Index Recommendation In Quantum

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Technology
IN
Faculty of Engineering

BY

Mohit Jha

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2025

Declaration of Originality

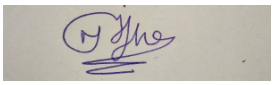
I, **Mohit Jha**, with SR No. **04-04-00-10-51-23-1-22642** hereby declare that the material presented in the thesis titled

Index Recommendation In Quantum

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2023-2025**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: 25/06/2025



Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: PROF. JAYANT HARITSA

Advisor Signature

DEDICATED TO

The Student Community

who can use and reuse this template to glory

Acknowledgements

I would like to express my heartfelt gratitude to my advisor, Prof. Jayant R. Haritsa, for introducing me to the core concepts of Database Engines and also for showing me how actual research needs to be done. I feel blessed to have worked under him for my M.Tech project. Though this was only a 2-year association, the amount of knowledge I have gained is invaluable. He has a very interesting thought process, very different and innovative. Also, his ability to point out shortcomings in me has always been accurate and constructive to help me become a better researcher. While working with him, I felt the true essence of what it means to be a student at IISc.

I am also deeply grateful to Manish Kesarwani (IBM) for guiding me throughout the project. He was very humble in understanding all my doubts and showed me the right direction every time. We had many technical as well as non-technical discussions, and his focus on improving the current state and not dwelling much on the past helped me make better progress in my project.

I would also like to thank my labmates at the Database System Labs at IISc, the DSLites, for having a very lively and supportive environment. I would greatly miss working with them.

Lastly, but most importantly, I am deeply thankful to my family, who have been my greatest support and source of inspiration. There have been times when I have struggled a lot, but they have always supported me and believed in me no matter what happened. The belief they had gave me the confidence to never give up.

Abstract

Indexes are useful in reducing the cost of workload comprising SQL queries. Database Administrators used to do this work by identifying certain query patterns and recommending indexes that minimize the overall workload cost. However, this is not feasible for workloads comprising complex SQL queries. Thus, several index advisor tools were proposed that automated the process of finding the set of indexes to minimize the cost of the workload. But these index advisor tools often generate suboptimal index configurations, as, due to the inherent computational hardness of the problem, they resort to various heuristics.

A hybrid Quantum-Classical Index Advisor [4] has previously demonstrated the potential to address this issue. However, the quality of the solution is limited by the initial set of candidate indexes generated by the database engine, which itself can be suboptimal due to various heuristics used in computing them by the database engine.

In order to improve the quality of the solution, we present a Quadratic Unconstrained Binary Optimization (QUBO) formulation that encodes the index advisor problem without relying on inherent heuristics used by the database engine. We use a Quantum Approximate Optimization Algorithm (QAOA)-based approach to solve the formulation. Our proposed solution is independent of the initial set of indexes generated by the database engine, thereby reducing the number of heuristics we rely on for generating index recommendations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Indexes are crucial for achieving high database workload performance [11]. Traditionally, indexes are identified, built, and maintained by the Database Administrator (DBA). However, it is not feasible for DBAs to identify good index configurations for complex query workloads. To automate this process, several index advisors have been proposed to automatically and judiciously find an appropriate set of indexes to optimize query performance. However, the Index Selection problem is NP-Complete [6] and thus these index advisors use several heuristics to solve the index selection problem. Due to these heuristics, the recommended indexes from Index Advisor tools can be highly suboptimal. For instance, DB2 IA reduces the index selection problem to an instance of the 0-1 Knapsack problem and then invokes a greedy heuristic-based solver to recommend the index configuration.

Consider an SQL workload comprising TPC-H Queries over a 1 GB TPC-H database:

```
SELECT *
FROM LINEITEM
WHERE L_QUANTITY > 49
  AND L_DISCOUNT > 0.099;

SELECT SUM(L_QUANTITY)
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-12-31'
  AND L_SHIPDATE < '1995-01-01';
```

We constructed all possible index configurations having a maximum of 2 columns for the above workload. The cost of workload when no indexes are there, the cost of workload under DB2 recommended indexes, and the cost of workload under optimal index configuration (having

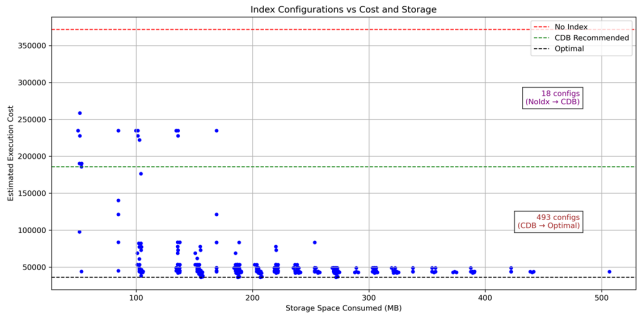


Figure 1.1: Costing of different configurations in DB2

a maximum of 2 columns) are plotted in 1.1.

This graph shows how heuristics impact the quality of index recommendations. From the graph, we can see there are only 18 configurations whose workload costs lie between the No Index case and CDB recommended case, whereas there are 493 configurations that lie between CDB recommended and Optimal. Many of these configurations have half the estimated workload cost than what was recommended by IBM's DB2 Index Advisor.

1.1 Problem Framework

Statement. We are given a workload comprising SQL Queries. The goal is to find an index configuration that minimizes the estimated cost of workload execution within the storage constraint specified by the user.

Assumption. Only single-column indexes are permitted over the workload, and the workload comprises of read-only queries; update statements are not permitted.

Output. Single-column indexes that are within the budget specified by the user.

1.2 Computational Hardness of the problem

The reason why we are only working with single-column indexes is that there is very limited number of qubits available. But even while working in the domain of only single-column indexes, the problem is still exponential. A column is indexable if it appears as a SARGABLE predicate. If n such indexable columns are present, then there are $2^n - 1$ configurations possible (except the empty configuration). In the future work, we will try to extend this formulation for multi-column indexes when more number of qubits are available.

1.3 Costing Configurations

In order to find the index configuration that gives minimal cost, we first need a way to cost the index configurations. Query optimizers compute the estimated cost of the queries using several metrics such as: access methods (table access or index access), selectivity of clauses, etc., without actually executing the query. We will use the help of the query optimizer to cost the configurations. Since there are exponential such index configurations, it is not possible to cost each of these with the help of an optimizer, as optimizer calls are expensive.

