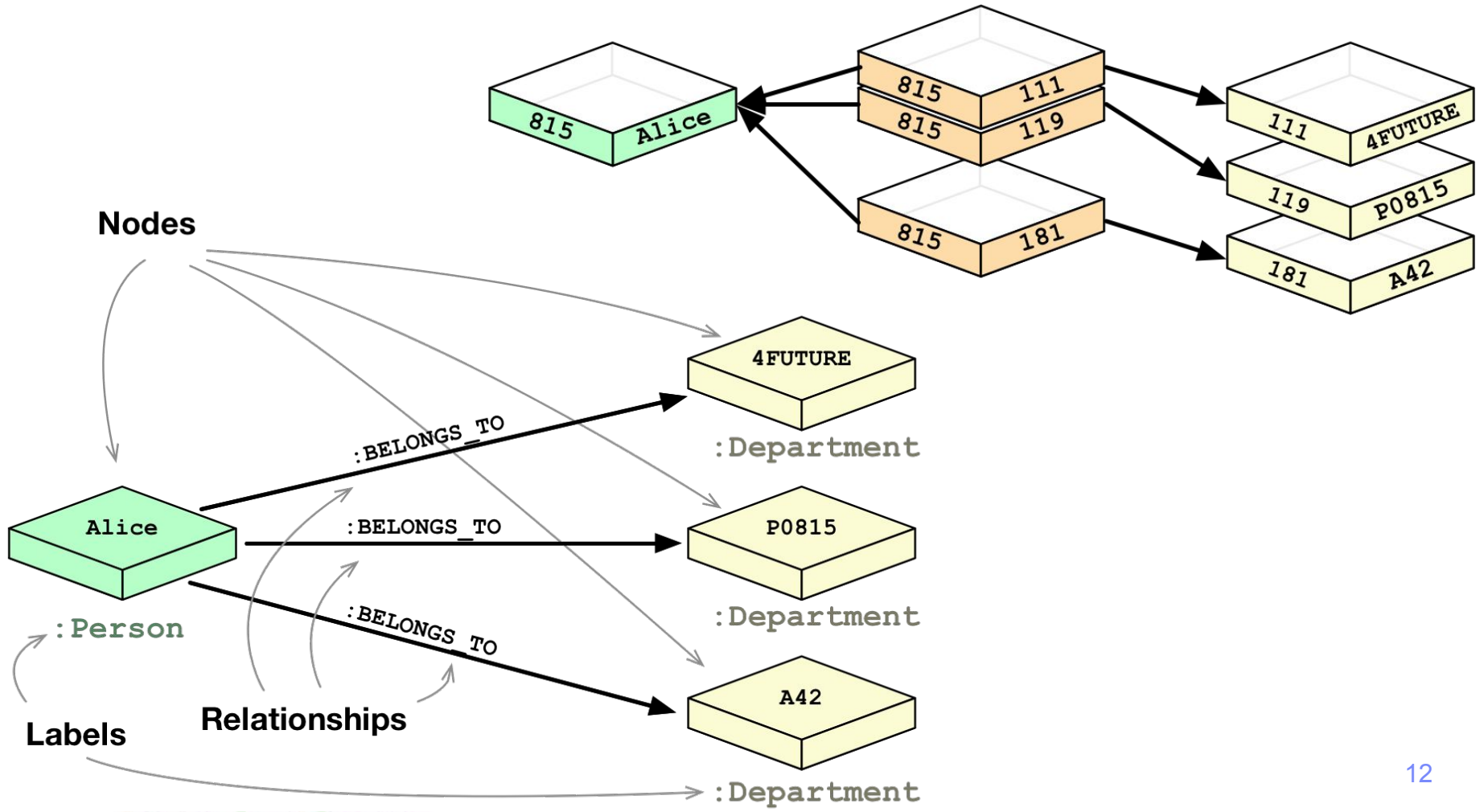

Graph Database

Graph Database

- Relationships are first-class citizens of the graph data model
- Graph databases enable us to build sophisticated models that map closely to our problem domain
- Each node (entity or attribute) in the graph database model directly and physically contains a list of relationship-records that represent its relationships to other nodes
- Whenever you run the equivalent of a *JOIN* operation, the database just uses this list and has direct access to the connected nodes, eliminating the need for a expensive search / match computation

Graph Database

- Support a very flexible and fine-grained data model
  - Manage rich domains in an easy and intuitive way



Graph Database

- Relational model to Graph model
  - Each entity table is represented by a label on nodes
  - Each row in a entity table is a node
  - Columns on those tables become node properties
  - Foreign keys become relationships

- Connections between nodes is directly link in such a way that relating data becomes a simple matter of following connections

- Helps companies to make sense of the masses of connect data that exist today

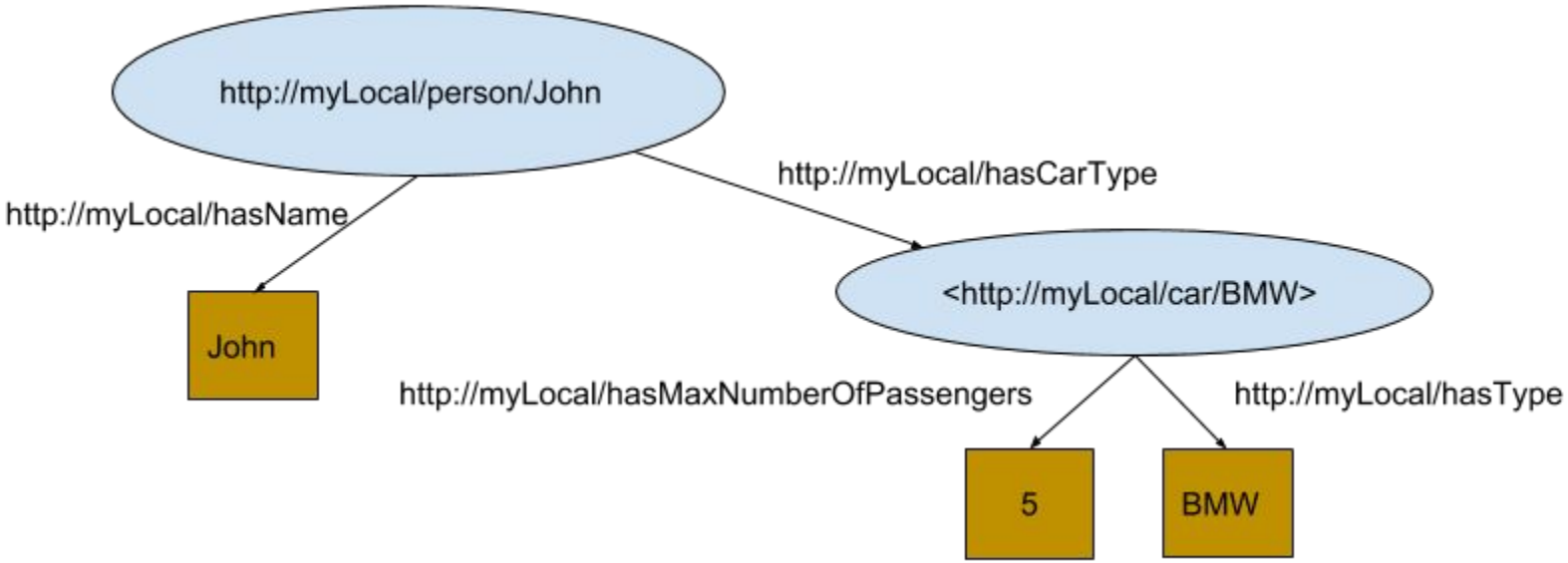
RDF

- Resource Description Framework (RDF)
  - W3C specification
  - Standard model for data interchange on the web
- The language description model allows the creation of statements about a resource either by defining its relationships with other resources or by defining its attributes
- Model is composed by triples
  - Subject
  - Predicate
  - Object
- Subjects, predicates and objects are normally composed by a URI and a node identification

RDF

- Example:

  A person called "John" that has a car "BMW", where this car has maximum number of passengers equals to "5".



