November 8, 2013 Database Seminar, U Washington

Factorized Relational Databases

http://www.cs.ox.ac.uk/projects/FDB/

Olteanu and Závodný, University of Oxford

Factorized Representations of Relations

Cust

ckey	name
1	Joe
2	Dan
3	Li
4	Mo

Ord

ckey	okey	date
1	1	1995
1	2	1996
2	3	1994
2	4	1993
3	5	1995
3	6	1996

Item

okey	disc
1	0.1
1	0.2
3	0.4
3	0.1
4	0.4
5	0.1

Consider a query Q joining the three relations above:

$$Q(\text{ckey}, \text{name}, \text{okey}, \text{date}, \text{disc}) \leftarrow$$
$$\text{Cust}(\text{ckey}, \text{name}), \text{Ord}(\text{ckey}, \text{okey}, \text{date}), \text{Item}(\text{okey}, \text{disc})$$

Q

ckey	name	okey	date	disc
1	Joe	1	1995	0.1
1	Joe	1	1995	0.2
2	Dan	3	1994	0.4
2	Dan	3	1994	0.1
2	Dan	4	1993	0.4
3	Li	5	1995	0.1

Factorized Representations of Relations

Q

ckey	name	okey	date	disc
1	Joe	1	1995	0.1
1	Joe	1	1995	0.2
2	Dan	3	1994	0.4
2	Dan	3	1994	0.1
2	Dan	4	1993	0.4
3	Li	5	1995	0.1

A *flat* relational algebra expression of the query result is:

$$\begin{array}{ccccccccccc}
\langle 1\rangle & \times & \langle Joe\rangle & \times & \langle 1\rangle & \times & \langle 1995\rangle & \times & \langle 0.1\rangle & \cup \\
\langle 1\rangle & \times & \langle Joe\rangle & \times & \langle 1\rangle & \times & \langle 1995\rangle & \times & \langle 0.2\rangle & \cup \\
\langle 2\rangle & \times & \langle Dan\rangle & \times & \langle 3\rangle & \times & \langle 1994\rangle & \times & \langle 0.4\rangle & \cup \\
\langle 2\rangle & \times & \langle Dan\rangle & \times & \langle 3\rangle & \times & \langle 1994\rangle & \times & \langle 0.1\rangle & \cup \\
\langle 2\rangle & \times & \langle Dan\rangle & \times & \langle 4\rangle & \times & \langle 1993\rangle & \times & \langle 0.4\rangle & \cup \\
\langle 3\rangle & \times & \langle Li\rangle & \times & \langle 5\rangle & \times & \langle 1995\rangle & \times & \langle 0.1\rangle &
\end{array}$$

It uses relational product ($\times$), union ($\cup$), and unary relations (e.g., $\langle 1\rangle$).

Factorized Representations of Relations

$$
\begin{array}{ccccccccccc}
\langle 1 \rangle & \times & \langle \mathit{Joe} \rangle & \times & \langle 1 \rangle & \times & \langle 1995 \rangle & \times & \langle 0.1 \rangle & \cup \\
\langle 1 \rangle & \times & \langle \mathit{Joe} \rangle & \times & \langle 1 \rangle & \times & \langle 1995 \rangle & \times & \langle 0.2 \rangle & \cup \\
\langle 2 \rangle & \times & \langle \mathit{Dan} \rangle & \times & \langle 3 \rangle & \times & \langle 1994 \rangle & \times & \langle 0.4 \rangle & \cup \\
\langle 2 \rangle & \times & \langle \mathit{Dan} \rangle & \times & \langle 3 \rangle & \times & \langle 1994 \rangle & \times & \langle 0.1 \rangle & \cup \\
\langle 2 \rangle & \times & \langle \mathit{Dan} \rangle & \times & \langle 4 \rangle & \times & \langle 1993 \rangle & \times & \langle 0.4 \rangle & \cup \\
\langle 3 \rangle & \times & \langle \mathit{Li} \rangle & \times & \langle 5 \rangle & \times & \langle 1995 \rangle & \times & \langle 0.1 \rangle &
\end{array}
$$

