

Consider the following Java-JDT plugin name in German:

Consider the following Java-JDT plugin name in German:

BOSTON UNIVERSITY
COLLEGE OF ENGINEERING

Dissertation

A BU THESIS LATEX TEMPLATE

by

JOE CANDIDATE

B.S., Some University, 2010
M.S., Another University, 2012

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2015

© 2015 by
JOE CANDIDATE
All rights reserved

Approved by

First Reader

First M. Last, PhD
Professor of Electrical and Computer Engineering

Second Reader

First M. Last
Associate Professor of ...

Third Reader

First M. Last
Assistant Professor of ...

*Facilis descensus Averni;
Noctes atque dies patet atri janua Ditis;
Sed revocare gradum, superasque evadere ad auras,
Hoc opus, hic labor est.* *Virgil (from Don's thesis!)*

Acknowledgments

Here go all your acknowledgments. You know, your advisor, funding agency, lab mates, etc., and of course your family.

As for me, I would like to thank Jonathan Polimeni for cleaning up old LaTeX style files and templates so that Engineering students would not have to suffer typesetting dissertations in MS Word. Also, I would like to thank IDS/ISS group (ECE) and CV/CNS lab graduates for their contributions and tweaks to this scheme over the years (after many frustrations when preparing their final document for BU library). In particular, I would like to thank Limor Martin who has helped with the transition to PDF-only dissertation format (no more printing hardcopies – hooray !!!)

The stylistic and aesthetic conventions implemented in this LaTeX thesis/dissertation format would not have been possible without the help from Brendan McDermot of Mugar library and Martha Wellman of CAS.

Finally, credit is due to Stephen Gildea for the MIT style file off which this current version is based, and Paolo Gaudiano for porting the MIT style to one compatible with BU requirements.

Janusz Konrad

Professor

ECE Department

A BU THESIS LATEX TEMPLATE

JOE CANDIDATE

Boston University, College of Engineering, 2015

Major Professors: First M. Last, PhD

Professor of Electrical and Computer Engineering

Secondary appointment

First M. Last, PhD

Professor of Computer Science

ABSTRACT

Have you ever wondered why this is called an abstract? Weird thing is that its legal to cite the abstract of a dissertation alone, apart from the rest of the manuscript.

Contents

| | | |
|----------|---|----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Problem at hand | 1 |
| 1.3 | Structure of thesis | 1 |
| 1.4 | Conclusion | 1 |
| 2 | Related Work | 2 |
| 2.1 | Introduction, Query optimization | 2 |
| 2.2 | Converting SQL queries to parse trees | 2 |
| 2.3 | Relational algebra | 3 |
| 2.3.1 | Select operator σ | 4 |
| 2.3.2 | Projection operator π | 5 |
| 2.3.3 | Duplicate Elimination operator δ | 5 |
| 2.3.4 | Aggregation operator γ | 6 |
| 2.4 | Converting Parse trees into logical expression | 6 |
| 2.5 | Explain difficulties/ Time complexity | 7 |
| 2.6 | Optimization using relation algebra | 7 |
| 2.7 | Introduction to Data Streams | 7 |
| 2.8 | Data stream windowing | 7 |
| 2.9 | Query Processing of data streams(Combine the DBMS and DSMS) . . | 7 |
| 2.10 | Challenges of query optimization on data streams | 7 |
| 2.11 | Conclusion and discussion | 7 |

| | | |
|----------|---|-----------|
| 3 | <i>Stream Optimization</i> | 8 |
| 3.1 | <i>Query Optimization of Data Streams</i> | 8 |
| 4 | <i>Implementation</i> | 9 |
| 4.1 | <i>Query Optimization of Data Streams</i> | 9 |
| 5 | <i>Stream Optimization</i> | 10 |
| 5.1 | <i>Query Optimization of Data Streams</i> | 10 |
| 6 | <i>Stream Optimization</i> | 11 |
| 6.1 | <i>Query Optimization of Data Streams</i> | 11 |
| A | <i>Proof of xyz</i> | 12 |
| | <i>Curriculum Vitae</i> | 13 |

List of Tables

List of Figures

List of Abbreviations

The list below must be in alphabetical order as per BU library instructions or it will be returned to you for re-ordering.