RDF

- RDF can be represented in different formats:
  - N-triples, turtle, RDF/XML, etc.

- N-triples representation of the previous example:

<http://myLocal/person/John> <http://myLocal/hasName> “John” .
<http://myLocal/person/John> <http://myLocal/hasCarType> <http://myLocal/car/BMW> .
<http://myLocal/car/BMW> <http://myLocal/hasType> “BMW” .
<http://myLocal/car/BMW> <http://myLocal/hasMaxNumberOfPassengers>
“5”^^<http://www.w3.org/2001/XMLSchema#integer> .

- RDF can also be represented by quadruples
  - Graphs

RDF

- There is no direct way to represent properties in relationships, but it can be accomplished by *Blank Nodes*
  - Resource for which a URI or literal is not given
- Can make use of ontologies
  - Vocabularies in a standard format
  - Establishes the relationship between variables
- Ex: Integration of data coming from different publishers
  - The data can be imported into a common RDF model, eg, by using converters to the publishers' databases. However, one database may use the term “author”, whereas the other may use the term “creator”
- To make the integration complete, and extra definition should be added to the RDF data
  - Describing the fact that the relationship described as “author” is the same as “creator”

SPARQL

- SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language
  - Semantic query language for databases able to retrieve data stored in RDF format
  - Support by W3C

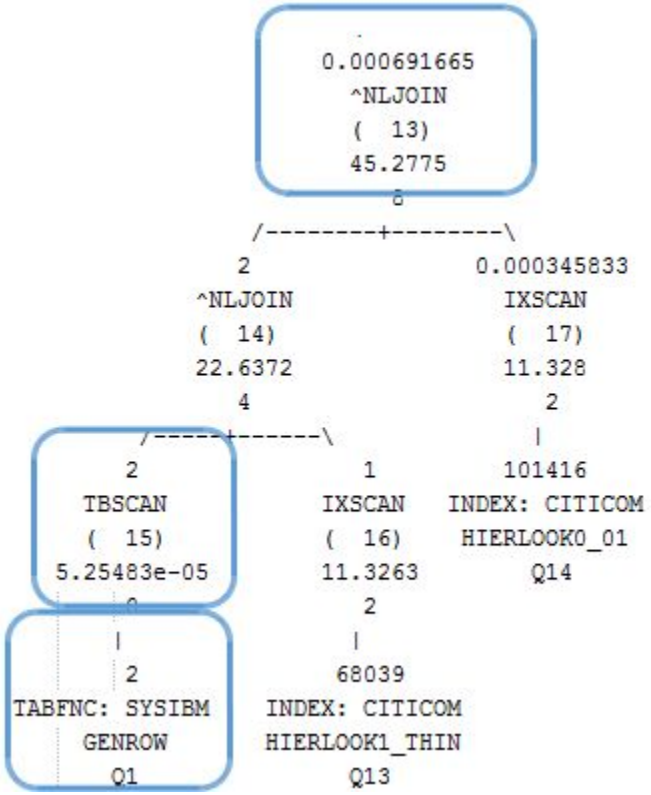
- SPARQL query that returns every person who has a BMW with maximum number of passengers as 5

```
PREFIX predicate: <http://myLocal/>
SELECT ?name
WHERE
{
        ?x      predicate:hasName                       ?name .
        ?x      predicate:hasCarType                    ?car .
        ?car    predicate:hasType                       "BMW" .
        ?car    predicate:hasMaxNumberOfPassengers 5
}
```

SPARQL

- Property paths
  - elt* → zero or more occurrences of *elt*
  - elt+ → one or more occurrences of *elt*
  - elt? → A path of zero or one *elt*
  - elt1 / elt2 → A sequence path of *elt1*, followed by *elt2*
  - ^elt → *Inverse path (object to subject)*

```
PREFIX predicate: <http://myLocal/>
SELECT ?a ?b ?c
WHERE
{
        ?a      predicate:hasType "NLJOIN" .
        ?a      predicate:hasLeftChild+ ?b .
        ?b      predicate:hasType "TBSCAN" .
        ?b      predicate:hasGenericChild ?c .
        ?c      predicate:hasType "BASEOB" .
}
```



Graph Database
Pros

- Flexibility
  - The data captured can be easily changed and extended for additional attributes and objects
- Data relationship exploration
  - No join index lookup (just follow connections)
  - Traverse millions of nodes per second
  - "Which supplier provided the products owned by this group of customers?"
- Indexing
  - Graph databases are naturally indexed by relationships (the strength of the underlying model), providing faster access compared to relational data for data

Graph Database
Cons

- Not efficient at
  - Processing high volumes of transactions
  - Handling queries that span the entire database
- For most common graph databases, you have to store all the data on one server
- Still fairly new compared to relational software, which has now existed for a full generation
  - It takes time to build a solid database market after all, regardless of data model
- Some graph databases only offer the graph model, but the underlying implementation is backed by a traditional

Relational Database x Graph Database

- Which one is better?
  - Depends

Graph Database

If your domain entities have relationships to other entities, and your queries rely on exploring those relationships

Relational Database

If you have large-volume analytics queries typical of data warehousing

Graph Database

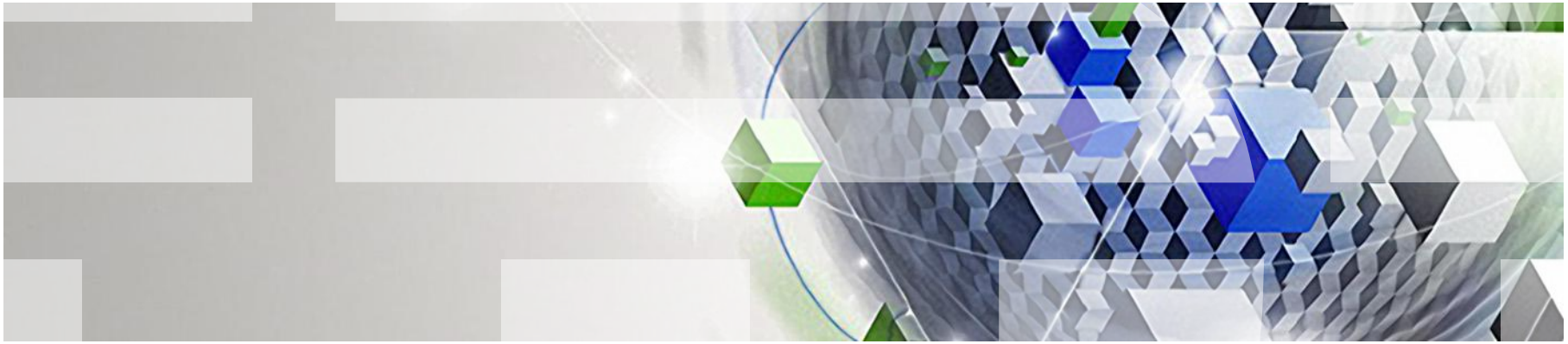
"It is important to note the consequence of using graph databases. The query latency in a graph is proportional to how much of the graph you choose to explore in a query, and is not proportional to the amount of data stored."