A *factorized* representation of the query result is:

$$
\begin{aligned}
&\langle 1 \rangle \times \langle \mathit{Joe} \rangle \times \langle 1 \rangle \times \langle 1995 \rangle \times (\langle 0.1 \rangle \cup \langle 0.2 \rangle) \cup \\
&\langle 2 \rangle \times \langle \mathit{Dan} \rangle \times (\langle 3 \rangle \times \langle 1994 \rangle \times (\langle 0.4 \rangle \cup \langle 0.1 \rangle) \cup \langle 4 \rangle \times \langle 1993 \rangle \times \langle 0.4 \rangle) \cup \\
&\langle 3 \rangle \times \langle \mathit{Li} \rangle \times \langle 5 \rangle \times \langle 1995 \rangle \times \langle 0.1 \rangle
\end{aligned}
$$

There are several *algebraically equivalent* factorized representations defined by distributivity of product over union and commutativity of product and union.

Factorized Representations of Relations

$$\langle 1\rangle \times \langle \mathit{Joe}\rangle \times \langle 1\rangle \times \langle 1995\rangle \times (\langle 0.1\rangle \cup \langle 0.2\rangle)\cup$$
$$\langle 2\rangle \times \langle \mathit{Dan}\rangle \times (\langle 3\rangle \times \langle 1994\rangle \times (\langle 0.4\rangle \cup \langle 0.1\rangle) \cup \langle 4\rangle \times \langle 1993\rangle \times \langle 0.4\rangle)\cup$$
$$\langle 3\rangle \times \langle \mathit{Li}\rangle \times \langle 5\rangle \times \langle 1995\rangle \times \langle 0.1\rangle$$

Compactly encode combinations of groups of values.

- Can be exponentially more succinct than the relations they encode.
- Use a mixture of vertical (product) and horizontal data partitioning (union).

Allow for constant-delay enumeration of tuples.

- Unlike general join decompositions and the trivial representation (Q, D).

Boost query performance.

- Queries can be evaluated on factorized data (unlike for general compression).

Spot the Factorized Database!

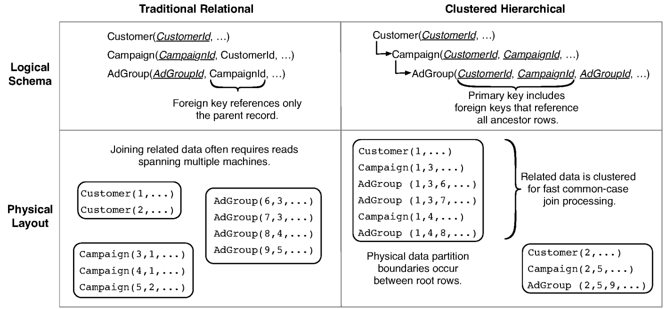
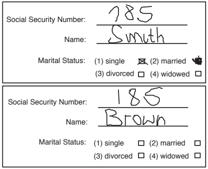


Figure 2: The logical and physical properties of data storage in a traditional normalized relational schema compared with a clustered hierarchical schema used in an F1 database.

F1: A Distributed SQL Database That Scales. PVLDB'13.

- Google's DB supporting their lucrative AdWords business
- Uses factorization of input database to increase data locality for common access patterns
  - DB tables pre-joined following an f-tree defined by key-foreign key constraints.
- Data partitioned across servers into factorization fragments.

Spot the Factorized Database!



$t_1.S$	$t_1.N$	$t_1.M$	$t_2.S$	$t_2.N$	$t_2.M$
185	Smith	1	186	Brown	1
185	Smith	1	186	Brown	2
185	Smith	1	186	Brown	3
185	Smith	1	186	Brown	4
185	Smith	2	186	Brown	1
185	Smith	2	186	Brown	2
185	Smith	2	186	Brown	3
185	Smith	2	186	Brown	4
		⋮			
785	Smith	2	186	Brown	4

Fig. 1. Two completed survey forms and a world-set relation representing the possible worlds with unique social security numbers.

$$\begin{array}{|cc|}\hline t_1.S & t_2.S \\ \hline 185 & 186 \\ 785 & 185 \\ 785 & 186 \\ \hline \end{array} \times \begin{array}{|c|}\hline t_1.N \\ \hline \text{Smith} \\ \hline \end{array} \times \begin{array}{|c|}\hline t_1.M \\ \hline 1 \\ 2 \\ \hline \end{array} \times \begin{array}{|c|}\hline t_2.N \\ \hline \text{Brown} \\ \hline \end{array} \times \begin{array}{|c|}\hline t_2.M \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array}$$

10^{10^6} *Worlds and Beyond: Efficient Representation and Processing of Incomplete Information.* ICDE'07.

