FDB: A Query Engine for Factorised Relational Databases

Nurzhan Bakibayev, Dan Olteanu, and Jakub Zavodny — Oxford CS

Christan Grant
cgrant@cise.ufl.edu

University of Florida

November 1, 2013

Contents

1. Intro
2. F-Representation
3. F-Trees
4. Query Evaluation
   - Normalisation Operator
   - Swap Operator
   - Cartesian Product Operator
   - Selection Operator
     - Merge Selection Operator
     - Absorb Selection Operator
     - Selection with Constant Operator
   - Projection Operator
5. Query Optimization
   - Cost of an F-Plan
   - Exhaustive Search
   - Greedy Heuristic
6. Experiments
   - Experiment #1
   - Experiment #2
   - Experiment #2
   - Experiment #3
   - Experiment #4
7. Conclusion
8. Thanks

Intro

- A succinct representation for large relations.
- It is used in Google's F1 data management system for ML.
- Used for large incomplete/uncertain relations.
- FDB can speed up expected queries.

Intro

- A succinct representation for large relations.
- It is used in Google's F1 data management system for ML.
- Used for large incomplete/uncertain relations.
- FDB can speed up expected queries.
- This paper discusses basics FDB system.

TL;DR

- Given a query: $Q_1 = \text{Order} \bowtie_{item} \text{Store} \bowtie_{location} \text{Disp}$

TL;DR

- Given a query: $Q_1 = \text{Order} \bowtie_{item} \text{Store} \bowtie_{location} \text{Disp}$
- And a result:

$Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$

oid	item	location	dispatcher
01	Milk	Istanbul	Adnan
01	Milk	Istanbul	Yasemin
01	Milk	Izmir	Adnan
01	Milk	Antalya	Volkan
		...	

TL;DR

- Given a query: $Q_1 = \text{Order} \bowtie_{item} \text{Store} \bowtie_{location} \text{Disp}$
- And a result:

$Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$

oid	item	location	dispatcher
01	Milk	Istanbul	Adnan
01	Milk	Istanbul	Yasemin
01	Milk	Izmir	Adnan
01	Milk	Antalya	Volkan
		...	

- We can represent it as this:

$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Yasemin}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Izmir}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Antalya}\rangle \times \langle \text{Volkan}\rangle \cup \ldots$$

TL;DR

- Given a query: $Q_1 = \text{Order} \bowtie_{item} \text{Store} \bowtie_{location} \text{Disp}$
- And a result:

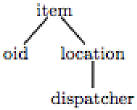
$Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$

oid	item	location	dispatcher
01	Milk	Istanbul	Adnan
01	Milk	Istanbul	Yasemin
01	Milk	Izmir	Adnan
01	Milk	Antalya	Volkan
		...	

- We can represent it as this:

$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Yasemin}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Izmir}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Antalya}\rangle \times \langle \text{Volkan}\rangle \cup \ldots$$

- Or a tree:



TL;DR

- Given a query: $Q_1 = \text{Order} \bowtie_{item} \text{Store} \bowtie_{location} \text{Disp}$
- And a result:

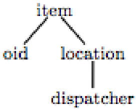
$Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$

oid	item	location	dispatcher
01	Milk	Istanbul	Adnan
01	Milk	Istanbul	Yasemin
01	Milk	Izmir	Adnan
01	Milk	Antalya	Volkan
		...	

- We can represent it as this:

$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Istanbul}\rangle \times \langle \text{Yasemin}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Izmir}\rangle \times \langle \text{Adnan}\rangle \cup$$
$$\langle 01\rangle \times \langle \text{Milk}\rangle \times \langle \text{Antalya}\rangle \times \langle \text{Volkan}\rangle \cup \ldots$$

- Or a tree:



- In exponentially *-space* and $+$*speed* for SPJ queries.