Jim Webber, author of *Graph Databases*

Query Performance Problem Determination with Knowledge Base in Semantic Web System OptImatch

Guilherme Fetter Damasio
Jaroslaw Szlichta
Piotr Mierzejewski
Calisto Zuzarte

University of Ontario Institute of Technology and
IBM Centre for Advanced Studies



Outline

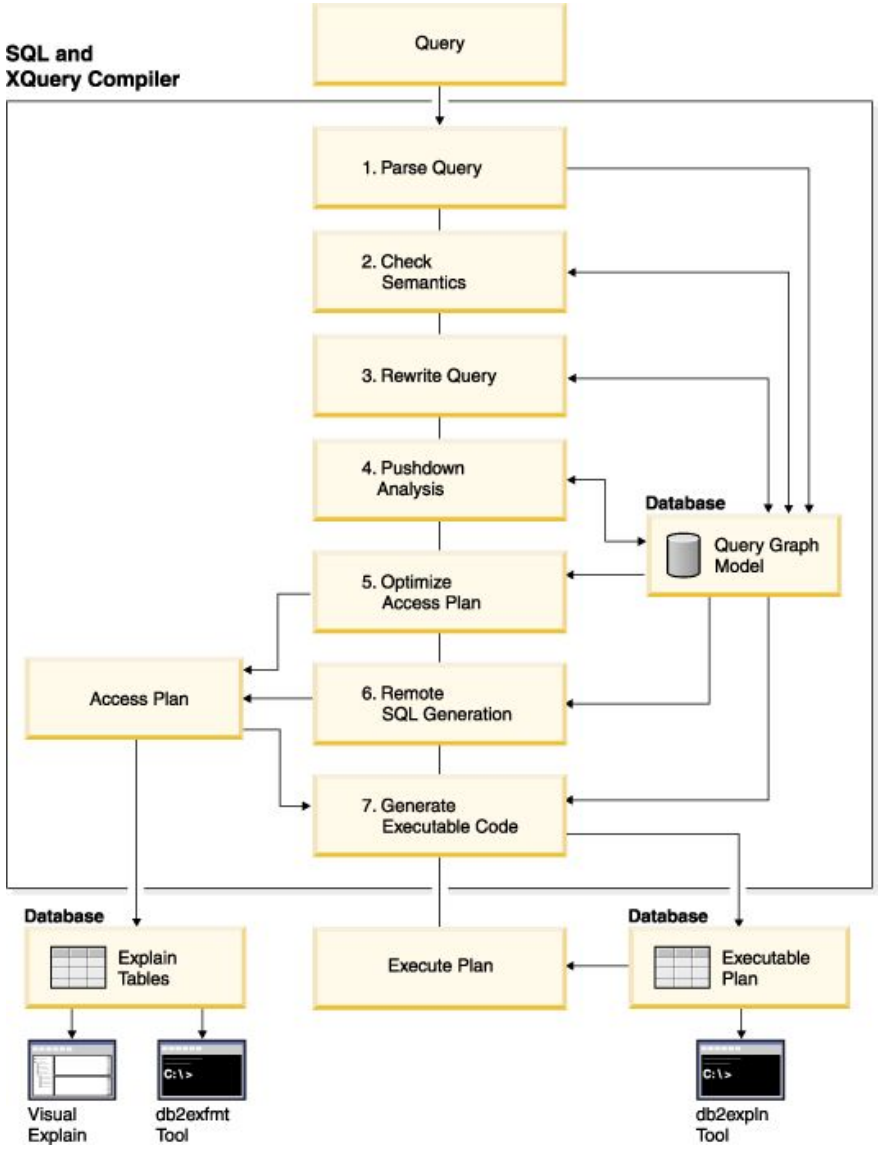
I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Outline

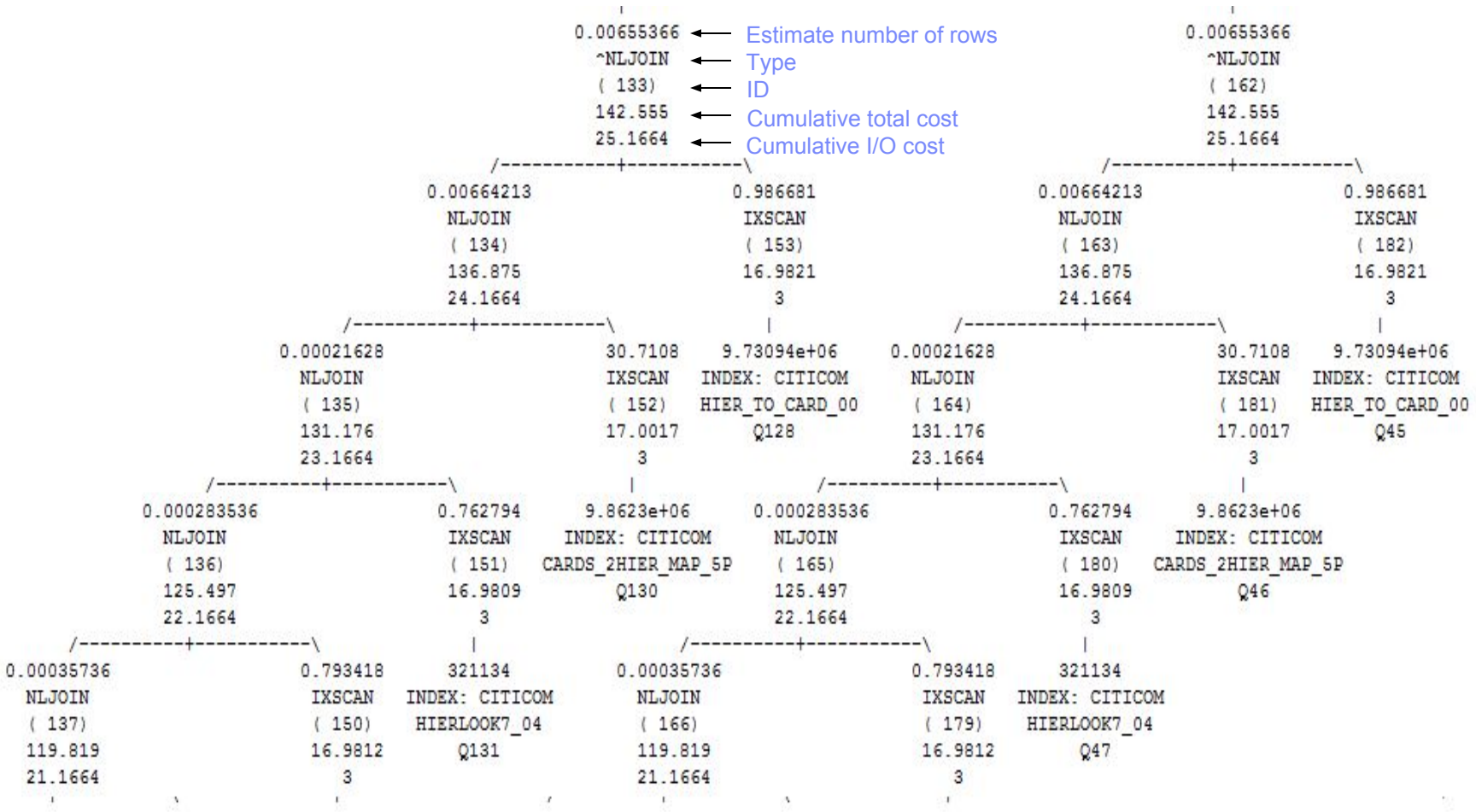
I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Background

