Factorised Relational Databases

Dan Olteanu and Jakub Závodný, University of Oxford
Factorised Representations of Relations

<table>
<thead>
<tr>
<th>Cust</th>
<th>Ord</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckey name</td>
<td>ckey okey date</td>
<td>okey disc</td>
</tr>
<tr>
<td>1</td>
<td>Joe</td>
<td>1 1 1995</td>
</tr>
<tr>
<td>2</td>
<td>Dan</td>
<td>1 2 1996</td>
</tr>
<tr>
<td>3</td>
<td>Li</td>
<td>2 3 1994</td>
</tr>
<tr>
<td>4</td>
<td>Mo</td>
<td>2 4 1993</td>
</tr>
</tbody>
</table>

Consider a query joining the three relations above:

| Cust ✶_{ckey} Ord ✶_{okey} Item |
| ckey name okey date disc |
| 1 | Joe | 1 1 1995 0.1 |
| 1 | Joe | 1 2 1996 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 |
Factorised Representations of Relations

\begin{center}
\begin{tabular}{cccc}
\hline
\text{Cust} & \text{Ord} & \text{Item} \\
\text{ckey} & \text{okey} & \text{Item} \\
\hline
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 \\
\hline
\end{tabular}
\end{center}

A \textit{flat} relational algebra expression of the query result is:

\begin{align*}
\langle 1 \rangle \times \langle Joe \rangle \times \langle 1 \rangle \times \langle 1995 \rangle \times \langle 0.1 \rangle \\ \langle 1 \rangle \times \langle Joe \rangle \times \langle 1 \rangle \times \langle 1995 \rangle \times \langle 0.2 \rangle \\ \langle 2 \rangle \times \langle Dan \rangle \times \langle 3 \rangle \times \langle 1994 \rangle \times \langle 0.4 \rangle \\ \langle 2 \rangle \times \langle Dan \rangle \times \langle 3 \rangle \times \langle 1994 \rangle \times \langle 0.1 \rangle \\ \langle 2 \rangle \times \langle Dan \rangle \times \langle 4 \rangle \times \langle 1993 \rangle \times \langle 0.4 \rangle \\ \langle 3 \rangle \times \langle Li \rangle \times \langle 5 \rangle \times \langle 1995 \rangle \times \langle 0.1 \rangle
\end{align*}

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

A factorised representation (or f-representation) of the query result is:

\[
\langle 1 \rangle \times \langle Joe \rangle \times \langle 1 \rangle \times \langle 1995 \rangle \times (\langle 0.1 \rangle \cup \langle 0.2 \rangle) \cup \\
\langle 2 \rangle \times \langle Dan \rangle \times \langle 3 \rangle \times \langle 1994 \rangle \times (\langle 0.4 \rangle \cup \langle 0.1 \rangle) \cup \\
\langle 2 \rangle \times \langle Dan \rangle \times \langle 4 \rangle \times \langle 1993 \rangle \times \langle 0.4 \rangle \cup \\
\langle 3 \rangle \times \langle Li \rangle \times \langle 5 \rangle \times \langle 1995 \rangle \times \langle 0.1 \rangle
\]

There are several algebraically equivalent factorised representations defined by distributivity of product over union and commutativity of product and union.
Applications of Factorised Representations

- Succinct representation of large intermediate/final results in query evaluation
  - Equality joins induce regularity in the query result and make it factorisable.

- Provenance databases and probabilistic databases
  - Compact encoding of large provenance (10MB/record in GeneOntology DB)
  - Factorisation of provenance polynomials is used for efficient query evaluation.

- Incompleteness and non-determinism (choice) in design specifications
  - Whenever we need to deal with a large space of possibilities or choices.

- Compiled relational databases
  - Compile data into compact factorised representation to speed up processing of many subsequent queries.

- Configuration problems
  - Represent the space of feasible solutions (valid combinations of components)
Properties of Factorised Representations of Relations

Factorised Representations

- Are relational algebra expressions.
- Can be exponentially more succinct than the relations they encode.
- Allow for fast (constant-delay) enumeration of tuples
- Reduce data redundancy and boost query performance using a mixture of
  - vertical data partitioning (product) and
  - horizontal data partitioning (union).
Key Challenges and Talk Overview

1. Characterise conjunctive queries based on succinctness of their factorised results.

2. Build a relational DBMS that uses f-representations at the physical layer.

Overview of the Rest of the Talk:

- Factorisations whose nesting structures are inferred from the query
- Tight bounds on size and readability of factorised query results
- FDB: Query engine for factorised databases
Factorisation Trees

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

$\mathcal{T}$ defines a nesting structure for f-representations of relations over $S$.

