School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Exploring Just-in-Time Compilation in Relational Database Engines

by

Nicolaas Johannes van der Merwe

Thesis submitted as a requirement for the degree of

Bachelor of Computer Science (Honours)

Submitted: November 2024

Supervisor: Dr Zhengyi Yang

Student ID: z5467476

Abstract

Abbreviations

ACID Atomicity, consistency, isolation, durability

CPU Central Processing Unit

DB Database

EXP Expression (expressions inside queries)

IR Intermediate Representation

JIT Just-in-time (compiler)

JVM Java Virtual Machine

LLC Last Level Cache

MLIR Multi-Level Intermediate Representation

QEP Query Execution Plan

RA Relational Algebra

SQL Structured Query Language

SSD Solid State Drive

TPC-H Transaction Processing Performance Council

Contents

List of Figures

Exploring Just-in-Time Compilation in Relational Database Engines	Nicolaas Johannes van der Merwe

Chapter 1

Introduction

1.1 Example Citation and Figure

This is an example of how to make a citation. You can cite papers like this [SJ20]. You can also cite multiple sources [SJ20, AW21].

1.1.1 Including Figures

Here is an example of how to include a figure in your report:

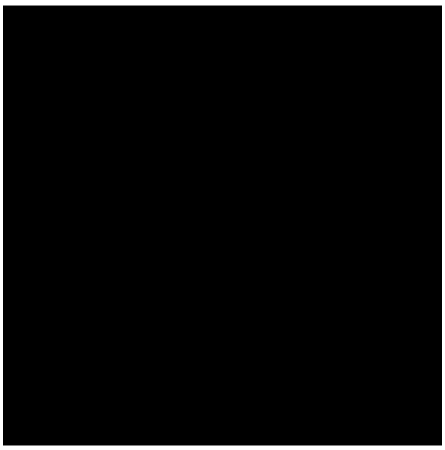

Figure 1.1: Example black figure

You can reference the figure in text like this: See Figure 1.1.

Chapter 2

Background

2.1 Overview

This is a condensed version of the literature review that was conducted earlier in this project. It will briefly walk through a typical database archicture (volcano model) and JIT compilers 2.2, the existing explored solutions and it's history, **??**, and finally finish with a summary of the original goals.

2.2 Fundamentals

Majority of databases are structured like Figure 2.1. Strucutre Query Language (SQL) is parsed, turned to RA (relational-algebra), optimized, executed, then materialized into a table.

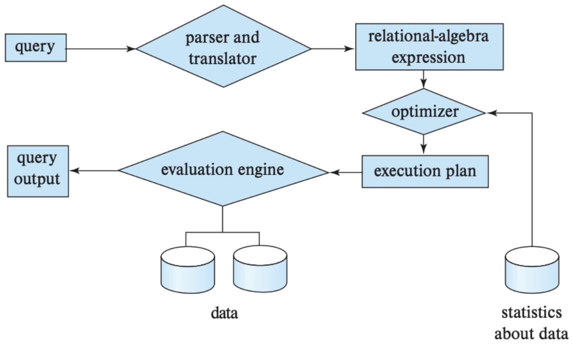

Figure 2.1: Database Structure
[SKS19]

For non-compiler databases they use a volcano operator model tree, such as Figure 2.2. The root node has a `produce()` function which calls its children's `produce()`, until it calls a leave node, which calls `consume()` on itself, then that calls its parent's `consume()` function. In other words, a post-order traversal through this tree where tuples are dragged upwards.

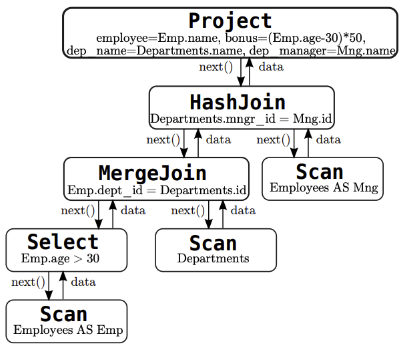

Figure 2.2: Volcano operator model tree.
[ZVBB11]

The fundamental issue with this classical model is that it is heavily underutilising the hardware. If we are only pulling up a single tuple, our CPU caches are barely used. This has lead to the vectorized execution model and the compiled model. With the vectorized model, we pull up groups of tuples instead. However, this leads to problems where it's common to have too many copy operations instead of having a pointer going upwards. For instance, if a sort or a join allocates new space that is too much for the cache, the handling can become poor. With the compiled approach, it introduces a lot of implementation complexity.