Managing a large set of possibilities or choices

- Configuration problems (space of valid solutions)
- Incomplete information (space of possible worlds)

Spot the Factorized Database!

98 5. INTENSIONAL QUERY EVALUATION

5.1.3 READ-ONCE FORMULAS

An important class of propositional formulas that play a special role in probabilistic databases are read-once formulas. We restrict our discussion to the case when all random variables X are Boolean variables.

Φ is called *read-once* if there is a formula Φ' equivalent to Φ such that every variable occurs at most once in Φ'. For example:

$$\Phi = X_1Y_1 \vee X_1Y_2 \vee X_2Y_3 \vee X_2Y_4 \vee X_2Y_5$$

is read-once because it is equivalent to the following formula:

$$\Phi' = X_1(Y_1 \vee Y_2) \vee X_2(Y_3 \vee Y_4 \vee Y_5)$$

Probabilistic Databases. Morgan & Claypool. 2011.

Provenance and probabilistic data

- Compact encoding for large provenance
- Factorization of provenance is used for efficient query evaluation in probabilistic databases.

Spot the Factorized Database!

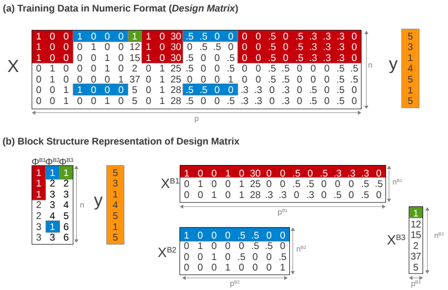


Figure 3: (a) In relational domains, design matrices X have large blocks of repeating patterns (example from Figure 2). (b) Repeating patterns in X can be formalized by a block notation (see section 2.3) which stems directly from the relational structure of the original data. Machine learning methods have to make use of repeating patterns in X to scale to large relational datasets.

Scaling Factorization Machines to Relational Data. PVLDB'13.

- feature vectors for predictive modelling represented as very large design matrices (= relations with high cardinality).
- standard learning algorithms cannot scale on design matrix representation
- use repeating patterns in the design matrix as key to scalability

Key Challenges

1. How compact can factorized query results be?

2. Can such factorizations speed up query evaluation?

Key Results

1. How compact can factorized query results be?

- Asymptotic size bounds for factorizations of query results.
- Characterize queries based on succinctness of their factorized results.

2. Can such factorizations speed up query evaluation?

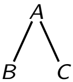
- FDB: a main-memory query engine for factorized relational databases.
- Compute factorizations directly from query and input data.

Factorization Trees

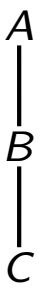
Nesting structure of a factorization.

A *factorization tree* (f-tree) $\mathcal{T}$ over relational schema $\mathcal{S}$ is a **rooted forest with nodes labelled by attributes from** $\mathcal{S}$.

Examples for a relation R over schema $\mathcal{S} = \{A, B, C\}$:



$$\longleftrightarrow \quad \bigcup_{a\in A}\Big(\langle a\rangle \times \Big(\bigcup_{b\in B}\langle b\rangle\Big) \times \Big(\bigcup_{c\in C}\langle c\rangle\Big)\Big).$$

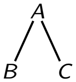


$$\longleftrightarrow \quad \bigcup_{a\in A}\Big(\langle a\rangle \times \Big(\bigcup_{b\in B}\langle b\rangle \times \Big(\bigcup_{c\in C}\langle c\rangle\Big)\Big)\Big).$$

Factorization Trees for Relations

However, not all f-trees work for all relations.

The f-tree



cannot factorize the relation R

R		
A	B	C
1	1	1
1	2	2

because for $A = 1$, the values of B and C are *dependent*:

$$R \text{ cannot be factorized as } \quad \langle 1 \rangle \times \Big(\bigcup_{b \in B} \langle b \rangle\Big) \times \Big(\bigcup_{c \in C} \langle c \rangle\Big).$$

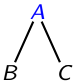
$$\langle 1 \rangle \times \langle 1 \rangle \cup \langle 2 \rangle \times \langle 2 \rangle \quad \neq \quad (\langle 1 \rangle \cup \langle 2 \rangle) \times (\langle 1 \rangle \cup \langle 2 \rangle)$$

Factorization Trees for Relations

Join results have (conditionally) independent attributes.