To reduce the number of optimizer calls, we use the notion of atomic configurations [2]. A configuration C is atomic for a query Q if there exists an optimizer-chosen plan that uses all the indexes in C.

The cost of any non-atomic configuration can be derived from atomic configurations as below:

$Cost(Q, C) = min\{Cost(Q, C_i)\}$; C_i are atomic configurations that are subsets of C

Here, we are assuming empty configuration is also one of the atomic configurations.

Let's take an example to better under understand atomic configurations. Consider the below query:

```
SELECT SUM(L_EXTENDEDPRICE),L_ORDERKEY,L_SUPPKEY,L_TAX
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-12-31' AND L_SHIPDATE < '1995-01-01'
GROUP BY L_ORDERKEY,L_SUPPKEY,L_TAX;

SELECT L_SHIPDATE,SUM(L_DISCOUNT),AVG(L_EXTENDEDPRICE)
FROM LINEITEM
WHERE L_QUANTITY > 49 AND L_TAX > 0.01 AND
      L_RECEIPTDATE >= '1990-12-31' AND L_RECEIPTDATE < '1991-01-01'
GROUP BY L_SHIPDATE;
```

From IBM's DB2 engine, the following atomic configurations were computed for both queries:

Atomic Config of Query 0:
(('L_SHIPDATE'), 3616.61)
Atomic config of query: 1
(('L_RECEIPTDATE',), 20.32)
(('L_QUANTITY',), 41655.71)

Suppose we want to compute the cost of non-atomic configuration C: `['L_SHIPDATE', 'L_RECEIPTDATE']`. For query 0, ['L_SHIPDATE'] is an atomic configuration subset of C and

has cost 3615.61 and for query 1, ['L_RECEIPTDATE'] and ['L_QUANTITY'] are atomic configurations subset of C, but atomic configuration ['L_RECEIPTDATE'] has lesser cost so this will be selected. The overall cost of the workload will be 3636.93 (by adding up the atomic config cost of both queries), and thus, in this way, we can derive the cost of non-atomic configurations without any optimizer calls.

1.4 Evaluating Performance

We evaluated 10 workloads on IBM's DB2 Index Advisor [8] and 6 workloads on Microsoft's Autoadmin [2]. The details about each workload is present in the appendix section. For each workload, we first find out the atomic configurations of each query. Using the brute-force method we find out the optimal set of single-column indexes under given atomic configurations over the workload (we call these as AC_1). Next we also compute AC_2, which is the optimal set of single and two-column indexes under the atomic configurations over the workload. The reason for computing AC_2 is to evaluate further reduction in estimated workload cost when allowing 2-column indexes compared to only single-column indexes in AC_1.

We also evaluated the performance of QIA (Quantum Index Advisor), which was proposed recently in 2024 [4]. QIA is a hybrid quantum-classical approach that extracts the initial set of indexes from the optimizer and then maps this initial set to a knapsack problem instance. Then it uses a Search-based QIA approach to recommend index configuration.

Plots in fig 1.2 and fig 1.3 show how estimated workload cost varies under the following configurations: 1) No index 2) Commercial Index Advisor Recommended (CDB) Indexes 3) QIA Recommended Indexes [4] 4) Optimal set of indexes from atomic config (1-col) 5) Optimal set of indexes from atomic config (2-col). For the SQL Server case, QIA Recommended Indexes are not shown because SQL Server doesn't provide a way to show what the initial set of candidate indexes it considers while recommending indexes for the workload. Also, in some plots, AC_2 is not reported because it wasn't feasible to continue the brute force method on those problem instances. The experiments are carried out under a 1 GB TPC-H database [9]. The capacity constraint for each workload is also mentioned alongside the workload name.

Results in Fig. 1.2 and Fig. 1.3 show a significant gap in the quality of index recommendations by DB2 as well as Autoadmin. In all the cases, AC_1 performance is better than CDB as well as QIA, even though AC_1 only had single-column indexes. Also, in most cases, the difference between AC_1 and AC_2 is not much, which further motivates us to solve the index advisor problem in the domain of only single-column indexes.