Contents

1. Intro
2. F-Representation
3. F-Trees
4. Query Evaluation
   - Normalisation Operator
   - Swap Operator
   - Cartesian Product Operator
   - Selection Operator
   - Projection Operator
5. Query Optimization
   - Cost of an F-Plan
   - Exhaustive Search
   - Greedy Heuristic
6. Experiments
   - Experiment #1
   - Experiment #2
   - Experiment #2
   - Experiment #3
   - Experiment #4
7. Conclusion
8. Thanks

Example Schema

Orders

oid	item
01	Milk
01	Cheese
02	Melon
03	Cheese
03	Melon

Store

location	item
Istanbul	Milk
Istanbul	Cheese
Istanbul	Melon
Izmir	Milk
Antalya	Milk
Antalya	Cheese

Disp

dispatcher	location
Adnan	Istanbul
Adnan	Izmir
Yasemin	Istanbul
Volkan	Antalya

Produce

supplier	item
Guney	Milk
Guney	Cheese
Dikici	Milk
Byzantium	Melon

Serve

supplier	location
Guney	Antalya
Dikici	Istanbul
Dikici	Izmir
Dikici	Antalya
Byzantium	Istanbul

Figure 1: An example database for a grocery retailer.

Factorized Representations

- Relational algebra expressions.
- Constructed with singleton relations $\langle v \rangle$, the union $\cup$ and product operators $\times$.
- exponentially more succinct than relations they encode.
- fast (constant-delay) enumeration of tuples

Factorized Representations

Definition 1

A factorised representation E, or f-representation for short, over a set $\mathcal{S}$ of attributes and domain $\mathcal{D}$ is a relational algebra expression of the form

- $\emptyset$, empty relation over $\mathcal{S}$;
- $\langle\rangle$, the relation consisting of the nullary tuplem if $\mathcal{S} = \emptyset$;
- $\langle A : a \rangle$, the unary relation with a single tuple with value a, if $\mathcal{S} = \{A\}$ and a is a value in the domain $\mathcal{D}$;
- (E), where E is an f-representation over $\mathcal{S}$;
- $E_1 \cup \cdots \cup E_n$, where each E_i is an f-representation over $\mathcal{S}$;
- $E_1 \times \cdots \times E_n$, where each E_i is an f-representation over $\mathcal{S}_i$ and $\mathcal{S}$ is the disjoint union over all $\mathcal{S}_i$.

Factorized Representations

Singleton

- $\langle A : a \rangle$ is a singleton
- $|E|$ the *size* is the number of singletons
- A single system may have many different f-representations.
- The tuples of a given f-representation can be enumerated with $O(|\mathcal{S}|)$ additional space and delay.

Factorized Representations

Singleton

- $\langle A : a \rangle$ is a singleton
- $|E|$ the *size* is the number of singletons
- A single system may have many different f-representations.
- The tuples of a given f-representation can be enumerated with $O(|\mathcal{S}|)$ additional space and delay.

Factorized Representation

$\langle 01 \rangle \times \langle \text{Milk} \rangle \times \langle \text{Istanbul} \rangle \times \langle \text{Adnan} \rangle \cup$
$\langle 01 \rangle \times \langle \text{Milk} \rangle \times \langle \text{Istanbul} \rangle \times \langle \text{Yasemin} \rangle \cup$
$\langle 01 \rangle \times \langle \text{Milk} \rangle \times \langle \text{Izmir} \rangle \times \langle \text{Adnan} \rangle \cup$
$\langle 01 \rangle \times \langle \text{Milk} \rangle \times \langle \text{Antalya} \rangle \times \langle \text{Volkan} \rangle \cup \ldots$

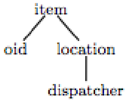
Compact Factorized Representation

$\langle \text{Milk} \rangle \times \langle 01 \rangle \times (\langle \text{Istanbul} \rangle \times (\langle \text{Adnan} \rangle \cup \langle \text{Yasemin} \rangle) \cup$
$\langle \text{Izmir} \rangle \times \langle \text{Adnan} \rangle \cup \langle \text{Antalya} \rangle \times \langle \text{Volkan} \rangle) \cup$
$\langle \text{Cheese} \rangle \times (\langle 01 \rangle \cup \langle 03 \rangle) \times (\langle \text{Istanbul} \rangle \times (\langle \text{Adnan} \rangle \cup \langle \text{Yasemin} \rangle) \cup$
$\langle \text{Antalya} \rangle \times \langle \text{Volkan} \rangle) \cup$
$\langle \text{Melon} \rangle \times (\langle 02 \rangle \cup \langle 03 \rangle) \times \langle \text{Istanbul} \rangle \times (\langle \text{Adnan} \rangle \cup \langle \text{Yasemin} \rangle)$