Example f-trees and corresponding factorisations over $S = \{A, B, C\}$:

![Diagram](image-url)

\[
\begin{align*}
A & \quad \overset{\leftrightarrow}{\longrightarrow} \quad \bigcup_{a \in A} \langle a \rangle \times \left( \bigcup_{b \in B} \langle b \rangle \right) \times \left( \bigcup_{c \in C} \langle c \rangle \right) .
\end{align*}
\]

\[
\begin{align*}
A & \quad \overset{\leftrightarrow}{\longrightarrow} \quad \bigcup_{a \in A} \langle a \rangle \times \left( \bigcup_{b \in B} \langle b \rangle \times \left( \bigcup_{c \in C} \langle c \rangle \right) .
\end{align*}
\]
Factorisation Trees for Relations

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

The f-tree

```
  A
 / \
B   C
```

cannot factorise the relation $R$

```
\[
\begin{array}{ccc}
A & B & C \\
1 & 1 & 1 \\
1 & 2 & 2 \\
\end{array}
\]
```

because

- For $A = 1$, the values of $B$ and $C$ are dependent, i.e.,
- Relation $\pi_{B,C} \sigma_{A=1}(R)$ cannot be factorised as $(\bigcup_{b \in B} \langle b \rangle) \times (\bigcup_{c \in C} \langle c \rangle)$:

\[
[(\langle 1 \rangle \cup \langle 2 \rangle) \times (\langle 1 \rangle \cup \langle 2 \rangle)] \neq [(\langle 1 \rangle \times \langle 1 \rangle) \cup (\langle 2 \rangle \times \langle 2 \rangle)]
\]
Factorisation Trees for Query Results

We statically infer from queries which f-trees *always* work for their results.

For a query $Q$ (without projections) and f-tree $T$

the result $Q(D)$ can be factorised according to $T$ for *any* database

iff

for all relations of $Q$, all attributes are on a single root-to-leaf path.

Similar but more involved condition holds for arbitrary conjunctive queries.
Factorisation Trees for Query Results

Consider query $Q = \sigma_\phi(R \times S \times T \times U)$, with

- schemas $R(A_R, B_R, C)$, $S(A_S, B_S, D)$, $T(A_T, E_T)$, and $U(E_U, F)$,
- condition $\phi = (A_R = A_S = A_T, B_R = B_S, E_T = E_U)$.

F-representations modelled on the left f-tree have the structure:

$$
\bigcup_{a \in A_R, A_S, A_T} \langle a \rangle \times \bigcup_{b \in B_R, B_S} \langle b \rangle \times \left( \bigcup_{c \in C} \langle c \rangle \right) \times \left( \bigcup_{d \in D} \langle d \rangle \right) \times \bigcup_{e \in E_T, E_U} \langle e \rangle \times \left( \bigcup_{f \in F} \langle f \rangle \right)
$$
Size of Factorised Representations

The size of an f-representation is the number of its singleton data elements.

\[ |(\langle 1 \rangle \cup \langle 2 \rangle \cup \langle 3 \rangle)(\langle 1 \rangle \cup \langle 2 \rangle)| = 5, \]
\[ |(\langle 1 \rangle \langle 1 \rangle \cup \langle 1 \rangle \langle 2 \rangle \cup \langle 2 \rangle \langle 1 \rangle \cup \langle 2 \rangle \langle 2 \rangle \cup \langle 3 \rangle \langle 1 \rangle \cup \langle 3 \rangle \langle 2 \rangle)| = 12. \]

The two sizes above differ, although

\[ (\langle 1 \rangle \cup \langle 2 \rangle \cup \langle 3 \rangle)(\langle 1 \rangle \cup \langle 2 \rangle) = (\langle 1 \rangle \langle 1 \rangle \cup \langle 1 \rangle \langle 2 \rangle \cup \langle 2 \rangle \langle 1 \rangle \cup \langle 2 \rangle \langle 2 \rangle \cup \langle 3 \rangle \langle 1 \rangle \cup \langle 3 \rangle \langle 2 \rangle) \]

How much space do we save by factorisation?
Tight Bounds on the Size of Factorised Representations

Given a query $Q$, for any f-tree $T$ of $Q$ there is a rational number $s(T)$ such that:

- For any database $D$, the factorisation of $Q(D)$ over $T$ has size $O(|D|^{s(T)})$.
- There exist arbitrarily large databases $D$ for which the factorisation of $Q(D)$ over $T$ has size $\Theta(|D|^{s(T)})$.