- DB2
  - Relational Database Management System
  - Query tuning
  - Access Plan or Query Execution Plan (QEP)
- QEP
  - Contains a graphical format portion represented in a tree structure
  - Contains a text format portion



Background


0.00655366
^NLJOIN
(133)
142.555
25.1664
Estimate number of rows
Type
ID
Cumulative total cost
Cumulative I/O cost
0.00664213
NLJOIN
(134)
136.875
24.1664
0.986681
IXSCAN
(153)
16.9821
3
0.00021628
NLJOIN
(135)
131.176
23.1664
30.7108
IXSCAN
(152)
17.0017
3
9.73094e+06
INDEX: CITICOM
HIER_TO_CARD_00
Q128
0.000283536
NLJOIN
(136)
125.497
22.1664
0.762794
IXSCAN
(151)
16.9809
3
9.8623e+06
INDEX: CITICOM
CARDS_2HIER_MAP_5P
Q130
0.00035736
NLJOIN
(137)
119.819
21.1664
0.793418
IXSCAN
(150)
16.9812
3
321134
INDEX: CITICOM
HIERLOOK7_04
Q131
0.00655366
^NLJOIN
(162)
142.555
25.1664
0.00664213
NLJOIN
(163)
136.875
24.1664
0.986681
IXSCAN
(182)
16.9821
3
0.00021628
NLJOIN
(164)
131.176
23.1664
30.7108
IXSCAN
(181)
17.0017
3
9.73094e+06
INDEX: CITICOM
HIER_TO_CARD_00
Q45
0.000283536
NLJOIN
(165)
125.497
22.1664
0.762794
IXSCAN
(180)
16.9809
3
9.8623e+06
INDEX: CITICOM
CARDS_2HIER_MAP_5P
Q46
0.00035736
NLJOIN
(166)
119.819
21.1664
0.793418
IXSCAN
(179)
16.9812
3
321134
INDEX: CITICOM
HIERLOOK7_04
Q47

Background

-

```
8.33333
TBSCAN
(   2)
5477.33
5116.67
```

```
2) TBSCAN: (Table Scan)
    Cumulative Total Cost:        5477.33
    Cumulative CPU Cost:          1.52058e+009
    Cumulative I/O Cost:          5116.67
    Cumulative Re-Total Cost:     2690.61
    Cumulative Re-CPU Cost:       7.83255e+008
    Cumulative Re-I/O Cost:       0
    Cumulative First Row Cost:    5477.33
    Estimated Bufferpool Buffers:    0

    Arguments:
    ---------
    MAXPAGES: (Maximum pages for prefetch)
        ALL
    PREFETCH: (Type of Prefetch)
        NONE
    SCANDIR : (Scan Direction)
        FORWARD
    SPEED   : (Assumed speed of scan, in sharing structures)
        SLOW
    THROTTLE: (Scan may be throttled, for scan sharing)
        FALSE
    VISIBLE : (May be included in scan sharing structures)
        FALSE
    WRAPPING: (Scan may start anywhere and wrap)
        FALSE
```

Outline

I. Background

II. Motivation

III. Architecture

IV. Transforming Diagnostic Data

V. Searching for Problem Patterns

VI. Providing Recommendations with Knowledge Base

VII. Experimental Study

VIII. Current Work

Motivation

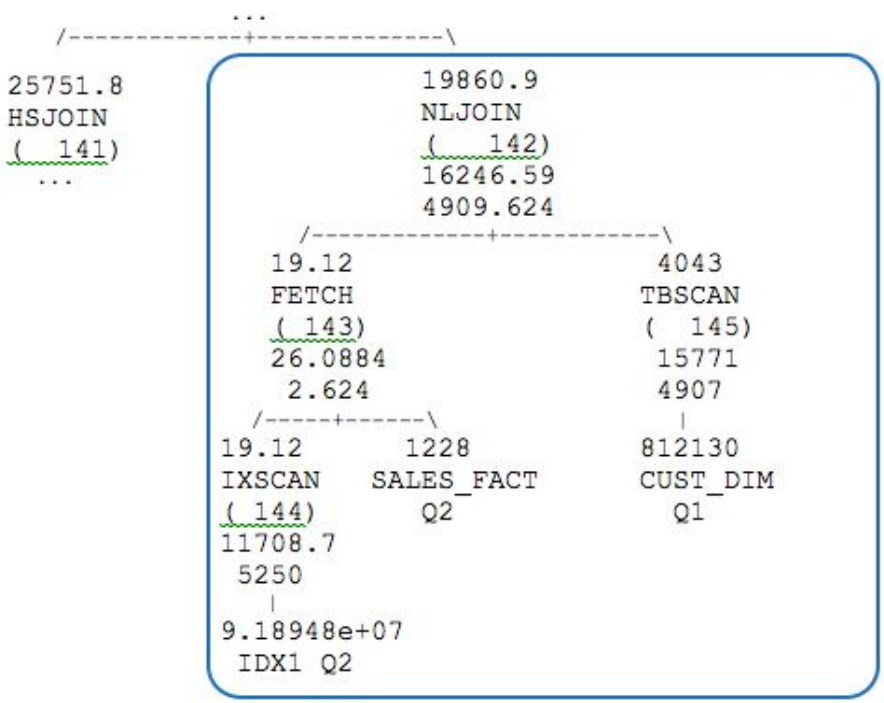
- World's most valuable data remains in relational databases
- Complex analytic queries are a part of business operations
- Database systems are becoming more sophisticated
- Currently, tools for query performance problem determination have limitations, so performance analysis is often best done by manually analysis

Motivation

- Existing tools provide running recommendations for specific known problems
  - IBM® Optim Query Tuner®, IBM Optim Workload Tuner®, etc.
- Some characteristics of query execution plan (QEPs) are not easily found by the use of search tools like “grep”
- Query performance problem determination is valuable inside IBM support of business clients and database optimizer development organization
- Optimizations and performance tuning strategies becomes necessary to maintain the usability of the database