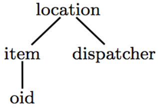
F-Trees
Definition

An *f-tree* for short, over a schema $\mathcal{S}$ of attributes is an **unordered rooted forest** with each node labelled by a non-empty subset of $\mathcal{S}$ such that each attribute of $\mathcal{S}$ labels exactly one node.

Q1 F-Tree



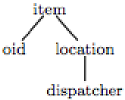
Q1 Alternate F-Tree



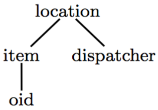
F-Trees
Definition

An *f-tree* for short, over a schema $\mathcal{S}$ of attributes is an **unordered rooted forest** with each node labelled by a non-empty subset of $\mathcal{S}$ such that each attribute of $\mathcal{S}$ labels exactly one node.

Q1 F-Tree



Q1 Alternate F-Tree



- Tree created in $O(|\mathcal{S}| \cdot |R| \cdot \log |R|)$.
- We can go back to R in $O(|\mathcal{S}| \cdot |R|)$.

F-Trees

Queries

Given a query $Q = \pi_{\mathcal{P}}\sigma_{\varphi}(R_1 \times \cdots \times R_n)$

- Given a query Q, an f-tree $\mathcal{T}$ is an f-tree of Q if and only if it satisfies the *path constraint*.
- all dependent attributes can only label nodes along a same root-to-leaf path.
- Nodes of a join are only dependent if they are in the $\mathcal{P}$ list or the same R_i.

F-Trees

Size Bounds

Let $s\mathcal{T}$ be the maximal root-to-leaf path edge cover $\mathcal{T}$.

- For and database **D** and f-tree $\mathcal{T}$ the result size is at most $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})})$

F-Trees

Size Bounds

Let $s\mathcal{T}$ be the maximal root-to-leaf path edge cover $\mathcal{T}$.

- For and database $\mathbf{D}$ and f-tree $\mathcal{T}$ the result size is at most $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})})$
- If we let $s(Q)$ be the minimal $s(\mathcal{T})$ over all f-trees.

F-Trees

Size Bounds

Let $s\mathcal{T}$ be the maximal root-to-leaf path edge cover $\mathcal{T}$.

- For and database $\mathbf{D}$ and f-tree $\mathcal{T}$ the result size is at most $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})})$
- If we let $s(Q)$ be the minimal $s(\mathcal{T})$ over all f-trees.
- $|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{Q})} \ll Q(\mathbf{D})$ — This can mean asymptotically smaller, exponential size and time savings

Contents

1. Intro
2. F-Representation
3. F-Trees
4. Query Evaluation
   - Normalisation Operator
   - Swap Operator
   - Cartesian Product Operator
   - Selection Operator
   - Projection Operator
5. Query Optimization
   - Cost of an F-Plan
   - Exhaustive Search
   - Greedy Heuristic
6. Experiments
   - Experiment #1
   - Experiment #2
   - Experiment #2
   - Experiment #3
   - Experiment #4
7. Conclusion
8. Thanks

F-plans

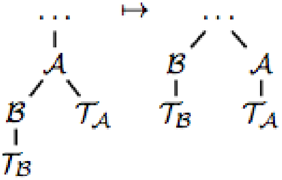
- Any select-project-join query can be evaluated by a sequential composition of operators called an *f-plan*.
- f-trees uniquely identify f-representations $\rightarrow$ operators are in tree form.
- The time complexity of each f-plan operator is $O(|\mathcal{T}|^2 N \log N)$, where N is the sum of input and output f-representations and $\mathcal{T}$ is the input f-tree.

