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

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

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.

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

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 discusses above. The following chapter list out the details if implementation, challenges face, evaulation 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

```
SELECT column_name_1, column_name_2
FROM table_name
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 semaContainment of Value Sets. If Y is an attribute appearing in several rela tions, then each relation chooses its values from

the front of a fixed list of values y1 , y2 , y3 , and has all the values in that prefix. As a consequence, if R and S are two relations with an attributContainment of Value Sets. If Y is an attribute appearing in several rela tions, then each relation chooses its values from the front of a fixed list of values y1 , y2 , y3 , and has all the values in that prefix. As a consequence, if R and S are two relations with an attribute Y, and V(R, Y) ::; V(S, Y), then every Y-value o f R will be a Y-value of S.e Y, and V(R, Y) ::; V(S, Y), then every Y-value o f R will be a Y-value of S.ntic checking, that is, check if relations used exist, check for ambiguity, and type checking. If a parse tree passes the preprocessing then it is said to be ***valid*** *We don't describe the exact grammar for the convertion 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 simplied.

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.*

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

- $\sigma_C(R \times S) = \sigma_C(R) \times S$
- $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$
- $\sigma_C(R \underset{D}{\bowtie} S) = \sigma_C(R) \underset{D}{\bowtie} S$
- $\sigma_C(R \cap S) = \sigma_C(R) \cap S$

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.*
- $\pi_L(R \underset{D}{\bowtie} S) = \pi_L(\pi_M(R) \underset{D}{\bowtie} \pi_N(S))$, *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.*
- $\delta(R \times S) = \delta(R) \times \delta(S)$
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(R \underset{D}{\bowtie} S) = \delta(R) \underset{D}{\bowtie} \delta(S)$

- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(R \cap_B S) = \delta(R) \cap_B S$

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.

- $< Query > \rightarrow < SFW >$.
- $< SFW > \rightarrow SELECT < SelList > FROM < FromList > WHERE < Condition >$.
- $< SelList > \rightarrow \pi_L$, *where* L *is the list of attributes in* $< SelList >$.
- $< Condition > \rightarrow \sigma_C$, *where* C *is the equivalent of* $< Condition >$

While there is nothign inherently wrong about the last statement, need to consider the case where $< Condition >$ *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 evaluted at every element of the argument relation. This shows the redundance 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 possbile, 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.*

Normally, parsers only have nodes with $0, 1, 2$ *children, which corresponds to uniary and binary operators. But many of the operators(natural join, union, and intersection) are commutative and associative, so it helps combine them on a single level to provide the opportunity to prioritize which ones are done first. To do this step, the following guidelines are sufficient.*

- *We must replace the natural joins with theta-joins that equate the at tributes of the same name.*

- *We must add a projection to eliminate duplicate copies of attributes involved in a natural join that has become a theta-join*
- *The theta-join conditions must be associative.*

2.5 Explain difficulties/ Time complexity

Currently, we have taken a query as an input and converted it into a logical plan, and then applied more transformation using relational algebra to make the optimal query plan. The next step is to convert it into a physical query plan which can be executed. To complete this step, need to compare the cost of various physical query plan derived from the logical query plan. This plan with the least cost is then passed query execution engine and executed. When trying to select a query plan, need to select:-

- *An order and grouping for associative-and-commutative operations like joins, unions, and intersections.*
- *An algorithm for each operator in the logical plan, for instance, deciding whether a nested-loop join or a hash-join should be used.*
- *Additional operators - scanning, sorting, and so on that are needed for the physical plan but that were not present explicitly in the logical plan.*
- *The way in which arguments are passed from one operator to the next, for instance, by storing the intermediate result on disk or by using iterators and passing an argument one tuple or one main-memory buffer at a time.*