The parameter $s(T)$ is

- a feasible solution to a linear program,
- the fractional edge cover number of a sub-query of $Q$.
  
  - this sub-query depends on the shape of $T$.
  - $1 \leq s(T) \leq |Q|$.
Example of Computing $s(T)$

Consider the following f-tree $T$.

- Attributes with the same colour belong to the same input relation.

$$
\begin{array}{c}
A_R, A_S, A_T \\
B_R, B_S & E_T, E_U \\
C & D & F
\end{array}
$$

Number of relations covering the path from root to any node $X$:

- For each node $X$ except for $F$, this number is 1.
- For node $F$, this number is 2.

$s(T)$ is the maximum of the number of covering relations for each node.

- Thus, $s(T) = 2$. 
Tight Bounds on the Size of Factorised Representations

Size bounds for f-trees can be lifted to queries by finding an optimal f-tree:

\[ s(Q) = \min_T s(T). \]

\( s(Q) \) characterises queries by factorisability of their results.

- For any database \( D \), there is a factorisation of \( Q(D) \) with size \( O(|D|^{s(Q)}) \).
- For f-trees derived from \( Q \), this bound is best possible.
Readability of Factorised Representations

- Assume we annotate tuples by distinct variables (\(\equiv\) provenance, keys).
- Factorised representations can be seen as polynomials over such variables
  - \(\cup\) becomes sum (+) and \(\times\) becomes product (\(\cdot\)).

Readability:
- A representation \(\Phi\) is read-\(k\) if the maximum number of occurrences of any variable in \(\Phi\) is \(k\).
- The readability of \(\Phi\) is the smallest number \(k\) such that there is a read-\(k\) representation equivalent to \(\Phi\).
- Readability has been proposed in the context of factorisation of Boolean functions [Golumbic et al.’06].

Example: \(\psi_1\) is read-3 and \(\psi_2\) is read-1. They are equivalent and have readability one.

\[
\psi_1 = c_1 o_1 i_1 + c_1 o_1 i_2 + c_2 o_3 i_3 + c_2 o_3 i_4 + c_2 o_4 i_5 + c_3 o_5 i_6.
\]

\[
\psi_2 = c_1 o_1 (i_1 + i_2) + c_2 (o_3 (i_3 + i_4) + o_4 i_5) + c_3 o_5 i_6.
\]
Two Readability Dichotomies

1. Let $Q$ be a query.
   - If $Q$ is hierarchical, the readability of $Q(D)$ for any database $D$ is bounded by a constant.
   - If $Q$ is non-hierarchical, for any f-tree $T$ of $Q$ there exist arbitrarily large databases $D$ such that $T(D)$ is read-$\Omega(|D|)$.

2. Let $Q$ be a query without repeating relation symbols.
   - If $Q$ is hierarchical, the readability of $Q(D)$ is 1 for any database $D$.
   - If $Q$ is non-hierarchical, there exist arbitrarily large databases $D$ such that the readability of $Q(D)$ is $\Omega(\sqrt{|D|})$. 
What are these hierarchical queries?

Hierarchical query $Q$:
- For any two equivalence classes of attributes in $Q$, either their sets of relation symbols are disjoint, or one is included in the other.

This is a key property for query characterisation in many applications:
- In probabilistic databases, any tractable non-repeating conjunctive query is hierarchical; non-hierarchical queries are intractable [Suciu&Dalvi’07].
- In the finite cursor machine model of computation [Grohe et al’07], any query that can be evaluated in one pass is hierarchical; non-hierarchical queries need more passes.
  - Assumption: we are allowed to first sort the input relations.
- In the Massively Parallel computation model, any query that can be evaluated with one synchronisation step is hierarchical. [Suciu et al’11]
Readability Width of a Query

There is a rational number \( r(Q) \) with properties similar to those of \( s(Q) \):

- For any database \( D \), the readability of the query result \( Q(D) \) is at most \( M \cdot |D|^{r(Q)} \), where \( M \) is the max number of repeating relation symbols in \( Q \).