- studied under the topic of join dependencies.

For instance, the f-tree



always factorizes the result of the join $R(A, B), S(A, C)$.

Factorization Trees for Query Results

For any conjunctive query Q,

we characterize **f-trees that always factorize** the result of Q.

If Q is an equi-join query and $\mathcal{T}$ any f-tree, then

the result $Q(\mathbf{D})$ can be factorized according to $\mathcal{T}$ for *any* database $\mathbf{D}$

iff

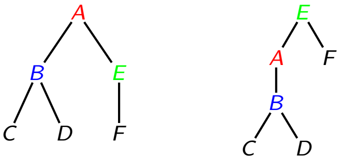
for each relation of Q, all its attributes are on a root-to-leaf path.

If Q has projections, a similar but more complicated result holds.

Factorization Trees for Query Results

Consider the query:

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$



Left f-tree induces the factorization structure:

$$\bigcup_{a\in A}\Big(\langle a\rangle \times \bigcup_{b\in B}\big(\langle b\rangle \times \big(\bigcup_{c\in C}\langle c\rangle\big) \times \big(\bigcup_{d\in D}\langle d\rangle\big)\big) \times \bigcup_{e\in E}\big(\langle e\rangle \times \big(\bigcup_{f\in F}\langle f\rangle\big)\big)\Big)$$

Challenge 1: Succinctness Characterization

Size of Factorized Representations

The *size* of a factorization is the number of its singleton data elements.

$$\left|\big(\langle 1\rangle \cup \langle 2\rangle \cup \langle 3\rangle\big) \times \big(\langle 1\rangle \cup \langle 2\rangle\big)\right| = 5,$$

$$\left|\big(\langle 1\rangle\langle 1\rangle \cup \langle 1\rangle\langle 2\rangle \cup \langle 2\rangle\langle 1\rangle \cup \langle 2\rangle\langle 2\rangle \cup \langle 3\rangle\langle 1\rangle \cup \langle 3\rangle\langle 2\rangle\big)\right| = 12.$$

How much space do we save by factorization?

Size of Factorized Representations: Characterization

For any conjunctive query Q there is a number $s(Q)$ such that

For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits a factorization of size $O(|\mathbf{D}|^{s(Q)})$.

- **Best possible bound** for factorizations whose nesting structures (i.e., f-trees) are inferred from Q, *without looking at* $\mathbf{D}$.

  There exists $\mathbf{D}$ such that all factorizations over f-trees are $\Omega(|\mathbf{D}|^{s(Q)})$.

Size of Factorized Representations: Characterization

For any conjunctive query Q there is a number $s(Q)$ such that

For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits a factorization of size $O(|\mathbf{D}|^{s(Q)})$.

- **Best possible bound** for factorizations whose nesting structures (i.e., f-trees) are inferred from Q, *without looking at* $\mathbf{D}$.

  There exists $\mathbf{D}$ such that all factorizations over f-trees are $\Omega(|\mathbf{D}|^{s(Q)})$.

- Worst-case optimality also for computing the factorization in case of queries without projections.

- Instance-optimal factorization of relations (i.e., dependent on $\mathbf{D}$) is hard.
  - progress so far: consider functional dependencies, sizes of relations, efficient heuristics that do not require guiding f-trees.

Sizes: Flat vs. Factorized Query Results

- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorization of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

$$1 \leq s(Q) \leq \rho^*(Q) \leq |Q|$$

There are classes of queries with $s(Q) \ll \rho^*(Q)$.

Intuition for Asymptotic Bounds

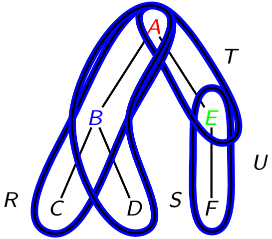
- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorization of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

What are $\rho^*(Q)$ and $s(Q)$?