Normalisation Operator[1]

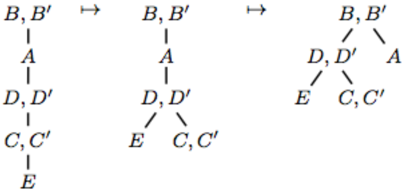
push-up operator $\psi_{\mathcal{B}}$
normalization $\eta(\mathcal{T})$

- A tree is *normalized* if the push up operator can not be applied.
- The push-up operator is applied bottom up.

$\psi_{\mathcal{B}}$



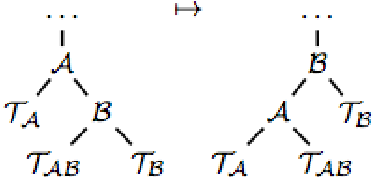
$\eta(\mathcal{T})$ — Normalization



[1] All other operators expect normalized inputs

Swap Operator $\mathcal{X}_{\mathcal{A},\mathcal{B}}$

Exchanges $\mathcal{B}$ with its parent $\mathcal{A}$ and promotes children.

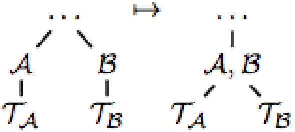


Cartesian Product Operator

$$\mathcal{T}' = \mathcal{T}_\infty \times \mathcal{T}_\in$$

Merge Selection Operator

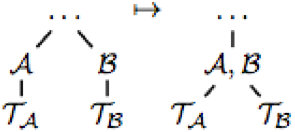
$\mu_{\mathcal{A},\mathcal{B}}$

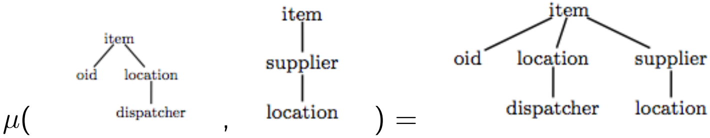
If $\mathcal{A}$ and $\mathcal{B}$ are siblings 

Merge Selection Operator

$\mu_{\mathcal{A},\mathcal{B}}$

If $\mathcal{A}$ and $\mathcal{B}$ are siblings

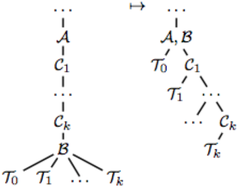




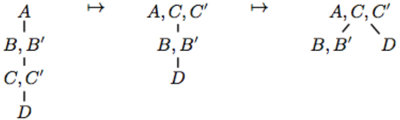
Absorb Selection Operator

$\alpha_{\mathcal{A},\mathcal{B}}$

If $\mathcal{A}$ is an ancestor of $\mathcal{B}$



Example:



Selection with Constant Operator

$\sigma_{A\theta c}$

Filters out all the values in $\langle A : a \rangle$ where $a \neg \theta c$

Projection Operator

π_{-A}

Removes a node from the tree. Only permitted if A is a leaf node or with other attributes.

Contents

1. Intro
2. F-Representation
3. F-Trees
4. Query Evaluation
   - Normalisation Operator
   - Swap Operator
   - Cartesian Product Operator
   - Selection Operator
   - Projection Operator
5. Query Optimization
   - Cost of an F-Plan
   - Exhaustive Search
   - Greedy Heuristic
6. Experiments
   - Experiment #1
   - Experiment #2
   - Experiment #2
   - Experiment #3
   - Experiment #4
7. Conclusion
8. Thanks

Query Optimization

Query optimization has two objectives:

- minimizing the *cost* of computing a factorised query result;
- minimizing the *size* of this output representation.

Query Optimization

- Optimize the f-tree then the f-representation.
- Products and Selections are always evaluated first (cheapest).
- Selection ($A = B$) can only be done if they are on the same path or siblings

Cost of an F-Plan

- use asymptotic bounds of $s(\mathcal{T})$.
- $\mathcal{T}_{\text{initial}} = \mathcal{T}_0 \overset{\omega_1}{\mapsto} \mathcal{T}_1 \overset{\omega_2}{\mapsto} \ldots \overset{\omega_k}{\mapsto} \mathcal{T}_k = \mathcal{T}_{\text{final}}$, with an evaluation time of $O(|\mathbf{D}^{s(f)} \cdot \log |\mathbf{D}|)$, where $s(f) = max(s(\mathcal{T}_0, \ldots s(\mathcal{T}_k))$.