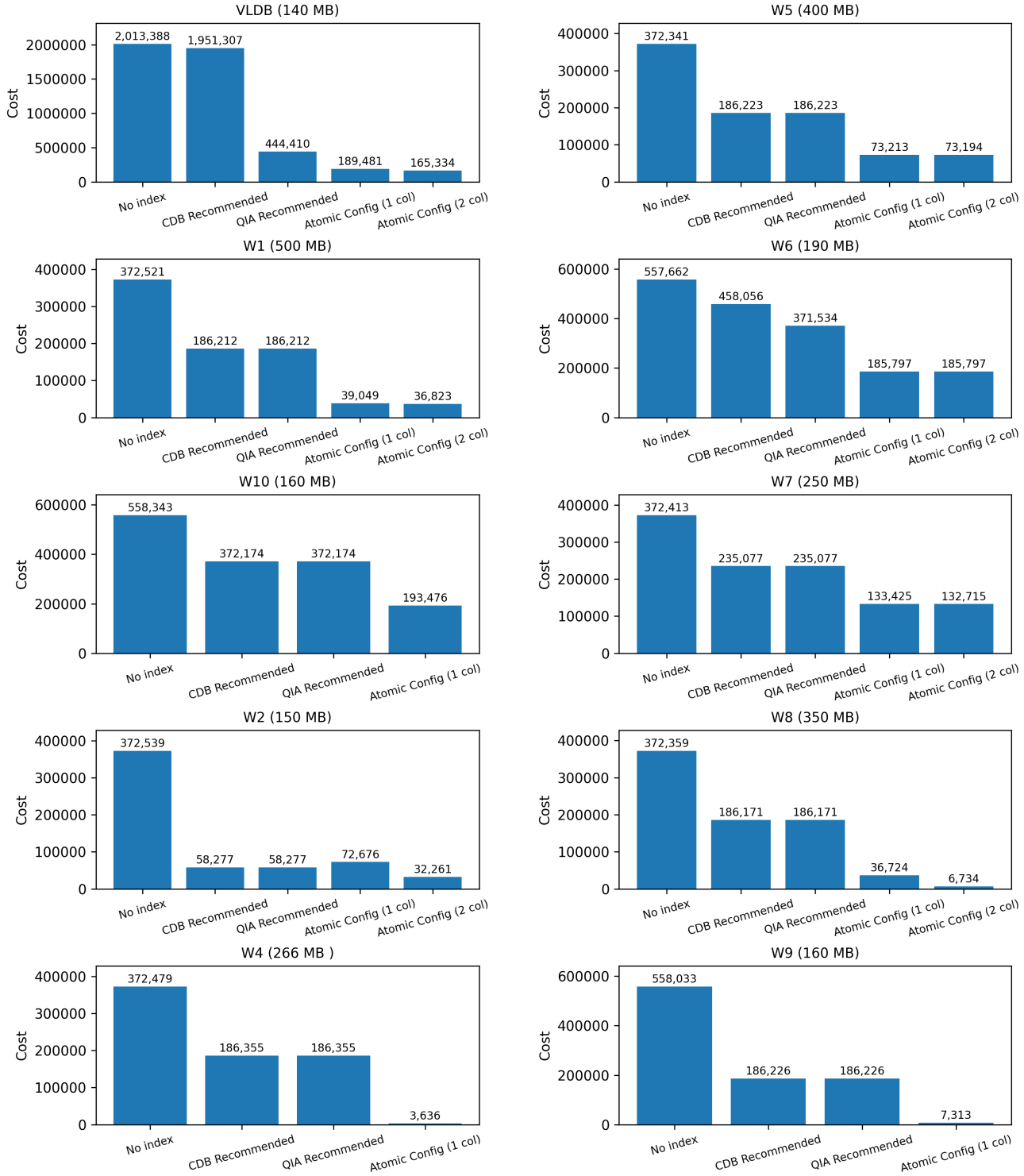


Figure 1.2: Results on IBM's DB2 Index Advisor

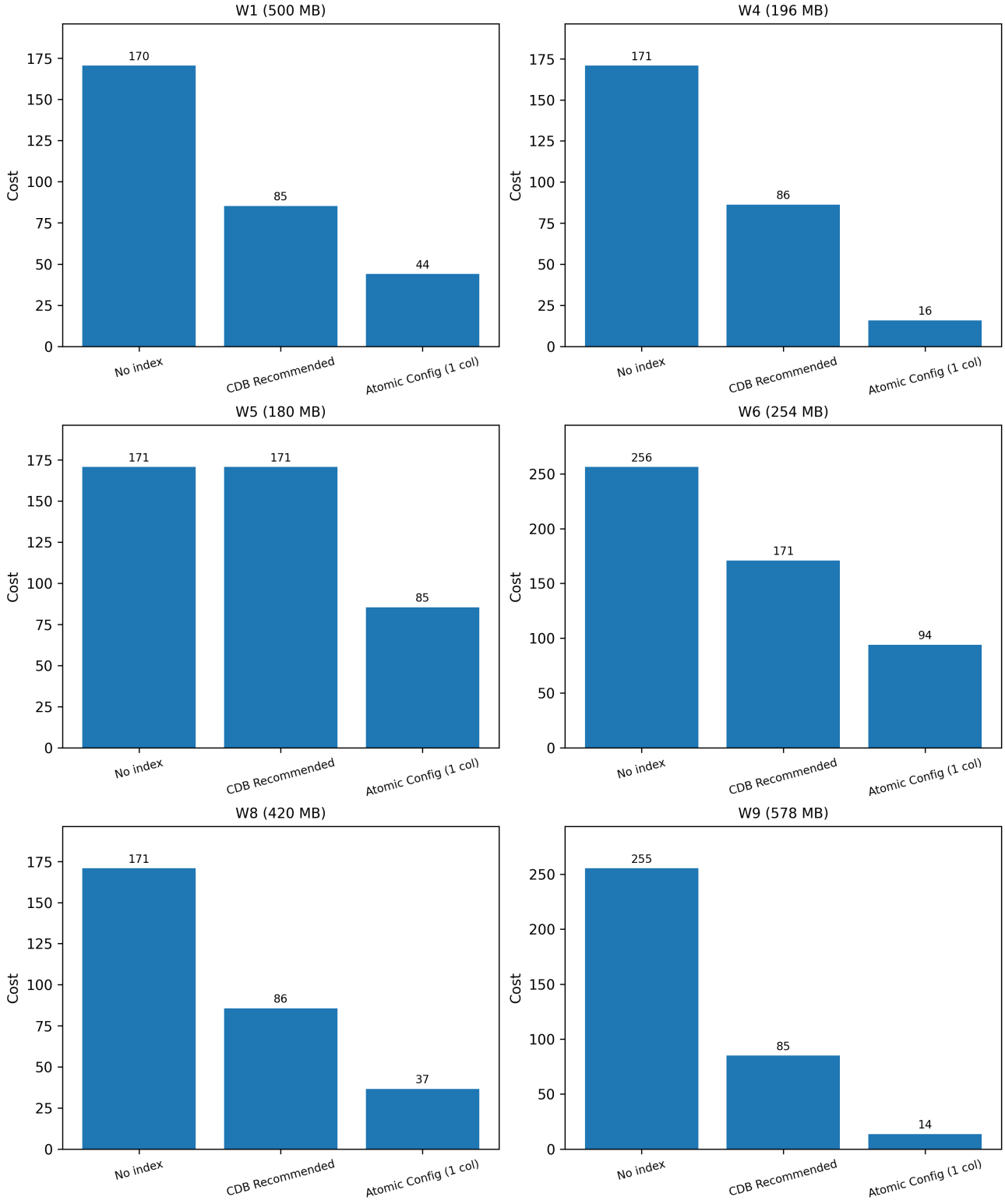


Figure 1.3: Results on Autoadmin's Index Advisor

Chapter 2

Methodology

To solve the index advisor problem, two quantum computing models can be used: 1) Quantum Annealing, 2) Quantum Circuit. Quantum Annealing is based on energy minimization, whereas gate-based quantum computing relies on quantum logic gates. The D-Wave quantum annealer is specifically designed to solve NP-hard optimization problems by formulating them as a QUBO (Quadratic Unconstrained Binary Optimization) problem. Any NP-hard problem can be mapped to a QUBO formulation and then solved using D-Wave's quantum annealer.