Motivation

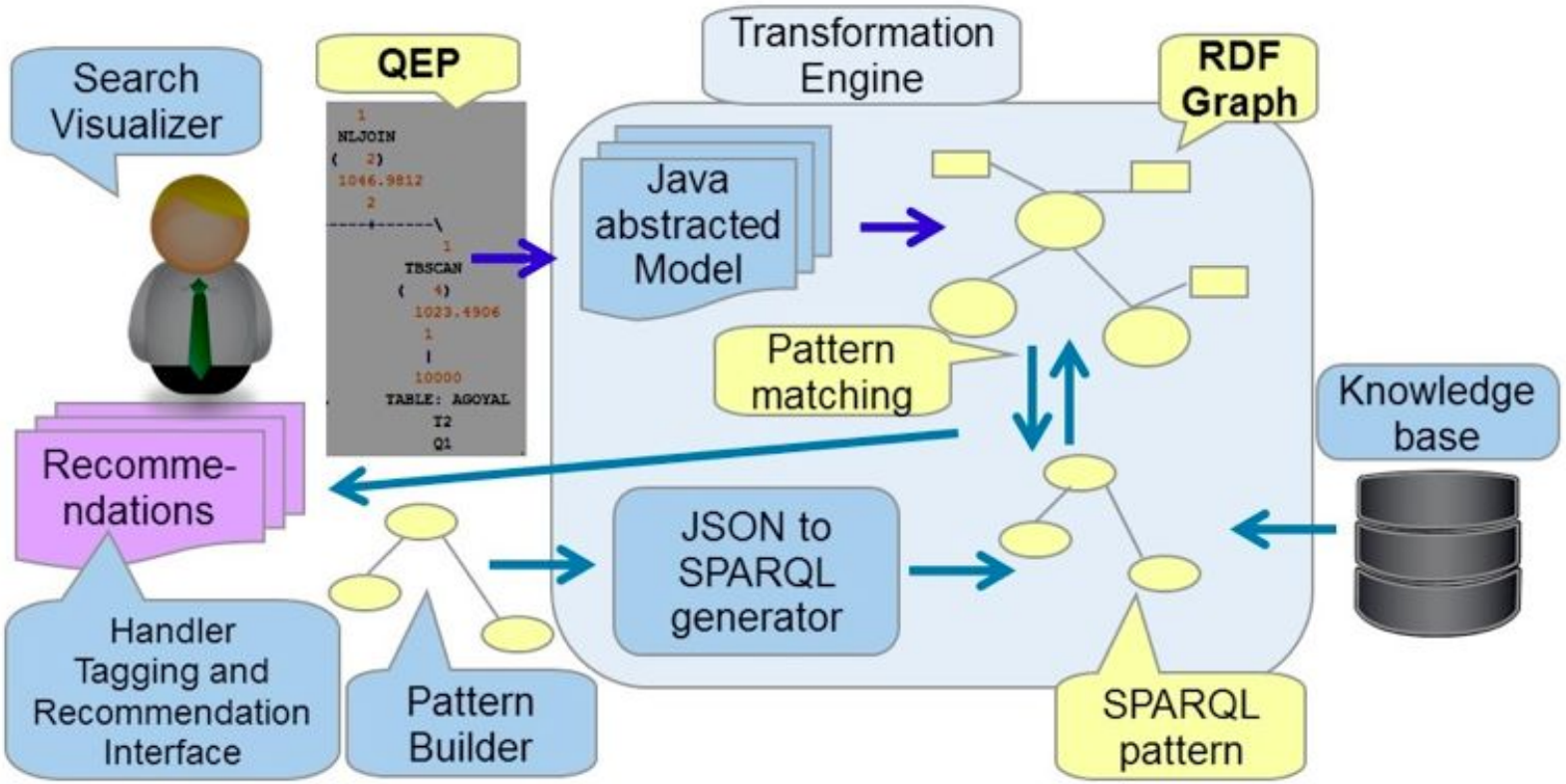
- Incorporating OptImatch many optimization problems could be automatically identified and resolved
- Text graph version of a snippet of a QEP from IBM DB2
- Problem: NLJOIN has inner stream of type table scan (TBSCAN)
  - Costly
  - NLJOIN scans entire inner TABLE each rows from the outer
- Solution: create an index of the target table of the TBSCAN



Outline

I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Architecture



Outline

I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Transforming Diagnostic Data

- Parser
  - Define the property structure and property name for the RDF

- RDF
  - Represents the QEP
  - RDF is supported by DB2
  - RDF does not enforce specific schema, but it can be enforced by specifying predicates and establishing the relationship between LOLEPOPs

Subject Predicate Object

```
<http://explainPlan/PlanPop/2> <http://explainPlan/PlanPred/hasPopType> "NLJOIN" .
<http://explainPlan/PlanPop/2> <http://explainPlan/PlanPred/hasOuterInputStream> <http://explainPlan/PlanPop/3> .
<http://explainPlan/PlanPop/2> <http://explainPlan/PlanPred/hasInnerInputStream> <http://explainPlan/PlanPop/5> .
<http://explainPlan/PlanPop/3> <http://explainPlan/PlanPred/hasPopType> "FETCH" .
<http://explainPlan/PlanPop/3> <http://explainPlan/PlanPred/hasEstimatedCardinality> "19.12" .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasPopType> "TBSCAN" .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasEstimatedCardinality> "4043.0".
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasTotalCost> "15771.0" .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasIOCost> "49007.0" .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasJoinInputLeg> "INNER" .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasOutputStream> <http://explainPlan/PlanPop/2> .
<http://explainPlan/PlanPop/5> <http://explainPlan/PlanPred/hasInputStreamPop> <http://explainPlan/PlanBaseObject/CUST_DIM> .
<http://explainPlan/PlanBaseObject/CUST_DIM> <http://explainPlan/PlanPred/hasEstimateCardinality> "812130.0"
```

Transforming Diagnostic Data

- SPARQL
  - RDF language
  - Property paths
  - Supported by DB2
- Handlers
  - Result (Retrieval)
  - Internal (Filtering)
  - Relationship
  - Blank Node

```
PREFIX predURI: <http://explainPlan/PlanPred/>
  SELECT (?pop1 AS ?TOP) (?pop2 AS ?ANY2)  (?pop4 AS ?BASE4)
  WHERE { ?pop1 predURI:hasPopType "NLJOIN" .
     ?pop1 predURI:hasOuterInputStream  ?BNodeOfpop2_to_pop1 .
     ?BNodeOfpop2_to_pop1 predURI:hasOuterInputStream   ?pop2 .
     ?pop2 predURI:hasOutputStream ?BNodeOfpop2_to_pop1.
     ?BNodeOfpop2_to_pop1 predURI:hasOutputStream ?pop1 .
     ?pop1 predURI:hasInnerInputStream ?BNodeOfpop3_to_pop1.
     ?BNodeOfpop3_to_pop1 predURI:hasInnerInputStream  ?pop3 .
     ?pop3 predURI:hasOutputStream ?BNodeOfpop3_to_pop1.
     ?BNodeOfpop3_to_pop1 predURI:hasOutputStream ?pop1 .
     ?pop3 predURI:hasPopType "TBSCAN" .
     ?pop3 predURI:hasEstimateCardinality ?internalHandler1 .
           FILTER ( ?internalHandler1 > 100) .
     ?pop3 predURI:hasInputStream ?BNodeOfpop4_to_pop3 .
     ?BNodeOfpop4_to_pop3 predURI:hasInputStream ?pop4 .
     ?pop4 predURI:hasOutputStream ?BNodeOfpop4_to_pop3.
     ?BNodeOfpop4_to_pop3 predURI:hasOutputStream?pop3 .
     ?pop4 predURI:isABaseObj  ?internalHandler2 . } ORDER BY ?pop1
```