Cost of an F-Plan

- use asymptotic bounds of $s(\mathcal{T})$.
- $\mathcal{T}_{\text{initial}} = \mathcal{T}_0 \overset{\omega_1}{\mapsto} \mathcal{T}_1 \overset{\omega_2}{\mapsto} \ldots \overset{\omega_k}{\mapsto} \mathcal{T}_k = \mathcal{T}_{\text{final}}$, with an evaluation time of $O(|\mathbf{D}^{s(f)} \cdot \log|\mathbf{D}|)$, where $s(f) = max(s(\mathcal{T}_0, \ldots s(\mathcal{T}_k))$.
- , or, use cardinality of estimate from intermediate f-trees using RDBMS style techniques

Exhaustive Search

- Enumerate plans and the one with the shortest path (Dijkstra) between $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ is least cost.
- Search space is $O(n^{4n})$ with $n - p$ attributes in the projection list

Greedy Heuristic

- Only restructures nodes that appear in selections and projections.
- In projection ordering chooses the least cost at each step.
- Computing the tree take $O(|\mathcal{T}|^2)$.

Contents

1. Intro
2. F-Representation
3. F-Trees
4. Query Evaluation
   - Normalisation Operator
   - Swap Operator
   - Cartesian Product Operator
   - Selection Operator
   - Projection Operator
5. Query Optimization
   - Cost of an F-Plan
   - Exhaustive Search
   - Greedy Heuristic
6. Experiments
   - Experiment #1
   - Experiment #2
   - Experiment #2
   - Experiment #3
   - Experiment #4
7. Conclusion
8. Thanks

Implementation

- Compare SQLite and PostgreSQL with FDB and RDB
- for SQLite and PostgreSQL tried to disable expensive writing (try to make it in-memory)
- Implemented in C++
- Optimal f-representation computed with multi-way merge-sort join algorithm.

Experiment Design

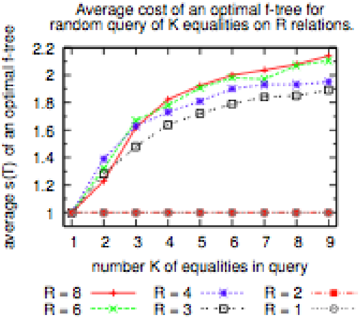
- Generate random data for queries
- Repeat a number of times for optimizations time, execution time, representation sizes and f-plan quality
- Tuples generated independently with a uniform or Zipf distribution.
- Queries are equality joins

Experiment #1

- Average time for optimising a query on flat data.
- $A = 40$ Attributes, $R = 1, \ldots, 8$ relations, $K = 1, \ldots, 9$ selections

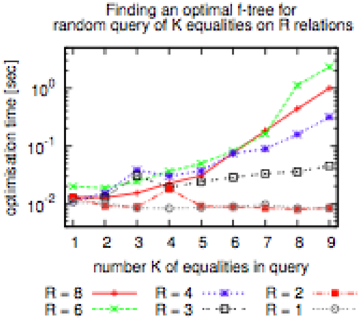
Experiment #1

- Average time for optimising a query on flat data.
- $A = 40$ Attributes, $R = 1, \ldots, 8$ relations, $K = 1, \ldots, 9$ selections