Since we did not have access to D-Wave's quantum hardware, we employed a gate-based approach instead. In particular, the Quantum Approximate Optimization Algorithm (QAOA) can be used to solve QUBO problems in the gate-based model. QAOA works by converting the QUBO problem into a corresponding Hamiltonian and then finding an assignment of variables that minimizes the energy of this Hamiltonian. However, our same formulation can also be used on D-Wave, and it can solve much larger problems than the gate-based approach. We only have 127 qubits (IBM provides a 127-qubit machine with a quantum time of 10 mins free per month), whereas D-Wave has a 5000-qubit quantum machine.

2.1 Preprocessing Step

2.1.1 Find Atomic Configurations

The cost of any non-atomic configuration can be effectively calculated using atomic configurations, as we saw earlier. This method reduces the need for any optimizer calls once the atomic configurations for all the queries of the workload are computed.

2.1.2 Identify Indexable columns

Extract all the columns that appear as sargable predicates. Columns appearing in GROUP BY, HAVING, ORDER BY, and SELECT clauses can also be considered for indexes.

2.1.3 Compute Storage Space Consumed By Each Indexabe Column

After identifying the set of indexable columns, compute the storage space required for each of these indices. This can be done using a query optimizer by physically creating the index and retrieving the storage space information from the optimizer. An alternative is to estimate the size of the index using various standard formulas; however, these estimates may not always be accurate.

2.1.4 Deriving the cost of Index Configurations

The cost of a query under any non-atomic configuration can be derived as:

$$\text{Cost}(Q, C) = \min_{C_i \subseteq C} \text{Cost}(Q, C_i); \text{C\_i are the atomic configurations of query Q.}$$

Since we have restricted ourselves to read-only queries (select query), the inclusion of an index in a configuration can only reduce the cost. Thus, we pick the maximal atomic configurations, and among the maximal atomic configurations, we pick the atomic configuration with the least cost.
A solution to the QUBO problem will be valid if it satisfies the following constraints:

- Only one atomic configuration for each query should be picked.
- At most one clustered index on each table.
- The indexes corresponding to the selected atomic configurations for each query should be a subset of the recommended index set over the workload.
- For each query, the selected atomic configurations should be maximal in size, and among the maximal ones, it must be minimal in cost.
- The recommended index configuration must be within the storage limit specified by the user.

Chapter 3

QUBO Transformation

3.1 Notations

We have used several variables and symbols in our QUBO formulation. All of those are defined in Table 3.1.

3.2 Exactly one atomic configuration selected per query

To ensure that only one atomic configuration per query should be picked, we use the below formulation:

$$S = \sum_{i=1}^{n} \left(\sum_{j=1}^{l_{q_i}} x_{q_i}^{j} - 1 \right)^2 \tag{3.1}$$

While trying to minimize the above equation to 0, for each atomic configuration of query q_i only one of the $x_{q_i}^{j}$ will be 1 rest will be 0. This means only one atomic configuration is picked per query.

3.2.1 Atmost one clustered index per relation

One clustered index per table can be represented in the below quadratic form:

$$B = \sum_{t \in W} \sum_{k=1}^{c_t} y c_t^k \leq 1$$

To convert above equation to QUBO form we introduce a slack variable s_b. The above equation can now be converted into QUBO form as follows:

$$B = \sum_{t \in W} \sum_{k=1}^{c_t} yc_t^k - 1 + s_b \tag{3.2}$$

$$s_b \in \{0, 1\}$$

3.3 Selected Atomic Configuration must be subset of recommended indexes

To ensure that atomic configuration picked per query should be subset of I i.e the index set over the workload, the below formulation is made:

$$V = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} x_{q_i}^j \left(\sum_{t_i \in q_i} \sum_{k=1}^{c_i} zc_{t_i}^{k_j} \cdot (1 - yc_{t_i}^k) + zu_{t_i}^{k_j} \cdot (1 - yu_{t_i}^k) \right) \tag{3.3}$$

In the above formulation, the two outermost sums iterate over all the atomic configurations for each query. Now if some atomic configuration $x_{q_i}^j$ is selected, the inner sum iterates over all the indexes present in atomic configuration $x_{q_i}^j$. For every index present in atomic configuration $x_{q_i}^j$, it checks whether the same index is being picked in the final recommended index or not.

3.4 Selection of maximal Size Atomic Configuration with minimal cost

3.4.1 Maximal Size Atomic Configurations

Here, maximal size atomic configurations refers to those atomic configurations that have larger number of indexes in it. We want to prioritize those atomic configurations that have more indexes so penalty should be lesser to them and vice versa. So, we define $f_{x_{q_i}^j}$ as below:

$$f_{x_{q_i}^j} = 1 - \frac{|A_{q_i}^j|}{|A_{q_i}^{max}|} \tag{3.4}$$

Now to ensure we select maximal atomic configurations we use the below formulation:

$$M = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} f_{x_{q_i}^j} . x_{q_i^j} \tag{3.5}$$

Type	Symbol	Description	Domain
Input	n	Number of queries in Workload	$\mathbb{R}$
Input	q_i	ith query of workload where $i \in [1, n]$	–
Input	W_{max}	Storage constraint	$\mathbb{R}$
Input	l_{q_i}	Total atomic configurations of query q_i	$\mathbb{Z}$
Input	c_{t_i}	Number of columns in table $t_i \in q_i$ where $i \in [1, n]$	$\mathbb{Z}$
Input	$x_{q_i}^j$	Indicator variable denoting j^{th} atomic configuration of query q_i	$\{0, 1\}$
Input	$A_{q_i}^j$	Denotes j^{th} atomic configuration of query q_i	–
Input	$A_{q_i}^{max}$	Denotes the atomic configuration with the most indexes among all atomic configurations of query q_i	–
Output	yc_t^k and yu_t^k	Indicator variable for index on k^{th} column of table t. yc indicates clustered index, yu indicates unclustered index. This defines which indexes are present over the workload.	$\{0, 1\}$
Input	$zc_{t_i}^{k_j}$ and $zu_{t_i}^{k_j}$	Indicator variable for index on k^{th} column of table t_i in j^{th} atomic configuration of query q_i. zc for clustered, zu for unclustered index.	$\{0, 1\}$
Input	$W(y_t^k)$	Total storage space taken by index on k^{th} column of table t	$\mathbb{Z}$
Input	$Cost(q_i, A_{q_i}^j)$	Cost of query q_i under j^{th} atomic configuration, computed via optimizer	$\mathbb{Z}$
Input	W	Workload comprising of n SQL queries	–

Table 3.1: Table of Inputs, Parameters, and Domains

3.4.2 Minimal Cost

The workload cost can be computed through the selected atomic configurations as follows:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} \left(\frac{Cost(q_i, A_{q_i}^{j})}{Cost(q_i, A_{q_i}^{max})} \right) .x_{q_i}^{j} \tag{3.6}$$

Here,the cost is normalized to make sure the coefficients of all binary variables are small. While minimizing the above equation, it will try to select those atomic configurations that have lesser cost.