Intuition – Edge Covers

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

The hypergraph of Q:



First observation:

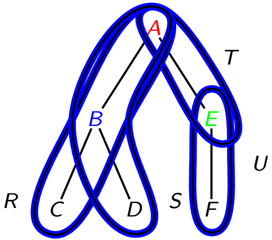
- Cover all attributes by k relations $\Rightarrow |Q(\mathbf{D})| \leq |\mathbf{D}|^k$.
- Set of m independent attributes $\Rightarrow$ construct $\mathbf{D}$ with $|Q(\mathbf{D})| \sim |\mathbf{D}|^m$.

$$\max_m = \mathrm{IndependentSet}(Q) \leq \mathrm{EdgeCover}(Q) = \min_k$$

Intuition – Fractional Edge Covers

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

The hypergraph of Q:



[AGM'08]:

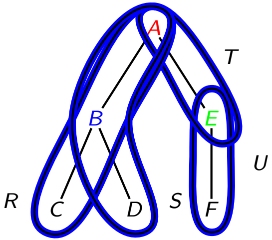
- *Fractional* edge cover of Q with weight k $\Rightarrow |Q(\mathbf{D})| \leq |\mathbf{D}|^k$.
- *Fractional* independent set of weight $m \Rightarrow$ construct $\mathbf{D}$ with $|Q(\mathbf{D})| \sim |\mathbf{D}|^m$.

By linear programming duality:

$$\max_m = \mathrm{FractionalIndependentSet}(Q) = \mathrm{FractionalEdgeCover}(Q) = \min_k$$

Example

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

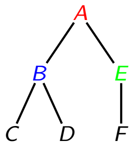


- Relations R, S, U cover the whole query.
  $\mathrm{FractionalEdgeCover}(Q) \leq 3$
- Each of the nodes C, D, and F must be covered by separate relations.
  $\mathrm{FractionalIndependentSet}(Q) \geq 3$

$\Rightarrow \rho^*(Q) = 3$

$\Rightarrow |Q(\mathbf{D})| = O(|\mathbf{D}|^3)$ and for some inputs $|Q(\mathbf{D})| = \Theta(|\mathbf{D}|^3)$.

Intuition – Size of Factorizations



$$\bigcup_{a\in A}\Big(\langle a\rangle \times \bigcup_{b\in B}\Big(\langle b\rangle \times \big(\bigcup_{c\in C}\langle c\rangle\big) \times \big(\bigcup_{d\in D}\langle d\rangle\big)\Big) \times \bigcup_{e\in E}\Big(\langle e\rangle \times \big(\bigcup_{f\in F}\langle f\rangle\big)\Big)\Big)$$

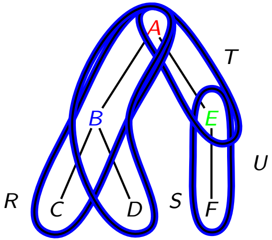
Attributes only depend on their ancestor attributes in the f-tree

- F only depends on E and A.
- One $\langle f\rangle$ for each $(a, e, f) \in Q(\mathbf{D})$.
- The number of F-singletons is $|\pi_{A,E,F}(Q(\mathbf{D}))|$.

Size of factorization = sum of sizes of results of **subqueries along paths**.

Example

$$Q(A,B,C,D,E,F) \leftarrow R(A,B,C), S(A,B,D), T(A,E), U(E,F).$$



- Path A, E, F has fractional edge cover 2.
  $\Rightarrow$ The number of F-singletons is $\leq |\mathbf{D}|^2$, but can be $\sim |\mathbf{D}|^2$.
- All other root-to-leaf paths have fractional edge cover 1.
  $\Rightarrow$ The number of other singletons is $\leq |\mathbf{D}|$.

$s(Q) = 2$ $\Rightarrow$ Factorization size $\sim |\mathbf{D}|^2$

Recall that $\rho^*(Q) = 3$ $\Rightarrow$ Flat size $\sim |\mathbf{D}|^3$

Size: Flat vs. Factorized Query Results

- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorization of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

- $\rho^*(Q)$ = fractional edge cover number of the **entire query**.
- $s(Q)$ = fractional edge cover number of **root-to-leaf paths in best f-tree**.

$$1 \leq s(Q) \leq \rho^*(Q) \leq |Q|$$

There are classes of queries with $s(Q) \ll \rho^*(Q)$.
(There are classes of queries with $s(Q) = 1$ and $\rho^*(Q) = |Q|$.)