Relational databases prioritise ACID requiremens - Atomicity, Consistency, Isolation and Durability [SKS19]. This is a critical requirement in this type of system, and usually one of the main reasons people pick a relational database. Atomicity refers to transactions are a single unit of work, consistency means it must be in a valid state before and after the query, isolation means concurrent transactions do not interact with each other, and durability means once something is committed it will stay committed. [SKS19]

It is common for on-disk databases to consider the cost of CPU operations to be $O(1)$ [SKS19]. This is partially due to when these systems were made, the disks were much slower and the caches were much slower. In part A of this project, this was disproved for PostgreSQL as it was found that the time spent in the CPU was substantial: between 34.87% and 76.56% with an average of 49.32% across the tested queries.

Just-in-time (JIT) compilers work by having multiple layers of compilation and are mostly used by interpreted languages to eliminate the ill-effects on performance [ZVBB11]. Advanced compilers can run the primary program, then dedicate some background threads to improving the optimisation of the code, and swap it over to the optimized version when it is ready [KLN18].

Due to branch-prediction optimization, JIT compilers can become faster than ahead of time compilers. In 1999, a benchmarking paper measured four commercial databases and found 10%−20% of their execution time was spent fixing poor predictions [ADHW99]. similarly, research specifically into branch prediction has said, "although branch prediction attained 99% accuracy in predicting static branches, ... branches are still a major bottleneck in the system performance" [Jos21]. Modern measurements still find 50% of their query times are spent resolving hardware hazards, such as mispredictions, with improvements in this area making their queries 2.6x faster [Ker21]. The Azul JIT compiler measured that their JIT solution's speculative optimizations can lead up to 50% performance gains [Azu22].

In the context of databases, most compilers can be split into only compiling expressions (typically called EXP for expression), and others that compile the entire Query Execution Plan (QEP) [MYH+24]. Within PostgreSQL itself, they have EXP support using $llvm - jit$, but in this paper, QEP will be explored as well.

The LLVM Project is a compiler infrastructure that supports making compilers so that common, but complex, compiler optimisations do not have to be re-implemented. Multi-Level Intermediate Representation is another, newer toolkit that is tightly coupled with the LLVM project. It adds a framework to define dialects, and lower through these dialects. One of the primary benefits of this is if you make a compiler, you can define a high level dialect, then another person can target your custom high-level dialect.

2.3 Literature Review

2.4 Summary of Goals

Chapter 3

Project

Lorem Ipsum

Conclusion

Lorem Ipsum

Acknowledgements

This work has been inspired by the labours of numerous academics in the Faculty of Engineering at UNSW who have endeavoured, over the years, to encourage students to present beautiful concepts using beautiful typography.

Further inspiration has come from Donald Knuth who designed TEX, for typesetting technical (and non-technical) material with elegance and clarity; and from Leslie Lamport who contributed LATEX, which makes TEX usable by mortal engineers.

John Zaitseff, an honours student in CSE at the time, created the first version of the UNSW Thesis LATEX class and the author of the current version is indebted to his work.

Lastly my supervisor and assessor for supporting me throughout this project.

Bibliography

[ADHW99] Anastassia Ailamaki, David J DeWitt, Mark D Hill, and David A Wood. Dbmss on a modern processor: Where does time go? In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 266–277, 1999.

[AW21] Jane Author and Bob Writer. Another example paper. In *Proceedings of Example Conference*, pages 456–470. Example Organization, 2021.

[Azu22] Azul. Jit performance: Ahead-of-time versus just-in-time. `https://www.azul.com/blog/jit-performance-ahead-of-time-versus-just-in-time/`, 2022. Accessed: 2025-11-08.

[Jos21] Rinu Joseph. A survey of deep learning techniques for dynamic branch prediction. *arXiv preprint arXiv:2112.14911*, 2021.

[Ker21] Timo Kersten. *Optimizing Relational Query Engine Architecture for Modern Hardware*. PhD thesis, Technische Universität München, 2021.

[KLN18] André Kohn, Viktor Leis, and Thomas Neumann. Adaptive execution of compiled queries. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 197–208, 2018.

[MYH+24] Miao Ma, Zhengyi Yang, Kongzhang Hao, Liuyi Chen, Chunling Wang, and Yi Jin. An empirical analysis of just-in-time compilation in modern databases. In Zhifeng Bao, Renata Borovica-Gajic, Ruihong Qiu, Farhana Choudhury, and Zhengyi Yang, editors, *Databases Theory and Applications*, pages 227–240, Cham, 2024. Springer Nature Switzerland.

[SJ20] John Smith and Jane Johnson. Example article title. *Example Journal*, 10(5):123–145, 2020.

[SKS19] Abraham Silberschatz, Henry Korth, and S Sudarshan. *Database System Concepts*. McGraw-Hill, New York, NY, 7 edition, 2019.

[ZVBB11] Marcin Zukowski, BV VectorWise, Peter Boncz, and Henri Bal. Just-in-time compilation in vectorized query execution. 2011.