Overall equations (3.5) and (3.6) together make sure that out of the maximal size atomic configurations minimal cost atomic configurations are picked.

3.5 Capacity Constraint

Now we want to make sure that the recommended index set over the workload must be within the storage budget constraint specified by user. This can be expressed as below:

$$\sum_{t \in W} \sum_{k=1}^{c_t} y_{t_i}^{k} . W(y_{t_i}^{k}) \leq W_{max}. \tag{3.7}$$

The above can be converted to a QUBO form. We introduce a slack variable for the inequality constraint and equation (5) can be rewritten as follows:

$$W_{max} - \sum_{t \in W} \sum_{k=1}^{c_t} y_{t_i}^{k} . W(y_{t_i}^{k}) = \sum_{m=1}^{log_2(W_{max})} 2^m * s_m \tag{3.8}$$

$$s_m \in \{0, 1\}^{log_2(W_{max})}$$

Now equation 3.8 can be reformulated as:

$$C = \left(W_{max} - \sum_{t \in W} \sum_{k=1}^{c_t} y_t^{k} . W(y_t^{k}) - \sum_{m=1}^{log_2(W_{max})} 2^m * s_m \right)^2 \tag{3.9}$$

However after expanding the above equation, the coefficients of binary variables can become very large. To avoid this, the above equation is normalized by dividing each term by W_{max}. So (3.9) will now become:

$$C = \left(1 - \sum_{t \in W} \sum_{k=1}^{c_t} \frac{y_t^k . W(y_t^k)}{W_{max}} - \sum_{m=1}^{log_2(W_{max})} \frac{2^m * s_m}{W_{max}}\right)^2 \tag{3.10}$$

3.6 Putting it all together

$$\min_{yc_t^k, yu_t^k, x_{q_i}^j, s_m, s_b} (p_s \cdot S + B + p_v \cdot V + O + p_m \cdot M + p_c \cdot C)$$

$$S = \sum_{i=1}^{n} \left(\sum_{j=1}^{l_{q_i}} x_{q_i}^j - 1\right)^2$$ (Enforces one atomic configuration picked per query)

$$B = \sum_{t \in W} \sum_{k=1}^{c_t} yc_t^k - 1 + s_b$$ (Atmost one clustered index per table)

$$V = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} x_{q_i}^j \left(\sum_{t_i \in q_i} \sum_{k=1}^{c_i} zc_{t_i}^{k_j} \cdot (1 - yc_{t_i}^k) + zu_{t_i}^{k_j} \cdot (1 - yu_{t_i}^k)\right)$$

(Makes sure that selected atomic configurations must be subset of recommended index configuration)

$$O = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} \left(\frac{Cost(q_i, A_{q_i}^j)}{Cost(q_i, A_{q_i}^{max})}\right) \cdot x_{q_i}^j$$ (Normalized cost of workload)

$$f_{x_{q_i}^j} = 1 - \frac{|A_{q_i}^j|}{|A_{q_i}^{max}|}$$

$$M = \sum_{i=1}^{n} \sum_{j=1}^{l_{q_i}} f_{x_{q_i}^j} \cdot x_{q_i^j}$$ (Makes sure that maximal size atomic configurations are selected)

$$C = \left(1 - \sum_{t \in W} \sum_{k=1}^{c_t} \frac{y_t^k . W(y_t^k)}{W_{max}} - \sum_{m=1}^{log_2(W_{max})} \frac{2^m * s_m}{W_{max}}\right)^2$$

(Enforces capacity constraint)

If no weights are assigned i.e $p_s = p_v = p_m = p_c = 1$ then the minimal solution of given QUBO will select atomic configurations with minimal cost without looking at their sizes.

In order to enforce the maximal size of atomic configurations, a small penalty can be applied to M. In our experiments, a value of $p_m = 2$ was found to be sufficient. Suppose all other weights are set to 1—then the minimal solution would be one where no atomic configuration for any query is selected. This is because the incentive to avoid the penalty p_m outweighs the benefit of selecting any atomic configuration. Therefore, a penalty $p_s \geq p_m + 1$ should be assigned to S to ensure that atomic configurations for queries are indeed selected. In our experiments, we assigned $p_s = 3$. Similarly, to prevent subset constraint violations, a penalty $p_v \geq p_m + 1$ should also be applied; we set $p_v = 3$ in our experiments. The penalty p_c can be any positive value; however, in our tests, $p_c = 1$ was sufficient to avoid any capacity constraint violations.

The number of variables used determines the lower bound of total number of qubits required as each variable requires atleast one qubit. The number of variables are as follows:

1. $x_{q_i}^j$: This variable represents the jth atomic configuration of query q_i. There are n queries and number of atomic configurations of query q_i is l_{q_i}. So total variables of this type is: $\sum_{i=1}^{n} l_{q_i}$.

2. yc_t^k: This variable defines which clustered indexes are present over the workload. A variable exist for each columns present in each query. In worst case a variable can exist for every table.

3. yu_t^k: This variable defines which unclustered indexes are present over the workload. A variable exists for each columns present in each query. In worst case a variable can exist for each column of every table.

4. s_m: From the equation we can see there are total $log_2(W_{max})$ slack variables.

5. s_b: One slack variable was used to make sure atmost one clustered index was used on every table. In worst case number of such variables will be equal to all the tables of our

database.