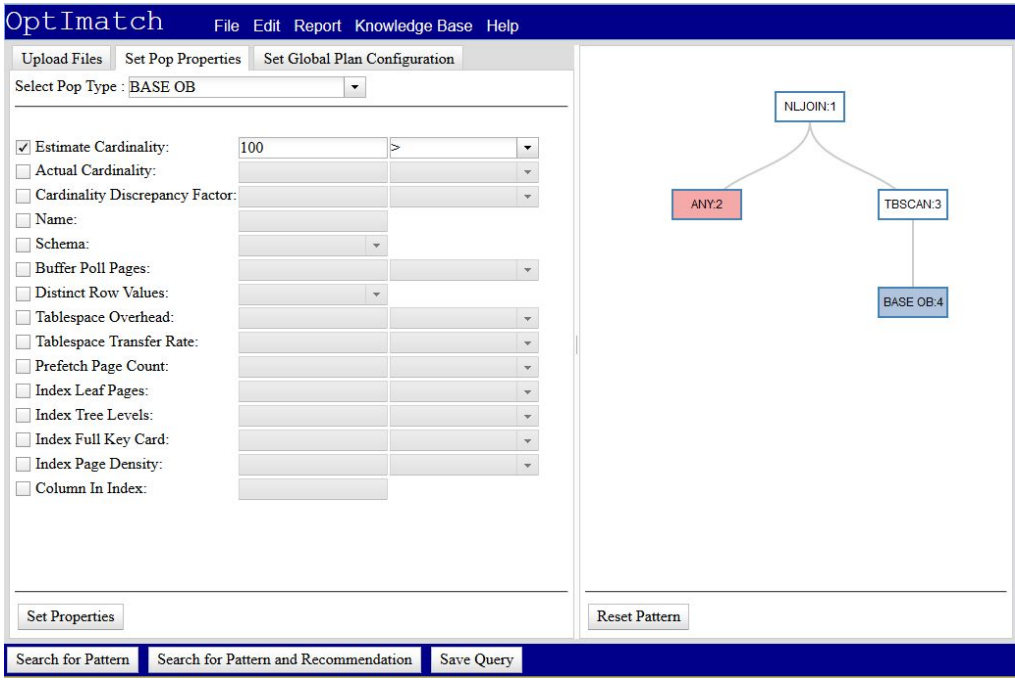
```
{"id":"hasEstimateCardinality",
     "value":"1","sign":"<"}        ──→  ?pop1 predURI:hasEstimateCardinality ?internalHandler1
                                         FILTER ( ?internalHandler1 < 1)

{"id":"hasEarlyOutFlag","value":"LEFT"} ──→  ?pop1 predURI:hasEarlyOutFlag "LEFT"

{"id":"hasOuterInputStream",
     "value":2,"sign":"Immediate Child"} ──→  ?pop1 predURI:hasOuterInputStream ?BNodeOfpop2_to_pop1
                                              ?BNodeOfpop2_to_pop1 predURI:hasOuterInputStream ?pop2
```

Outline

I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Searching for Problem Patterns

- Web-based graphical interface



- Upload Files
- Properties and relationship between nodes
- Pattern #1 example (High read cost)
  - (i) is of type “NLJOIN”
  - (ii) outer input stream of type “ANY” and cardinality > 1
  - (iii) has inner input stream of type “TBSCAN”
  - (iv) inner input stream has cardinality > 100
  - (v) “TBSCAN” has generic input stream of type “BASE OB”
- Solution
  - Create an index in the table read by the TBSCAN

Searching for Problem Patterns

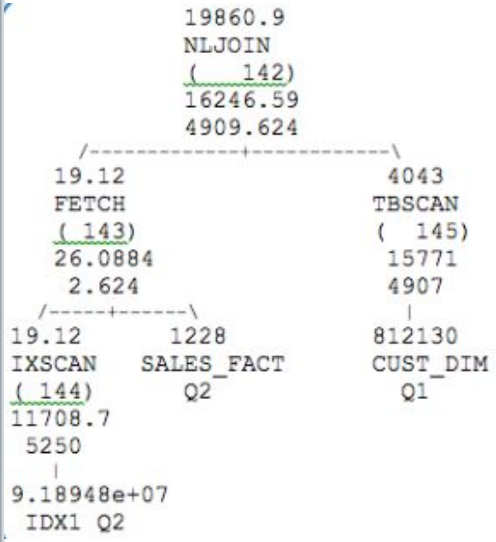
Query Result

```
********************************************************************************
Time to Analysis: 0 hr, 0 min, 0 sec, 217 milli
Number of Files Analized: 20
Files Matching Pattern: 2
Files Matching Recommendation: 0
********************************************************************************

====================
Result for the file: 1000sql_3_ADVITO_HOTEL_REPORT.sql.explain
```

TOP	ANY2	TBSCAN3	BASE4
142	143	37	COUNTRY_CCRS

```
====================
Result for the file: 1000sql_7__CARD_HOLDER_DEMOGRAPHICS_REPORT.sql.explain
```

TOP	ANY2	TBSCAN3	BASE4
4	5	47	COUNTRY_CCRS



Outline

I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Providing Recommendations with Knowledge Base

- Populated with predetermined problem patterns and associated query plan recommendations
- Problem pattern → SPARQL query with result handlers
- Use of alias to specify a result handler
- Use of handlers tagging

```
PREFIX predURI: <http://explainPlan/PlanPred/>
  SELECT (?pop1 AS ?TOP) (?pop2 AS ?ANY2)  (?pop4 AS ?BASE4)
```

- Static and dynamic components
- Use of "@" sign to represent a result handler

Providing Recommendations with Knowledge Base

- Use of 1 handler: @TOP
- Use of multiple handlers: [@TOP, @BASE4]
- Limit number of occurrences: [@TOP, @BASE4]:1
  - List only 1 result
- Use of helper functions
- List columns: @TOP.listColumns(“PREDICATE”)
  - List columns from an alias handler in the predicate
- Example: *“Create index on table @BASE4 on columns @TOP.listColumns("INPUT”)”*

Providing Recommendations with Knowledge Base

Query Result

```
********************************************************************************
Time to Analysis: 0 hr, 0 min, 2 sec, 284 milli
Number of Files Analized: 20
Files Matching Pattern: 2
Files Matching Recommendation: 2
********************************************************************************

====================
Result for the file: 1000sql_3_ADVITO_HOTEL_REPORT.sql.explain
```

TOP	ANY2	TBSCAN3	BASE4
9	10	37	COUNTRY_CCRS

```
RECOMMENDATIONS:
  TAG: QCI001
  RECOMMENDATION:
    Crie index na tabela {COUNTRY_CCRS} nas colunas {Q11.PARENT_HIERARCHY_NUM}, {Q15.A3},
{Q16.CURRENCY_TEXT}, {Q2.ACCOUNT_NUMBER}, {Q2.BILLING_AMOUNT}, {Q2.BILLING_CURRENCY},
{Q2.DB_CR_FLAG}, {Q2.MCC(A)}, {Q2.MERCHANT_CITY}, {Q2.MERCHANT_STATE}, {Q2.MERCHANT_ZIP},
{Q2.MERCH_ACCEPTOR_ID}, {Q2.MERCH_COUNTRY}, {Q2.MERCH_DESCRIPTION}, {Q2.POST_DATE},
{Q2.TRANS_DATE}, {Q2.TXNSKEY}, {Q3.SHORT_NAME}
-------------------------


====================
Result for the file: 1000sql_7__CARD_HOLDER_DEMOGRAPHICS_REPORT.sql.explain
```

TOP	ANY2	TBSCAN3	BASE4
4	5	47	COUNTRY_CCRS

```
RECOMMENDATIONS:
  TAG: QCI001
  RECOMMENDATION:
    Crie index na tabela {COUNTRY_CCRS} nas colunas {Q21.ACCOUNT_NUMBER}, {Q21.ACCOUNT_TYPE},
{Q21.ADDRESS_LINE_1}, {Q21.ADDRESS_LINE_2}, {Q21.ADDRESS_LINE_3}, {Q21.AMOUNT_OF_LAST_PAYMENT},
{Q21.AMOUNT_OVER_LIMIT}, {Q21.AMOUNT_PAST_DUE_CYCLE1}, {Q21.BILL_TYPE}, {Q21.CARDH_FIRST_NAME},
{Q21.CARDH_LAST_NAME}, {Q21.CARD_ACTIVATION_DATE}, {Q21.CITY}, {Q21.COST_CENTER},
{Q21.COST_CENTER_DESCRIPTION}, {Q21.COUNTRY_NUM}, {Q21.CREDIT_LIMIT}, {Q21.CREDIT_RATING},
{Q21.DEPARTMENT}, {Q21.DISCRETIONARY_CODE}, {Q21.EMAIL}, {Q21.EMPLOYEE_ID}, {Q21.FIRST_NAME},
{Q21.HIERARCHY_NUM}, {Q21.ID_VERIFICATION_CODE}, {Q21.LAST_NAME}, {Q21.LAST_PAY_DATE},
{Q21.NAME}, {Q21.OPEN_DATE}, {Q21.OTHER_ACCT_NBR}, {Q21.PARENT_HIERARCHY_NUM},
```