| | | |
|----------------|-------|---|
| <i>CAD</i> | | <i>Computer-Aided Design</i> |
| <i>CO</i> | | <i>Cytochrome Oxidase</i> |
| <i>DOG</i> | | <i>Difference Of Gaussian (distributions)</i> |
| <i>FWHM</i> | | <i>Full-Width at Half Maximum</i> |
| <i>LGN</i> | | <i>Lateral Geniculate Nucleus</i> |
| <i>ODC</i> | | <i>Ocular Dominance Column</i> |
| <i>PDF</i> | | <i>Probability Distribution Function</i> |
| \mathbb{R}^2 | | <i>the Real plane</i> |

Chapter 1

Introduction

1.1 Motivation

1.2 Problem at hand

Hello

1.3 Structure of thesis

Works

1.4 Conclusion

The next chapter gives an in-depth view of the pipeline used by the current state of art technology for query optimization in traditional data bases including the mathematical knowledge for simplification and the overall framework. The next chapter also introduces the reader to data stream and how data bases are used for them called DSMS and showcases an approach to optimize queries on data streams for the problem discussed above. The following chapter list out the details of implementation, challenges face, evaluation methods used, benchmark test case timings, followed by a summary of the paper.

Chapter 2

Related Work

2.1 Introduction, Query optimization

A database can be thought of as a list of tables, where in each table itself can be considered as a list of data points ordered initially in the sequence they are entered.

There are various tools which can be used to connect to a database, here we focus on structured query languages(SQL). A simple SQL query looks like this

```
1  SELECT column_name_1 , column_name_2  
2  FROM table_name  
3  WHERE condition
```

This query is essentially asking to display the 2 columns from the table where the condition given is satisfied. This to particular query might be looking simple, but if the condition introduced is a complex one or if the table from which we need to return the output is complex, the question of how to execute the query optimally becomes difficult to answer.

2.2 Converting SQL queries to parse trees

This step has several functions.

If a "view" is used in the query as a relation, then each instance has to be replaced by the parse tree.

The preprocessor also has to conduct semantic checking, that is, check if relations used exist, check for ambiguity, and type checking. If a parse tree passes the pre-

processing then it is said to be **valid**. We don't describe the exact grammar for the conversion to the parse tree. In these parse trees, there are 2 types of nodes, one the atoms, which are essentially keywords in SQL, operators, constants and attributes. The second is Syntactic categories, these are names for families of subqueries in triangular brackets. Each of the syntactic category has unique expansion into atoms and further syntactic categories.

2.3 Relational algebra

As we saw above, order of operations matters, if the order of operations is not thoughtout and done blindly alot of redundant steps are executed and memory is moved around unnecessarily. There are few ways to atleast look and analyse the operations and how they can be simplified.

Let R, S be relations. Some simple laws, associativity and commutativity can easily be verified:-

- $R \times S = S \times R$
- $(R \times S) \times T = R \times (S \times T)$
- $R \bowtie S = S \bowtie R$
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $R \cup S = S \cup R$
- $(R \cup S) \cup T = R \cup (S \cup T)$
- $R \cap S = S \cap R$
- $(R \cap S) \cap T = R \cap (S \cap T)$

When applying associative law on relations, need to be careful whether the conditions actually makes sense after the order is changed.

While the above identities work on both sets and bags(bags allow for repeatition). To show that laws for sets and bags do differ an easy way is to consider the distributive property.

$$A \cap_S (B \cup_S C) = (A \cap_S B) \cup_S (A \cap_S C)$$

$$A \cap_B (B \cup_B C) \neq (A \cap_B B) \cup_B (A \cap_B C)$$

We can simply show it with an example. Let $A = \{t\}, B = \{t\}, C = \{t\}$. The LHS comes to be $\{t\}$, whereas RHS is $\{t, t\}$

2.3.1 Select operator σ

First we start with simple properties of the σ operator. Need to be careful about the attributes used in the select operator condition when pushing it down.