Number of qubits needed = $\sum_{i=1}^{n} l_{q_i} + 2 \cdot \sum_{t \in W} \sum_{k=1}^{c_t} k + log_2(W_{max}) + 1 + \sum_{t \in W} 1$

In a nutshell, number of qubits needed = O(Number of Atomic Configurations) + O(Total columns in query) + $O(\log W_{max})$.

Since the number of atomic configurations that we are computing for each query is quadratic to the number of columns present in those queries, the number of qubits required is also quadratic to the number of columns present in the workload.

3.7 Formal Analysis

In this section, we will prove that the minimal solution of the QUBO problem will always give the optimal set of single column indexes computed through 1-column atomic configurations. Basically our formulation will satisfy the below equation for all the queries of the workload:

$$\text{Cost}(Q, C) = \min_{C_i \in \mathbb{C}_{\max}^{q_i}} \text{Cost}(Q, C_i) \tag{3.11}$$

Here, C is the recommended index configuration of the workload, $C_i \subseteq C$, storage space consumed by indexes in C must be less than W_{max} and $C_{max}^{q_i}$ are the maximal atomic configurations of query q_i within the storage budget W_{max}.

Claim: The minimal solution of QUBO always satisfies equation (3.11).

Let A_1 and A_2 be atomic configurations of query q_i. Without loss of generality, let's assume S, V, and B are 0 and C is some negligible constant (i.e, both the atomic configurations are within the space constraint W_{max}) when only A_1 or only A_2 is selected by QUBO for query q_i. We are now left with $O + p_m \cdot M$.
Let's discuss two cases:

1) If $|A_1| > |A_2|$

This means $f_{A_1} < f_{A_2}$ as the penalty for a larger size atomic configuration is less than the penalty for a smaller size atomic configuration.

To make sure A_1 is preferred over A_2 the below constraint should be satisfied:

$$p_m \cdot M_{A_1} + O_{A_1} < p_m \cdot M_{A_2} + O_{A_2} \tag{3.12}$$

If $p_m \cdot M_{A_2} > p_m \cdot M_{A_1} + 1$ then (3.12) is always true because $O_{A_1} \in [0, 1]$. Thus if $p_m > \frac{1}{M_{A_2} - M_{A_1}}$ then equation (3.12) will always be true even if cost of query q_i corresponding A_1 is more than cost of query q_i corresponding A_2. In the experiments done by me p_m was assigned as 2, and it worked, but depending on the problem instance, p_m needs to be computed beforehand.

2) If $|A_1| = |A_2|$ and cost(q,A_1) ¡ cost(q,A_2)

In this case also it is easy to see that A_1 will be preferred over A_2 as it incurs lower cost corresponding to A_2 i.e equation (3.12) will also hold true for this case.

Thus, the QUBO formulation proposed by us always selects valid atomic configurations that are subsets of recommended index configuration and also selects maximal size minimal atomic configurations thereby making sure the minimal solution to our QUBO problem is also the best solution for the index selection problem.

Chapter 4

Solving QUBO using QAOA

The QAOA is a general technique that can be used to find approximate solutions to combinatorial optimization problems, in particular problems that can be cast as searching for an optimal bitstring. QAOA consists of the following steps:

1. Define a cost hamiltonian H_C such that its ground state encodes the solution to the optimization problem.
2. Define a mixer hamiltonian H_M.
3. Construct the circuits $e^{i\gamma H_C}$ and $e^{i\alpha H_M}$. These are called cost and mixer layers, respectively.
4. Choose a parameter $p \geq 1$ and build the circuit

$$U(\gamma, \alpha) = e^{i\gamma_p H_M} e^{i\gamma_p H_C} ... e^{i\gamma_1 H_M} e^{i\gamma_1 H_C}$$

   consisting of repeated cost and mixer layers.
5. Prepare an initial state $U(\gamma, \alpha)$ and use classical techniques to optimize the parameters.
6. After the circuit has been optimized, measurements of the output state reveal approximate solutions to the optimization problem.

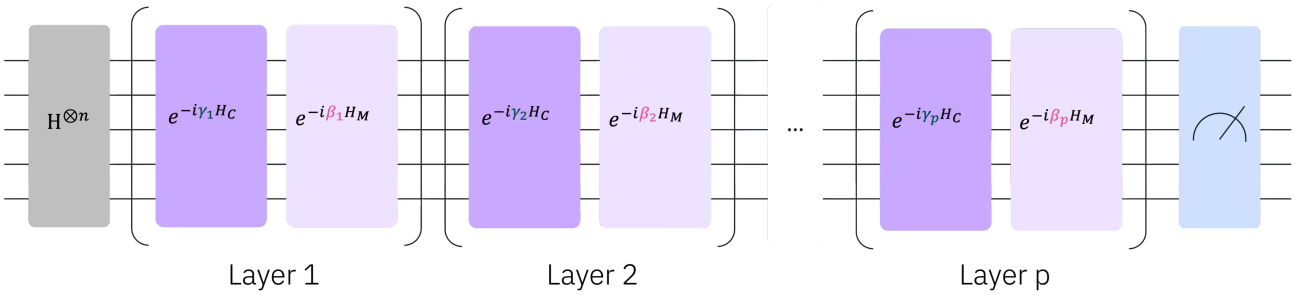


Figure 4.1: QAOA Protocol [7]

In summary, the starting point of QAOA is the specification of cost and mixer Hamiltonians. Then time evolution and layering is used to create a variational circuit and optimize it's parameters. The algorithm concludes by sampling from the circuit to get an approximate solution to the optimization problem.

Chapter 5

Experiments

We have used Qiskit's QAOA library to solve the QUBO. But there are three things than we need to find out in order to get good results:

- Value of p
- Initial parameters for QAOA
- Which classical optimizer to use

For initializing the parameters of QAOA circuit we have used the fourier initialization technique proposed in [12]. They claim that their initialization technique performs better than random initialization technique that too at lower value of p.

[1] does a comprehensive study of how different optimizers work with QAOA and for noisy quantum machines it is best to use gradient-free methods. We used two gradient-free methods: COBYLA and POWELL. In the experiments done by us, we found that POWELL takes many more function evaluations than COBYLA; thus, POWELL is computationally expensive. Hence, we avoided using POWELL while submitting jobs on the actual hardware machine of IBM.

