Automatic Layout Optimization for Relational Queries

John K. Feser

ABSTRACT

Selecting views and indices is an old problem in the database literature. In this paper we propose a new approach to the problem of selecting an appropriate physical layout for a query workload. We define a language of physical database layouts which can express many common layout schemes, such as row stores, column stores, and clustered indices. Synthesizing a program in our language of layouts

1. INTRODUCTION

2. MOTIVATING EXAMPLE

3. ALGORITHM

In this section we discuss the layout optimization algorithm.

3.1 Layout Language

Our layout language is a language of nested structures which subsumes standard database layouts such as row and column stores and allows for an efficient implementation. The layout language is designed to work with the relational algebra so that queries can be written over specialized layouts. The layout language can compactly represent the results of executing common relational algebra operations, so query processing can be replaced by data layout. This allows queries to be incrementally rewritten using semantics preserving transformations, as described in Section 3.2.

3.1.1 Syntax

3.1.2 Semantics

A layout can be evaluated, yielding a stream of tuples. Evaluation occurs in a context which binds the

The semantics of the relational algebra are standard. The stream of tuples produced by scanning a layout can be treated in the same way that a scan over a standard database is.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 11
Copyright 2017 VLDB Endowment 2150-8097/17/08.

$$
\begin{aligned}
v,\ k &::= \text{<primitive value>} \\
f &::= \text{<field>} \\
o &::= \text{Asc} \mid \text{Desc} \\
L &::= \emptyset \\
&\mid \text{Value } v \\
&\mid \text{UnorderedList } [L_1,\ \ldots,\ L_n] \\
&\mid \text{OrderedList } o\ f\ [L_1,\ \ldots,\ L_n] \\
&\mid \text{CrossTuple } [L_1,\ \ldots,\ L_n] \\
&\mid \text{ZipTuple } [L_1,\ \ldots,\ L_n] \\
&\mid \text{Table } f\ \{k_1 \mapsto L_1,\ \ldots,\ k_1 \mapsto L_1\} \\
R &::= L \mid \pi_F R \mid \sigma_\phi R \mid \bowtie_\phi R_1 R_2 \mid g_{F,a} R \mid R_1 +\!\!+ R_2
\end{aligned}
$$

Figure 1: Syntax of the layout language.

$$\text{E-Emp}\frac{}{\Gamma \vdash \emptyset \Rightarrow [\,]} \qquad \text{E-Val}\frac{}{\Gamma \vdash \text{Value } v \Rightarrow [(v)]}$$

$$\text{E-UL}\frac{\Gamma \vdash L_1 \Rightarrow s_1, \ldots, \Gamma \vdash L_n \Rightarrow s_n}{\Gamma \vdash \text{UnorderedList } [L_1,\ \ldots,\ L_n] \Rightarrow s_1 +\!\!+ \ldots +\!\!+ s_n}$$

$$\text{E-OL}\frac{\Gamma \vdash L_1 \Rightarrow s_1, \ldots, \Gamma \vdash L_n \Rightarrow s_n}{\Gamma \vdash \text{OrderedList } [L_1,\ \ldots,\ L_n] \Rightarrow s_1 +\!\!+ \ldots +\!\!+ s_n}$$

$$\text{E-CT1}\frac{\Gamma \vdash L \Rightarrow s}{\Gamma \vdash \text{CrossTuple } [L] \Rightarrow s}$$

$$\text{E-CT2}\frac{\Gamma \vdash L \Rightarrow s_l,\ \Gamma \vdash \text{CrossTuple } LS \Rightarrow s_{ls}}{\Gamma \vdash \text{CrossTuple } (L : LS) \Rightarrow s_l \times s_{ls}}$$

$$\text{E-ZT}\frac{\Gamma \vdash L_1 \Rightarrow s_1, \ldots, \Gamma \vdash L_n \Rightarrow s_n \qquad |s_1| = \cdots = |s_n|}{\begin{array}{l}\Gamma \vdash \text{ZipTuple } [L_1,\ \ldots,\ L_n] \Rightarrow \\ [t_1 +\!\!+ \ldots +\!\!+ t_n \mid t_1 \leftarrow s_1, \ldots, t_n \leftarrow s_n]\end{array}}$$

$$\text{E-T}\frac{m[\Gamma[f]] = L \qquad \Gamma \vdash L \Rightarrow s}{\Gamma \vdash \text{Table } f\ m \Rightarrow s}$$

Figure 2: Semantics of the layout language.

$$\sigma_{\phi \wedge \psi}\ q \equiv \sigma_{\psi}\ \sigma_{\phi}\ q \quad \text{(EQ-And1)}$$

$$\sigma_{\phi \wedge \psi}\ q \equiv \sigma_{\phi}\ \sigma_{\psi}\ q \quad \text{(EQ-And2)}$$

$$\sigma_{\phi \vee \psi}\ q \equiv \sigma_{\phi} q +\!\!+ \sigma_{\psi}\ q \quad \text{(EQ-Or)}$$

$$\sigma_{f=x}\ L \equiv \texttt{partition}\ f\ L \quad \text{(EQ-Eq)}$$

$$\sigma_{f \geq x}\ L \equiv \texttt{order}\ f\ L \quad \text{(EQ-Ge)}$$

$$\bowtie_{f_1=f_2}\ L_1\ L_2 \equiv \sigma_{f_1=f_2}(\text{CrossTuple}\ [L_1, L_2]) \quad \text{(EQ-EqJoin)}$$

$$\pi_F\ L \equiv \texttt{project}\ F\ L \quad \text{(EQ-Proj)}$$

$$L_1 +\!\!+ L_2 \equiv \text{UnorderedList}\ [L_1, L_2] \quad \text{(Eq-Concat)}$$

Figure 3: Equivalence relations between queries.

3.2 Equivalence Rules

We define a set of equivalence relations between relational algebra expressions. These relations allow us to transform a query and the layout that it operates on simultaneously. These equivalence rules rely on layout transformation functions. For example, `partition` transforms a layout containing a field f into a Table with f as the key field.

3.3 Synthesis Algorithm

To find the best layout for a query, we use the equivalence rules to find the set of equivalent queries. We measure the cost of each equivalent query and select the query with the lowest cost.

Algorithm 1 Create a set of candidate programs by taking the fixpoint of the equivalence rules.

Require: q is a relational algebra expression. R is a collection of equivalence rules.

```
function SYNTHESIZE(q, R)
    P ← {q}
    loop
        P' = ∅
        for r ∈ R do
            if r matches q then
                P' = P' ∪ {r(q') | q' ∈ P}
        if P = P' then
            return P
        else
            P ← P'
```

3.4 Cost Model

4. EVALUATION

5. RELATED WORK

6. CONCLUSION