- For any f-tree \( T \) of \( Q \) there exist arbitrarily large databases \( D \) such that the f-representation \( T(D) \) is at least \( \text{read-}(|D|/|Q|)^{r(Q)} \).

- \( r(Q) = 0 \) for hierarchical queries \( Q \) only and \( r(Q) > 0 \) for all others.

- \( r(Q) \) defines the \textit{readability width} of \( Q \).
FDB: A Query Engine for Factorised Databases

- Uses f-representations to encode relational data

Query evaluation
- Relational operators: selection, projection, product
- New operators for restructuring factorisations
- Any query can be evaluated by a sequence of operators

Query optimisation
- Find the best query and factorisation plan

Implementation of an in-memory engine in C++
- flat/factorised data → flat/factorised data

Experimental evaluation with FDB and relational engines
- Factorised query results up to 6 orders of magnitude smaller than equivalent relations.
- FDB up to 5 orders of magnitude faster than PostgreSQL/SQLite/our in-memory relational engine.
Thanks!
Query Operators

Restructuring operators

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

```
  ... → ...
   \_   \_   \_   \_
  A     B     A     A
   \_ \_ \_   \_ \_ \_
   B  T_A  B  T_B  T_A
   \_ \_ \_
   T_B  
```

- **Swap** exchanges a node with its parent while preserving normalisation. Example: $T_A$ depends on $A$ only, $T_B$ depends on $B$ only, $T_{AB}$ depends on both $A$ and $B$

```
  ... → ...
   \_   \_   \_   \_
  A     B     A     A
   \_ \_ \_   \_ \_ \_
   T_A  B  T_A  T_B
   \_ \_ \_ \_ \_ \_
   T_{AB}  T_B  T_A  T_{AB}
```
Query Operators

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

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

Further query algebra operators

- **Cartesian product** of two f-trees is their forest

- **Projection** on attribute list $\bar{A}$ removes from the f-tree all attributes but those in $\bar{A}$; empty leaf nodes are removed.
  - The projection operation is more involved if the resulting f-tree must not allow f-representations with duplicates.

- Work in progress: **Order-by**, **Group-by**, **Aggregates**.
Query Optimisation

Goal: Find the best f-plan = query and factorisation plan
- Optimal f-representation 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 Optimisation: 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 $s(T_i)$ over all $T_i$'s in the f-plan):

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

   - swap $\{A,D\}, B$
   - absorb $B, F$

   ![Intermediate F-Trees]

2. All three f-trees have cost 1.

   - swap $E, F$
   - merge $B, F$
Experimental Evaluation

Query optimisation. $K$ equalities on $R$ relations with $A = 40$ attributes.

- exhaustive search used above.
- heuristics perform up to 4 orders of magnitude better, the cost differs by at most 0.5.
Experimental Evaluation

Query evaluation on flat data: FDB vs. Relational DB (RDB).

- The trend is the same for time performance.
Experimental Evaluation

Query evaluation on factorised data: FDB (dotted lines) vs. RDB (solid lines).

The trend is the same for size.
In search for the rational number $s(Q)$ - step 1

Disclaimer: Discussion for queries without projection!

For any attribute $A$ in an f-tree $T$, the number of occurrences of $A$-values in the factorisation of $Q(D)$ over $T$ is $|\pi_{\text{path}(A)}(Q(D))|$. 

Example

- $\text{path}(F) = \{A_R, A_S, A_T, E_T, E_U, F\}$.
- The number of occurrences of $F$-values is then $|\pi_{\text{path}(F)}(Q(D))|$. 

Next step:

- The trouble is that $\pi_{\text{path}(A)}(Q(D))$ requires to know $Q(D)$.
- We would like to express it as a function of $Q$ and $D$. 

In search for the rational number $s(Q)$ - step 2

Restrict $Q = \pi_P(\sigma_\phi(R_1 \times \cdots \times R_n))$ and $D$ to the attributes in $\text{path}(A)$:

- $Q_A = \sigma_{\phi_{\text{path}(A)}}(\pi_{\text{path}(A)}R_1 \times \cdots \times \pi_{\text{path}(A)}R_n)$,
- $D_A$ obtained by projecting $D$ onto $\text{path}(A)$.