Since in IBM's Qiskit SDK we can only simulate up to 29 qubits, not all the workloads could be simulated. Also, at the moment, we only have hardware results for one workload (W1). Because of high queuing time and very limited quantum time given for free accounts, other experiments are still pending. Another issue that we are encountering is that we are not able to find an optimizer that gives good quality solutions with a lesser number of function evaluations (meaning less expensive).

Table 5.1: **Experiments (Simulator)**

Workload Name	**p**	**Optimizer**	**Number of function evaluations**	**Qubits**	**Depth**	**Output**
W1	1	COBYLA	34	20	97	Same as AC.1
W5	1	COBYLA	30	23	115	Same as AC.1

Table 5.2: **Experiments (Hardware)**

Workload Name	**p**	**Optimizer**	**Number of function evaluations**	**Qubits**	**Depth**	**Output**	**Machine Name**
W1	1	COBYLA	30	20	97	Same as AC_1	`ibm_sherbrooke (127-qubit)`

To evaluate the solution quality of our QUBO formulation, we used the CPLEX solver. And for all the constructed workloads, we got optimal single column indexes under constructed atomic configurations for the workloads for both the database engines, i.e., IBM DB2 as well as Microsoft's Autoadmin. By optimal, we mean the solution from QUBO was the same as the brute force solution for single column indexes under constructed atomic configurations for all the workloads and under both the database engines. To evaluate our experiments, you can navigate to our GitHub repository [3].

Chapter 6

Related Work

Two papers formulate the index selection problem as an optimization problem. [] introduces an ILP based approach to solving the index selection problem and [] introduces a QUBO formulation for the same. Let's review both of these papers in detail.

6.1 ILP Based Approach [5]

6.1.1 Summary

This paper proposes a model for index selection based on Integer Linear Programming (ILP). They have formulated their index selection problem as: Given a workload consisting of m queries and a set of n indexes I_1 - I_n, with sizes s_1 - s_n, find a configuration C for the workload under storage constraint S that gives maximum benefit. Here benefit is calculated as the difference between the workload cost without configuration C and the workload cost with configuration C.

They have used the notion of atomic configurations originally proposed in Autoadmin's first paper in 1997 [2]. Their ILP formulation takes into account the following constraints:

1. Select those atomic configurations whose indexes are present in the selected configuration C.

2. The total space taken by all the indexes in C must be within the storage constraint S.

3. Each query must use at most one atomic configuration.

To solve the ILP problem, they use the branch and bound method. It gives optimal solution if the program runs to completion or else it can be interrupted mid-way and a sub-optimal solution is returned. For large problems the ILP might not run to completion so they propose

using the branch and bound method only for a few iterations and stop the search once a solution with an acceptable distance from optimal has been reached.

6.1.2 Strong Points

1. The upper bound on maximum benefit can be computed due to which we can see how close to optimal we are. It helps the user to decide whether further increase in storage is needed to get better quality solutions or not.

6.1.3 Weak Points

1. Though they have used the concept of atomic configurations, the original paper [] that proposed the idea recommends selecting those atomic configurations for each query that are maximal in size (within the budget) and amongst the maximal atomic configurations, select the one with the least cost. This is not being done by this paper; they simply select the ones that give maximum benefit.

2. The experiments section contains only one experiment done on 5 TPC-H queries. Information about which of the TPC-H queries was used is also not given.

6.2 Annealing based approach [10]

6.2.1 Summary

This paper proposes a QUBO for solving the index selection probem on a DWave Quantum Annealer. This paper introduces certain techniques for exploiting the qubits of the quantum annealer, which in turn helps them to solve larger-sized problem instances that can be represented with the given number of qubits. They have formulated the index selection problem as follows: Given a workload, a set of index candidates, and a storage space bound, determine an optimal subset of indices that gives maximum benefit under given storage constraint.

Their QUBO formulation has three aspects: i) Utility, ii) Storage Constraint, iii) Mutual Exclusion (Number of clustered indices per table should be atmost one). Utility can be regarded as the benefit of generating an index. So their QUBO formulation tries to maximize the overall utility of the workload. They propose two alternative formulas for both storage constraint as well as mutual exclusion. The number of qubits needed is not actually bounded by the number of variables but by the number of quadratic products between decision variables that appear in the formulation. So, to reduce the number of qubits needed, they propose a formulation that has a lesser number of quadratic products than the naive quadratic formula for both mutual exclusion as well as storage constraint.

6.2.2 Strong Points

1. Reducing the number of qubits needed by transforming the QUBO into a form that has a smaller number of quadratic products.

2. A physical embedding algorithm that maps qubits to variables that exploit the connections between variables and helps them map larger problem instances.

6.2.3 Weak Points

1. They have not stated the model used for finding the utility of the index.

2. The solution quality is restricted by the initial candidate set.

Chapter 7

Conclusion and Future Work

Even after restricting ourselves to only single-column indexes, we are able to recommend significantly better-quality index recommendations than commercial index advisors, as well as QIA.

In the future, when more number of qubits are available, our formulation can be extended to multi-column indexes. Our formulation can also be used on D-Wave's quantum machine, which has a much larger number of qubits (D-Wave's Advantage Quantum Computer has over 5000 qubits!).

Bibliography

[1] Konstantinos Blekos, Dylan Brand, Alberto Ceschini, Chia-Hsiu Chou, Runyu Li, Kunal Pandya, and Andrew Summer. A review on quantum approximate optimization algorithm and its variants. *Physics Reports*, 2023. doi: 10.1016/j.physrep.2024.03.002. URL https://doi.org/10.1016/j.physrep.2024.03.002. In press. 19

[2] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, page 146–155, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604707. 3, 4, 21

[3] Mohit Jha and DSL-IISc. Qubo formulation of index selection problem. https://github.com/DSLIISC/Mohit_Final, 2025. 20

[4] Manish Kesarwani and Jayant R. Haritsa. Index advisors on quantum platforms. *Proc. VLDB Endow.*, 17(11):3615–3628, July 2024. ISSN 2150-8097. doi: 10.14778/3681954.3682025. URL https://doi.org/10.14778/3681954.3682025. ii, 4, 27