But after selecting these for a physical plan, one obivious way to calculate the time is actually executing the plan to see the cost, but that way essentially every plan is carried out. That is expensive and redundant. Is there a better way?

2.5.1 Estimating size and cost

$B(R) \coloneqq$ *is the number of blocks needed to hold all the tuples of relation* R.

$T(R) \coloneqq$ *is the number of tuples of relation* R.

$V(R, a) \coloneqq$ *is the value count for attribute* a *of relation* R*, that is, the number of distinct values relation* R *has in attribute* a.

$V(R, [a_1, a_2, ..., a_n]) \coloneqq$ *is the number of distinct values* R *has when all of attributes* $a_1, a_2, ..., a_n$ *are considered together, that is, the number of tuples in* $\delta(\pi_{a_1,a_2,...,a_n}(R))$

Need to remember that physical plan is selected to minimize the estimated cost of evaluating the query and that intermidate/ temporary relations calculated will also incur a cost on the system. So need to take them into account as well. Over all the estimation method should pass the following sanity check:-

- *Give accurate estimates. No matter what method is used for executing query plans.*
- *Are easy to compute.*
- *Are logically consistent; that is, the size estimate for an intermediate relation should not depend on how that relation is computed. For instance, the size estimate for a join of several relations should not depend on the order in which we join the relations.*

But there is no agreed upon method to do this, but the goal is to help select a query plan and not to find the exact minimum. So even if the estimated cost is wrong, but if it is wrong similarly then we will still have the least costing plan.

2.5.2 Estimation of Projection

The projection operator is a bag operation, so it does not reduce the number of tuples, only reduces the size of each tuple.

$T(R) = T(\pi(R))$

$B(R) \leq B(\pi(R))$

2.5.3 Estimation of Selection

Let $S = \sigma_{A=c}(R)$, here A is an attribute of R and c is a constant

$T(S) = \frac{T(R)}{V(R,A)}$*, is it important to note that this is an estimate and not the actual value, this will be the actual value if all the attributes in A have equal occurance. An even better estimate can be obtained if the DBMS stores a statistic know as histogram.*

The above calculate was easy because of the equality. if $S = \sigma_{A<c}(R)$, we simply estimate $T(S) = \frac{T(R)}{3}$

Now if $S = \sigma_{A\neq c}(R)$, can take $T(S) = T(R)$ or $T(S) = T(R) - \frac{T(R)}{V(R,A)}$

If $S = \sigma_{A=c_1 \vee c_2}$, take them to be independent conditions.

2.5.4 Estimation of Join, single attribute

For natural joins, we assume, the natural join of two relations involves only the equality of two attributes. That is, we study the join $R(X,Y) \bowtie S(Y,Z)$, but initially we assume that Y is a single attribute although X and Z can represent any set of attributes. But it is hard to find a good estimate as $T(R \bowtie S) \in [0, T(R) * T(S)]$, to help with this two assumptions are made.

- ***Containment of Value Sets*** *If Y is an attribute appearing in several relations, then each relation chooses its values from the front of a fixed list of values $y_1, y_2, y_3, ...$ and has all the values in that prefix. As a consequence, if R and S are two relations with an attribute Y, and $V(R,Y) \leq V(S,Y)$, then every Y-value of R will be a Y-value of S.*

- ***Preservation of Value Sets*** *If we join a relation R with another relation, then an attribute A that is not a join attribute (i.e., not present in both relations)*

does not lose values from its set of possible values. More precisely, if A is an attribute of R but not of S, then $V(R \bowtie S, A) = V(R, A)$. Note that the order of joining R and S is not important, so we could just as well have said that $V(S \bowtie R, A) = V(R, A)$.

Using the above assumptions we can claim,

$$T(R \bowtie S) = \frac{T(R) * T(S)}{max(V(R, Y), V(S, Y))}$$

Let $V(R, Y) \leq V(S, Y)$, then every tuple t of R has a chance of $\frac{1}{V(S,Y)}$ of joining with a given tuple of S. Since there are $T(S)$ tuples in S, the expected number of tuples that t joins with is $\frac{T(S)}{V(S,Y)}$. As there are $T(R)$ tuples of R, the estimated size of $R \bowtie S$ is $\frac{T(R)*T(S)}{V(S,Y)}$, if it was $V(R, Y) \geq V(S, Y)$, then $\frac{T(R)*T(S)}{V(R,Y)}$

Guidelines for other type of joins:-

- *The number of tuples in the result of an equijoin can be computed exactly as for a natural join, after accounting for the change in variable names.*
- *Other theta-joins can be estimated as if they were a selection following a product.*

2.6 Introduction to Data Streams

2.7 Data stream windowing

2.8 Query Processing of data streams(Combine the DBMS and DSMS)

2.9 Challenges of query optimization on data streams

2.10 Conclusion and discussion

Chapter 3

Stream Optimization

3.1 Query Optimization of Data Streams

Chapter 4

Implementation

4.1 Mathematics

Chapter 5

Evaluation

5.1 Measures used

Chapter 6

Conclusion and Further work

6.1 Conclusion

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.