Providing Recommendations with Knowledge Base

- OptImatch can provide advanced guidance with variety of recommendations

- Pattern #2 (estimation of the execution cost by optimizer)
  - (i) LOLEPOP of type index Scan (IXSCAN) or table scan (TBSCAN)
  - (ii) has cardinality < 0.001
  - (iii) has a generic input stream of type Base Object (BASE OB)
  - (iv) the generic input stream has cardinality > 100000

- Recommendation:
  - Create column group statistics (CGS) on equality local predicate columns and CGS on equality join predicate columns of the Base Object.

Providing Recommendations with Knowledge Base

- Pattern #3 (poor join order)
  - (i) LOLEPOP of type JOIN
  - (ii) has a descendant outer input stream of type JOIN
  - (ii) has a descendant inner input stream of type JOIN
  - (iii) the descendant outer input stream join is a Left Outer Join
  - (iv) descendant inner input stream join is a Left Outer Join

- Recommendation:
  - Rewrite the query from the following structure (T1 LOJ T2) … JOIN … (T3 LOJ T4) to ((T1 LOJ T2).... JOIN ....T3) LOJ T4 as the rewritten query is more efficient

- With all provided patterns and recommendations, a user, with no particular knowledge or training, can run a general test of all predetermined problem patterns against a given query workload

Outline

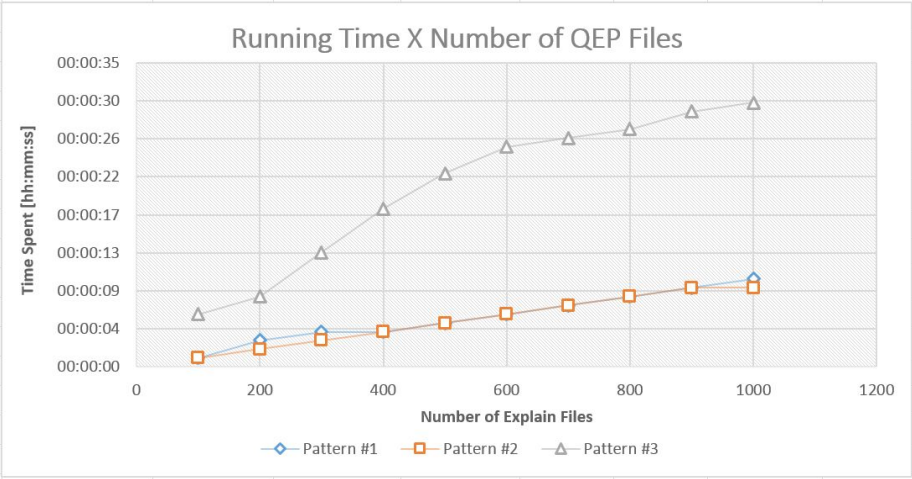
I. Background
II. Motivation
III. Architecture
IV. Transforming Diagnostic Data
V. Searching for Problem Patterns
VI. Providing Recommendations with Knowledge Base
VII. Experimental Study
VIII. Current Work

Experimental Study

- Focuses on 3 objectives:
  - Effectiveness using real IBM customer query workload.
  - Scalability and performance:
    - Sizes of the query workload,
    - Number of LOLEPOPs
    - Number of recommendations in the knowledge base.
  - Comparative study with manual search for patterns by experts
    - Time and precision

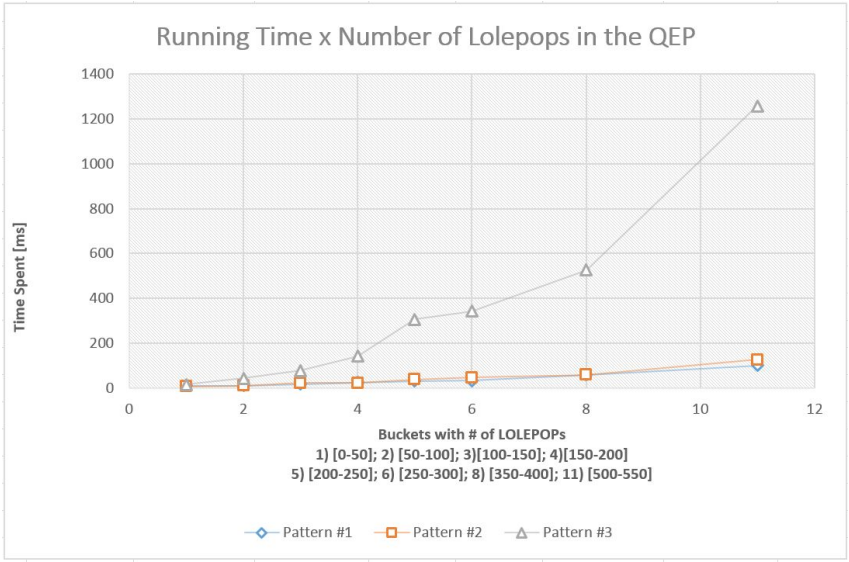
- Patterns used through experimental study:
  - Pattern #1
  - Pattern #2
  - Pattern #3

Performance and Scalability Study
Size of Query Workload



- Search increases linearly with the number of QEP files
- For 1000 QEPs, time to perform the search is less than 30 seconds
- Pattern #3 takes longer because it is more complex, containing descendant nodes (using recursion to analyze)

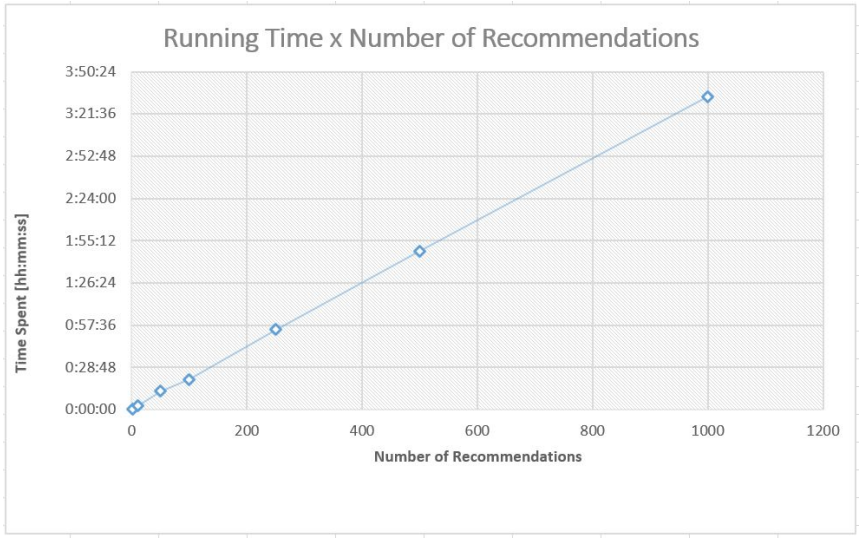
Performance and Scalability Study
Number of LOLEPOPs