Then number of $A$-values $= |\pi_{\text{path}(A)}(Q(D))| \leq |Q_A(D_A)|$.

Rough estimate:

- Cover all attributes of $Q_A$ by $k \leq |Q_A|$ relations.
- Then, $|Q_A(D_A)| \leq |D|^k$.
- Best $k$ is the edge cover number of the hypergraph of $Q_A$.

Better estimate:

- From edge cover number $k$ to fractional edge cover number $\rho^*(Q_A)$.
In search for the rational number \( s(Q) \) - step 3

For a query \( Q = \sigma_\phi(R_1 \times \cdots \times R_n) \), the \textit{fractional edge cover number} \( \rho^*(Q) \) is the cost of an optimal solution to the linear program with variables \( \{x_{R_i}\}_{i=1}^n \):

\[
\begin{align*}
\text{minimising} & \quad \sum_i x_{R_i} \\
\text{subject to} & \quad \sum_{i: R_i \in \text{rel}(\mathcal{A})} x_{R_i} \geq 1 \text{ for all attribute classes } \mathcal{A}, \\
& \quad x_{R_i} \geq 0 \quad \text{for all } R_i.
\end{align*}
\]

- \( x_{R_i} \) is the weight of relation \( R_i \).
- \( \text{rel}(\mathcal{A}) \) are relations with attributes in \( \mathcal{A} \).
- Each node \( \mathcal{A} \) has to be covered by relations in \( \text{rel}(\mathcal{A}) \) such that the sum of the weights of these relations is greater than 1.
- The objective is to minimise the sum of the weights of all relations.
- In the non-weighted edge cover, the variables \( x_{R_i} \) can only be assigned the values 0 and 1.

Then \( |Q(D)| \leq |D|^{\rho^*(Q)} \) for all databases \( D \). [Atserias, Grohe, Marx; FOCS’08]
In search for the rational number \( s(Q) \) - step 4

1. Number of \( A \)-values = \( |\pi_{\text{path}}(A)(Q(D))| \leq |Q_A(D_A)| \leq |D_A|^{\rho^*(Q_A)} \leq |D|^{\rho^*(Q_A)} \).

2. Define \( s(\mathcal{T}) = \max_A \rho^*(Q_A) \).
   - \( s(\mathcal{T}) \) = maximal possible \( \rho^*(Q_A) \) over all attributes \( A \) from \( Q \).
   - Then, the size of the factorisation of \( Q(D) \) over \( \mathcal{T} \) is \( \leq |Q| \cdot |D|^{s(\mathcal{T})} = O(|D|^{s(\mathcal{T})}) \).

3. Define \( s(Q) = \min_{\mathcal{T}} s(\mathcal{T}) \).
   - \( s(Q) \) = minimum possible \( s(\mathcal{T}) \) over all f-trees \( \mathcal{T} \) for \( Q \).
   - Then, there exists an f-representation of \( Q(D) \) with size \( O(|D|^{s(Q)}) \).
Example of computing $s(Q)$

- For each node $X$ except for $F$, we have that $\rho^*(Q_X) = 1$ since all attributes in $\text{path}(X)$ are covered by one relation.

- $\text{path}(F)$ is not covered by one relation and $\rho^*(Q_F) = 2$.

Thus, $s(\mathcal{T}) = 2$.

- $s(Q) = 2$ since $s(\mathcal{T}) = 2$ is the smallest possible value for any f-tree $\mathcal{T}$ of the query $Q$. 

Projections

With $R(A_R, B_R), S(A_S, B_S)$, the query $Q = \sigma_{A_R=A_S} (R \times S)$ has $s(Q) = 1$:

```
A_R, A_S
/   \
|    |
B_R  C_S
```

However the query $Q' = \pi_{B_R, C_S} (\sigma_{A_R=A_S} (R \times S))$ has $s(Q') = 2$.

```
B_R
 |
 C_S
```

After projecting away $A_R, A_S, B_R$ and $C_S$ are dependent and cannot be siblings.

$\Rightarrow$ Projection may increase the factorisation size.
More Succinct Representations: DAG

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

\[
A_R, A_S, A_T \quad B_R, B_S \quad E_T, E_U
\]

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

- 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 \( \langle a \rangle \).
  \[ \Rightarrow \text{store the mapping } \langle e \rangle \mapsto \bigcup_{f \in F} \langle f \rangle \text{ separately.} \]

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