Challenge 2: Speed Up Query Evaluation

The FDB Query Engine: Support and Design Choices

Current support

- flat/factorized $\rightarrow$ flat/factorized query processing.
- queries with selection, projection, equi-join, agg, group-by, order-by, limit.
- data types: int, double, string (mapped to int).
- Research prototype, still lots to do and improve...

Design choices

- Implemented in C++ for in-memory use
- Single computation node
- Block-oriented execution model
  - *factorized-table-at-a-time* processing
- Factorizations represented as in-memory trees.
  - one inner node per n-ary union/product operations.
  - all leaves that are children of a node stored in a sorted array.

Wish list

- distributed computation, factorized data shipped between nodes if necessary.
- order-preserving value compression in addition to structure compression.

The FDB Query Engine: Challenges

Query optimization has two tasks:

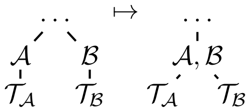
1. Find a good query evaluation plan and
2. Find a good **factorization plan**.

Query evaluation:

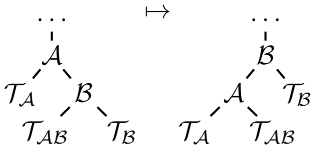
- Operators defined as mappings between f-trees/factorizations.
- New operators for locally restructuring the factorization.

A Glimpse at Query Operators

- **Absorb** and **Merge** (depicted) for selections $A = B$



- **Swap** to restructure by swapping a child with its parent



- **Filter** for selections with constant
- **Project** for discarding one leaf attribute
- Group-by and order-by supported via restructuring.
- Further operators for aggregates and limit.

Experimental Evaluation

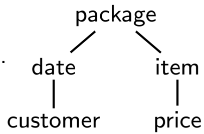
Natural use cases for FDB:

- Static data with $\boxed{\text{many-to-many relationships}}$
- Queries on factorized materialized views
  - factorize once, speed up all subsequent processing

Data set:

- (Factorized) Materialized view R = Orders $\bowtie$ Items $\bowtie$ Packages
- Scale s:
  - $800s$ order dates, $100\sqrt{s}$ items, $40\sqrt{s}$ packages, $20\sqrt{s}$ items.
  - $80s$ order dates/customer, 2 orders/date.
  - For $s = 32$, R has:
    - 280M tuples (1.4G singletons) and
    - 4.2M singletons when factorized.

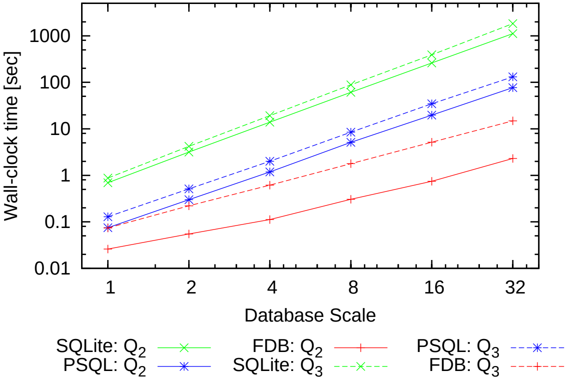
F-tree:



Queries:

- five group-by + aggregate on top of the materialized view R.
- relational engines: sort+scan of R.
- FDB: various degree of restructuring necessary.

Performance for Aggregates



- Competitors: SQLite 3.7.7 and PostgreSQL 9.1.8 (I/O cost $\rightarrow$ 0).
- Aggregates on top of materialized view R

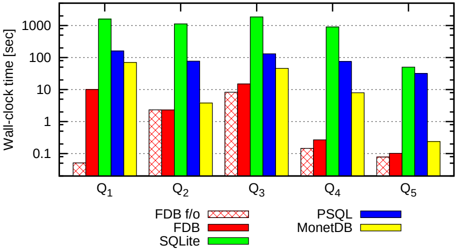
$$Q_2 = \varpi_{\text{customer; revenue} \leftarrow \text{sum(price)}}(R)$$

$$Q_3 = \varpi_{\text{date, package; sum(price)}}(R)$$

- Result is flat for all engines.