- For patterns #1 and #2 the search increases linearly with the number LOLEPOPs

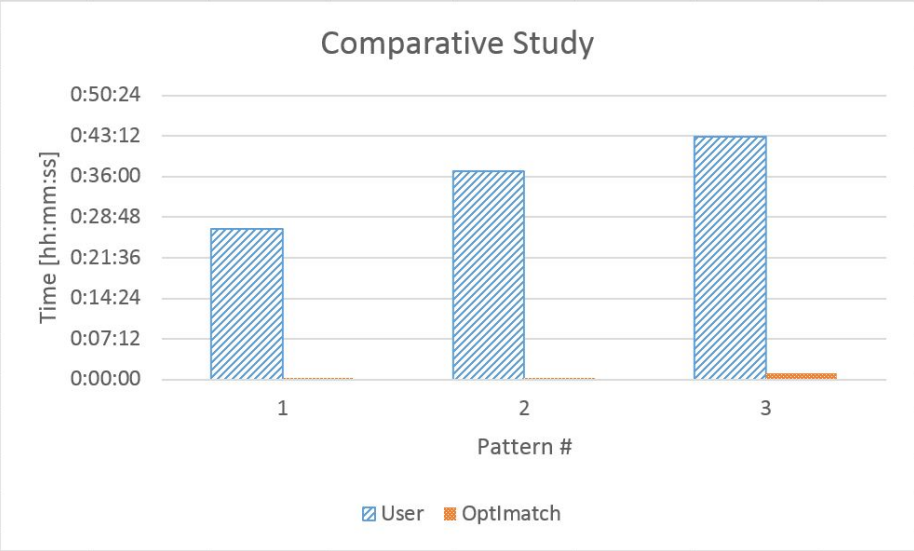
- Even large and complex queries (with around 500 LOLEPOPs) can be processed efficiently by our tool (less than 1400 milliseconds)

Performance and Scalability Study
Number of Recommendations in Knowledge Base



- OptImatch has linear dependency over the number of recommendations in the knowledge base
- Process a 1000 query workload against 1000 problem patterns and recommendations in around 3hr30min

Comparative User Study



Pattern #	#1	#2	#3
Precision	88%	71%	81%

- OptImatch drastically reduces the time to search for a pattern
- OptImatch does not only perform significantly faster than a manual search but it also guarantees correctness.

Outline

Current Work

- Extension of OptImatch
  - Automatically discover, based on previously knowledge, of the best QEP to be applied to a given query
  - Automatically recommendations such as changing database configuration in order to improve performance
- Knowledge Base
  - Information about QEP and queries
- Matching similarities

Current Work

- Normalization
  - Reduction of data redundancy

- 3 main pieces:
  - BaseData (tables and columns)
  - LinkData (sql) → Ex: tables being joined, predicates used
  - Knowledge Base (our current RDF)

BaseData		
Table	Column	Index
Card Column[ ] Npages Mpages Fpages	Type IsNull Length	Column[ ]

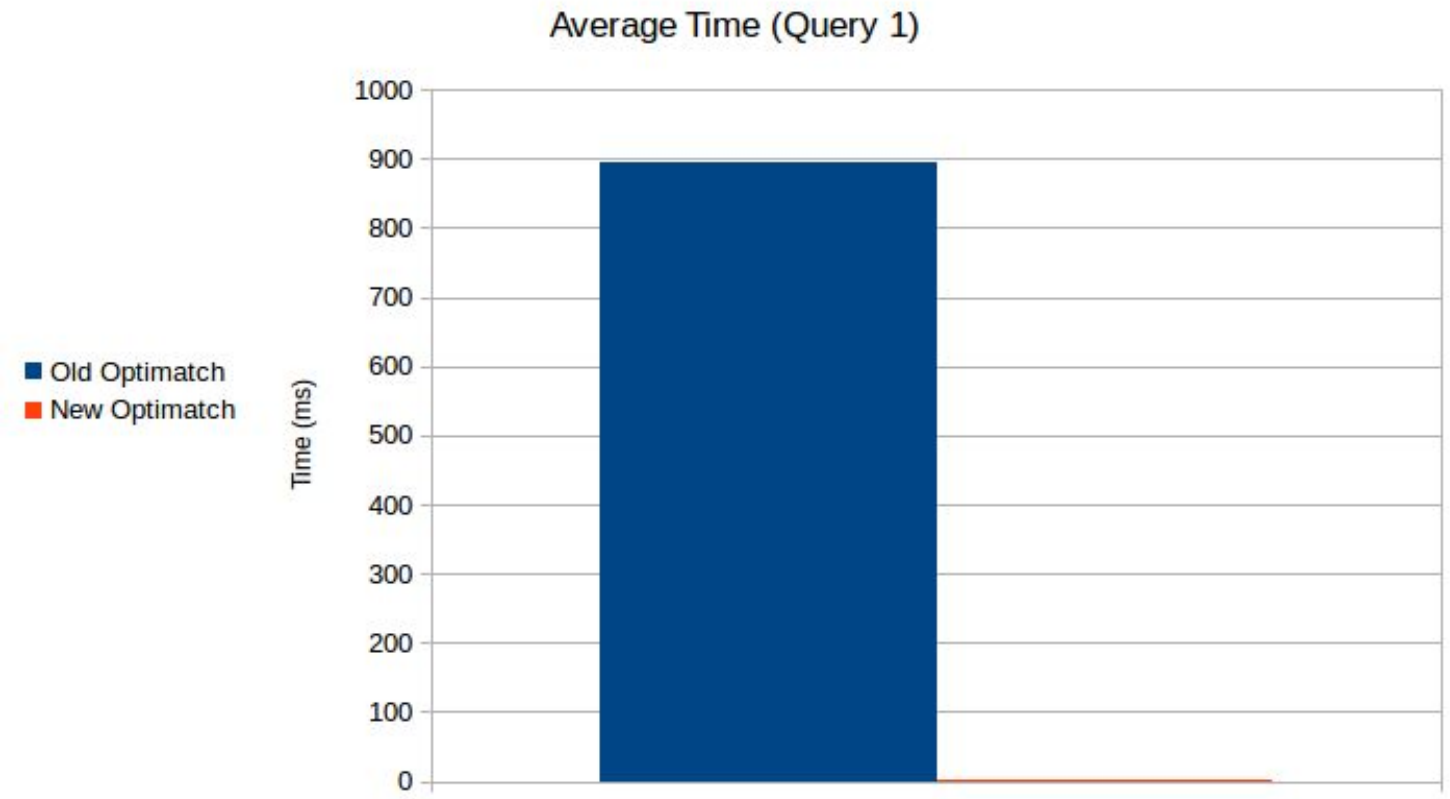
LinkData		
SQL	Table	Predicate
NumOfTables Table[ ] Predicate[ ] BestPlanID	NumOfColums Column[ ]	Column RelOp Col / LocPred

Knowledge Base Record			
Query	SPARQL	GUIDELINE	RDF_ID
			ID

Knowledge Base	
ID	RDF
ID	BestPlanRDF

Current Work

- Old OptImatch: 894 ms
- New OptImatch: 1.52 ms



Publications

- EDBT
  - G. Damasio, P. Mierzejewski, J. Szlichta and C. Zuzarte: Query Performance Problem Determination with Knowledge Base in Semantic Web System OptImatch. EDBT 2016, 515-526

- ICDE
  - G. Damasio, P. Mierzejewski, J. Szlichta and C. Zuzarte: OptImatch: Semantic Web System for Query Problem Determination. IEEE ICDE 2016, 1334-1337