- $\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$
- $\sigma_{C_1 \vee C_2}(R) = (\sigma_{C_1}(R)) \cup_S (\sigma_{C_2}(R))$
- $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
- $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S) = \sigma_C(R) - S$
- $\sigma_C(R \times S) = \sigma_C(R) \times S$
- $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$
- $\sigma_C(R \bowtie_D S) = \sigma_C(R) \bowtie_D S$
- $\sigma_C(R \cap S) = \sigma_C(R) \cap S$

2.3.2 Projection operator π

While for the Select operator(σ) the identities were quite straight forward with not many things to consider, the identities for Projection operator (π) are bit more involved.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$, where M, N are attributes required for the join or they are inputs to the projection.
- $\pi_L(R \bowtie_D S) = \pi_L(\pi_M(R) \bowtie_D \pi_N(S))$, similar to above identity/ law.
- $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$
- $\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$
- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$

2.3.3 Duplicate Elimination operator δ

The δ operator eliminates duplicates from bags.

- $\delta(R) = R$, if R does not have any duplicates.
- $\delta(R \times S) = \delta(R) \times \delta(S)$
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(R \bowtie_D S) = \delta(R) \bowtie_D \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(R \cap_B S) = \delta(R) \cap_B S$

2.3.4 Aggregation operator γ

It is difficult to give identities for the aggregation operator, like done for the above operators. This is mostly due to how the details of how the aggregation operator is used.

- $\sigma(\gamma_L(R)) = \gamma_L(R)$
- $\gamma_L(R) = \gamma_L(\pi_M(R))$, where M must at least contain the attributed used in L .

2.4 Converting Parse trees into logical expression

Till now, the only SQL related information presented is how to convert a Query into the parse tree, which is grammar dependent. Given the parse tree, need to substitute nodes by operators seen above, later this expression is optimized to be later converted to a physical query plan.

Now to convert the parse tree into the logical expression. First, look at the transformation of select-from-where statement.

- $\langle \text{Query} \rangle \rightarrow \langle \text{SFW} \rangle$.
- $\langle \text{SFW} \rangle \rightarrow \text{SELECT } \langle \text{SelList} \rangle \text{ FROM } \langle \text{FromList} \rangle \text{ WHERE } \langle \text{Condition} \rangle$.
- $\langle \text{SelList} \rangle \rightarrow \pi_L$, where L is the list of attributes in $\langle \text{SelList} \rangle$.
- $\langle \text{Condition} \rangle \rightarrow \sigma_C$, where C is the equivalent of $\langle \text{Condition} \rangle$

While there is nothign inherently wrong about the last statement, need to consider the case where $\langle \text{Condition} \rangle$ involves subqueries. A simple explanation about why it isn't allowed is a convention normally the subscript has to be a boolean condition, and if it was allowed otherwise, it would be a very expensive operation, as the subscript

in the select operator has to be evaluated at every element of the argument relation. This shows the redundancy of it. While if this is allowed, it can be simplified and made efficient, but has to be done on a case by case basis with the use of \bowtie, \times functions.

Overall it is a good idea to not use subquerying and rather using joins.

At this point, by making the substitutions mentioned above and using the algebraic identities, we obtain a starting logical query plan. The query has to be transformed into a query which the compiler believes to be the cheapest or the optimal. But, a thing which further complicates the process is the join order.

With the current knowledge, few optimizing rules are evident.

- **Selection repositioning**, Selections should be pushed down as much as possible, but sometimes, might need to take the selection operator a level up first.
- Pushing projections down the parse tree, being careful with the new projections made in the process.
- Duplicate removal needs to be repositioned.
- σ combined with \times below can result in equijoin, which is much more efficient.

While

- 2.5 Explain difficulties/ Time complexity
- 2.6 Optimzation using relation algebra
- 2.7 Introduction to Data Streams
- 2.8 Data stream windowing
- 2.9 Query Processing of data streams(Combine the DBMS and DSMS)
- 2.10 Challenges of query optimization on data streams
- 2.11 Conclusion and discussion

Chapter 3

Stream Optimization

3.1 Query Optimization of Data Streams

Chapter 4

Implementation

4.1 Query Optimization of Data Streams

Chapter 5

Stream Optimization

5.1 Query Optimization of Data Streams

Chapter 6

Stream Optimization

6.1 Query Optimization of Data Streams

Appendix A

Proof of xyz

This is the appendix.

CURRICULUM VITAE

Joe Graduate

Basically, this needs to be worked out by each individual, however the same format, margins, typeface, and type size must be used as in the rest of the dissertation.