Performance for Aggregates

- Same dataset, now only for scale 32.
- FDB f/o = FDB with factorized output.



$$Q_1 = \varpi_{\text{package, date, customer; revenue} \leftarrow \text{sum(price)}}(R)$$

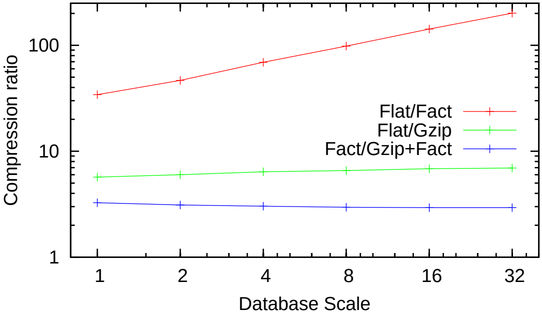
$$Q_2 = \varpi_{\text{customer; revenue} \leftarrow \text{sum(price)}}(R)$$

$$Q_3 = \varpi_{\text{date, package; sum(price)}}(R)$$

$$Q_4 = \varpi_{\text{package; sum(price)}}(R)$$

$$Q_5 = \varpi_{\text{sum(price)}}(R)$$

Materialized Views: Factorized vs. Gzipped



Setup:

- Flat = flat relation R in CSV text format
- Gzip (compression level 6) outputs binary format
- Fatorized output in text format (each digit represented as one byte character)

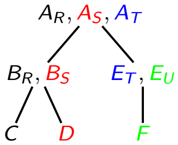
Observations:

- Gzip does not exploit repetitions!
- Factorizations can be arbitrarily more succinct than gzipped relations.
- Gzipping factorizations only improves the compression by a constant factor.

Thanks!

More Succinct Representations: DAG

Avoid repeating identical expressions: store them once and use pointers.



$$\bigcup_{a \in A_R, A_S, A_T} \left[\langle a \rangle \times \cdots \times \bigcup_{e \in E_T, E_U} \left(\langle e \rangle \times \left(\bigcup_{f \in F} \langle f \rangle\right)\right)\right]$$

- Node $\{F\}$ only depends on $\{E_T, E_U\}$.
- A fixed $\langle e \rangle$ binds with the same $\bigcup_{f \in F} \langle f \rangle$ for each $\langle a \rangle$.

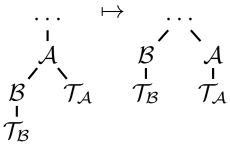
  $\Rightarrow$ store the mapping $\langle e \rangle \mapsto \bigcup_{f \in F} \langle f \rangle$ separately.

$$\bigcup_{a \in A_R, A_S, A_T} \left[\langle a \rangle \times \cdots \times \bigcup_{e \in E_T, E_U} \left(\langle e \rangle \times U_e\right)\right]; \qquad \left\{U_e = \bigcup_{f \in F} \langle f \rangle\right\}$$

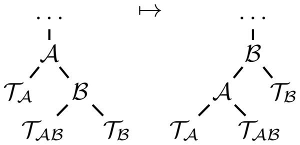
Some Query Operators in More Detail

Restructuring operators

- **Normalisation** factors out expressions common to all terms of a union. Example: f-tree nodes $\mathcal{A}$ and $\mathcal{B}$ do not have dependent attributes.



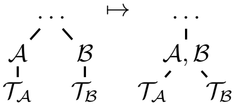
- **Swap** exchanges a node with its parent while preserving normalisation. Example: $\mathcal{T}_{\mathcal{A}}$ depends on $\mathcal{A}$ only, $\mathcal{T}_{\mathcal{B}}$ depends on $\mathcal{B}$ only, $\mathcal{T}_{\mathcal{AB}}$ depends on both $\mathcal{A}$ and $\mathcal{B}$



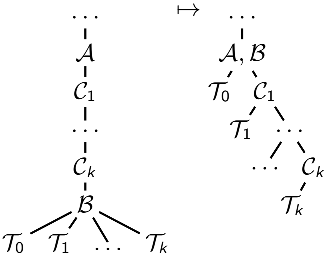
Some Query Operators in More Detail

Selection operators $A = B$, where A and B label nodes $\mathcal{A}$ and $\mathcal{B}$ respectively.