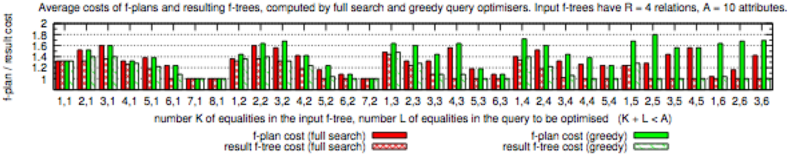


Experiment #1

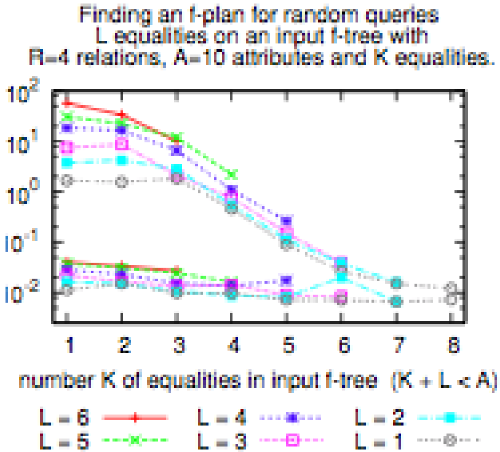
- Average time for optimising a query on flat data.
- $A = 40$ Attributes, $R = 1, \ldots, 8$ relations, $K = 1, \ldots, 9$ selections



Experiment #2



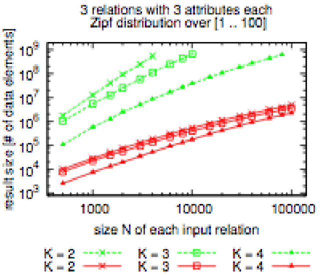
Experiment #2



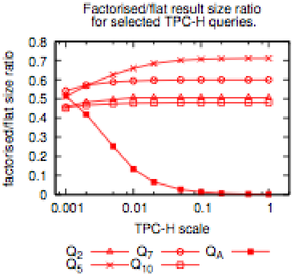
Experiment #3

Compare performance of FDB, RDB, SQLite and PostgreSQL

Results size



Evaluation Time

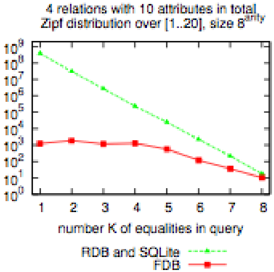


Experiment #3

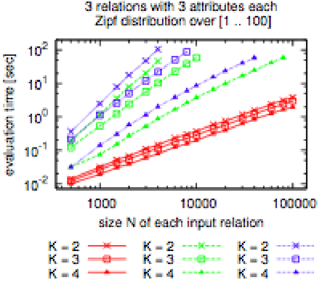
Compare performance of FDB, RDB, SQLite and PostgreSQL.

Each tuple size decreases results size by 20

Results size



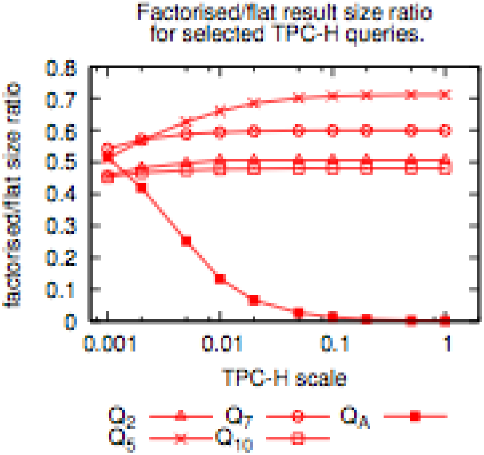
Evaluation time



Experiment #3

Compare performance of FDB, RDB, SQLite and PostgreSQL.

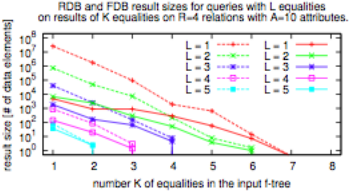
In Q_A is a many-many query. This is highly compressible so the factorized result is small.



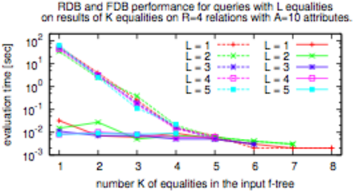
Experiment #4

Experiment over Factorized data

Results size



Evaluation time



Conclusion

- Factorizing Relational Data prevents the explosion of data size for large joins.
- This paper described techniques and experiments to decrease this size and improving query performance.
- For MLNs and other graph DBs that require many joins this can significantly improve query performance.

Thank you

Questions?