- **Merge** siblings $\mathcal{A}$ and $\mathcal{B}$ into a single node



- **Absorb** $\mathcal{B}$ into its ancestor $\mathcal{A}$. Example: $\mathcal{T}_i$ depends on $\mathcal{B}$ and $\mathcal{C}_i$



Select $A\theta c$ does not change the f-tree; it removes from the factorization all products containing A-singletons $\langle a \rangle$ for which $a \neg\theta c$.

Query Optimization

Goal: Find the best f-plan = query **and** factorization plan

- Optimal factorization of the query result
- Minimal computation cost, i.e., the sizes of intermediate results
- Cost computation based on $s(Q)$ or cardinality and selectivity estimates

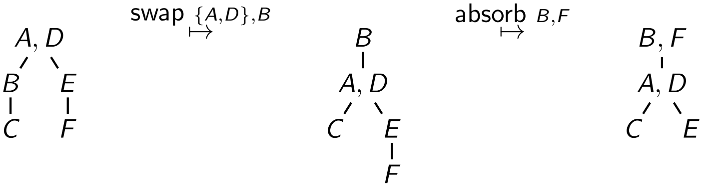
Search space defined by

- selection operators may require several swaps before application,
- choice of selection operators and f-tree transformations for each join,
- choice of order for join conditions,
- projection push-downs.

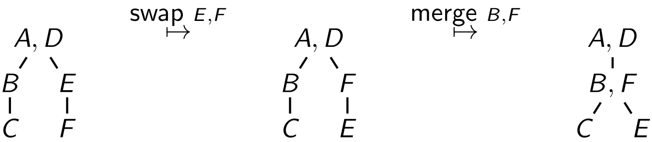
Query Optimization: Example

Build f-plan for selection $B = F$ on the leftmost f-tree, with dependencies $\{A, B, C\}$ and $\{D, E, F\}$.
Alternative f-plans (cost given by $\max s(\mathcal{T}_i)$ over all $\mathcal{T}_i$'s in the f-plan):

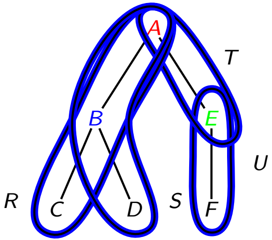
1. Input and output f-trees with cost 1, intermediate with cost 2



2. All three f-trees have cost 1.



Intuition – Fractional Edge Covers



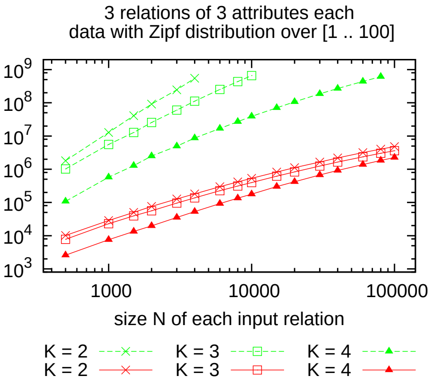
For a query $Q = R_1 \bowtie \cdots \bowtie R_n$, the *fractional edge cover number* $\rho^*(Q)$ is the cost of an optimal solution to the linear program

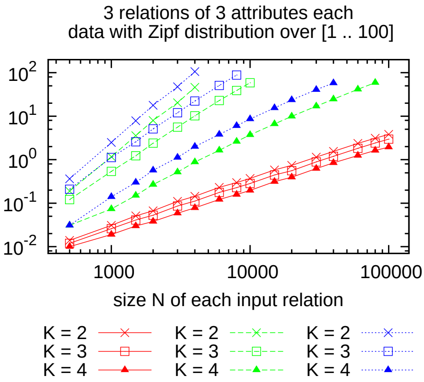
$$\begin{array}{lll} \text{minimizing} & \sum_i x_{R_i} & \\ \text{subject to} & \sum_{i:R_i \text{ has attribute } A} x_{R_i} \geq 1 & \text{for all attributes } A, \\ & x_{R_i} \geq 0 & \text{for all } R_i. \end{array}$$

- x_{R_i} is the weight of relation R_i.
- Each attribute has to be covered by relations with sum of weights ≥ 1.
- In the non-weighted edge cover, the variables $x_{R_i} \in \{0, 1\}$

Performance for conjunctive queries

Performance follows the size gap between flat and factorized input/output data.





- Left: Size gap (in number of singletons).
- Right: performance gap (in seconds; wall-clock time).
- K = number of equi-joins.
- Query engines: FDB, RDB, SQLite.