[5] Stratos Papadomanolakis and Anastassia Ailamaki. An integer linear programming approach to database design. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, ICDEW '07, page 442–449, USA, 2007. IEEE Computer Society. ISBN 9781424408313. doi: 10.1109/ICDEW.2007.4401027. URL https://doi.org/10.1109/ICDEW.2007.4401027. iv, 21

[6] Gregory Piatetsky-Shapiro. The optimal selection of secondary indices is np-complete. *SIGMOD Rec.*, 13(2):72–75, January 1983. ISSN 0163-5808. doi: 10.1145/984523.984530. URL https://doi.org/10.1145/984523.984530. 1

[7] Qiskit Tutorials. Qiskit tutorials. https://www.ibm.com/quantum/qiskit#tutorials, 2025. Retrieved June 9, 2025. v, 18

[8] Alan Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the 16th International Conference on Data Engineering*, ICDE '00, page 101, USA, 2000. IEEE Computer Society. ISBN 0769505066. 4

[9] Transaction Processing Council. TPC-H. https://www.tpc.org/tpch, n.d. Accessed: June 24, 2025. 4

[10] Immanuel Trummer and Davide Venturelli. Leveraging quantum computing for database index selection. In *Proceedings of the 1st Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*, Q-Data '24, page 14–26, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705533. doi: 10.1145/3665225.3665445. URL https://doi.org/10.1145/3665225.3665445. iv, 22

[11] Sai Wu, Dawei Jiang, Beng Chin Ooi, and Kun-Lung Wu. Efficient b-tree based indexing for cloud data processing. *Proc. VLDB Endow.*, 3(1–2):1207–1218, September 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920991. URL https://doi.org/10.14778/1920841.1920991. 1

[12] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2):021067, 2020. doi: 10.1103/PhysRevX.10.021067. URL https://doi.org/10.1103/PhysRevX.10.021067. 19

Appendix

Experiment Workloads (Based on the corresponding TPC-H schema)

VLDB QIA Workload [4]

```
SELECT SUM(L_EXTENDEDPRICE * L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= DATE '1994-01-01' AND L_SHIPDATE < DATE
   ↪ '1995-01-01' AND L_DISCOUNT BETWEEN 0.05 AND 0.07 AND L_QUANTITY
   ↪  < 24;
SELECT 100.00 * SUM(CASE WHEN P_TYPE LIKE 'PROMO%' THEN
   ↪ L_EXTENDEDPRICE * (1 - L_DISCOUNT) ELSE 0 END) / SUM(
   ↪ L_EXTENDEDPRICE * (1 - L_DISCOUNT)) AS PROMO_REVENUE FROM
   ↪ LINEITEM, PART WHERE L_PARTKEY = P_PARTKEY AND L_SHIPDATE >=
   ↪ DATE '1995-09-01' AND L_SHIPDATE < DATE '1995-10-01';
SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
   ↪ FROM (SELECT SUBSTRING(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL FROM
   ↪  CUSTOMER WHERE SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23',
   ↪ '29', '30', '18', '17') AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL)
   ↪ FROM CUSTOMER WHERE C_ACCTBAL > 0.00 AND SUBSTRING(C_PHONE,1,2)
   ↪ IN ('13', '31', '23', '29', '30', '18', '17')) AND NOT EXISTS (
   ↪ SELECT * FROM ORDERS WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
   ↪ GROUP BY CNTRYCODE ORDER BY CNTRYCODE;
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
   ↪ WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'BRAND#23' AND
   ↪ P_CONTAINER = 'MED BOX' AND L_QUANTITY < (SELECT 0.2*AVG(
   ↪ L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY);
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
   ↪ WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'BRAND#23' AND
   ↪ P_CONTAINER = 'MED BOX' AND L_QUANTITY < (SELECT 0.2*AVG(
```

```
   ↪ L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY);
```

W1

```
SELECT SUM(L_QUANTITY) FROM LINEITEM WHERE L_SHIPDATE >= '1994-12-31'
   ↪ AND L_SHIPDATE < '1995-01-01';
SELECT * FROM LINEITEM WHERE L_QUANTITY > 49 AND L_DISCOUNT > 0.099;
```

W2

```
SELECT MAX(L_DISCOUNT) FROM LINEITEM WHERE L_SHIPDATE >= '1990-12-31'
   ↪  AND L_SHIPDATE < '1995-01-01' AND L_RECEIPTDATE >= '1994-12-31'
   ↪  AND L_RECEIPTDATE <  '1995-01-01';
SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_DISCOUNT > 0.099;
```

W4

```
SELECT SUM(L_EXTENDEDPRICE),L_ORDERKEY,L_SUPPKEY,L_TAX FROM LINEITEM
   ↪ WHERE L_SHIPDATE >= '1994-12-31' AND L_SHIPDATE < '1995-01-01'
   ↪ GROUP BY L_ORDERKEY,L_SUPPKEY,L_TAX;
SELECT L_SHIPDATE,SUM(L_DISCOUNT),AVG(L_EXTENDEDPRICE) FROM LINEITEM
   ↪ WHERE L_QUANTITY > 49 AND L_TAX > 0.01 AND L_RECEIPTDATE >=
   ↪ '1990-12-31' AND L_RECEIPTDATE < '1991-01-01' GROUP BY
   ↪ L_SHIPDATE;
```

W5

```
SELECT L_TAX,L_ORDERKEY FROM LINEITEM WHERE L_LINENUMBER = 6 AND
   ↪ L_DISCOUNT = 0 GROUP BY L_TAX,L_ORDERKEY;
SELECT SUM(L_QUANTITY) FROM LINEITEM WHERE L_TAX > 0.01 AND
   ↪ L_RETURNFLAG = 'N' AND L_PARTKEY = 200000;
```

W6

```
SELECT L_ORDERKEY FROM LINEITEM WHERE L_DISCOUNT > 49;
SELECT SUM(L_QUANTITY)  FROM LINEITEM WHERE L_LINENUMBER = 6 AND
   ↪ L_EXTENDEDPRICE > 104928 GROUP BY L_QUANTITY HAVING L_QUANTITY >
   ↪  49;
SELECT L_PARTKEY FROM LINEITEM WHERE L_TAX > 0.01;
```

W7

```
SELECT L_ORDERKEY,L_TAX,L_DISCOUNT FROM LINEITEM WHERE L_DISCOUNT >
   ↪ 0.099 AND L_QUANTITY > 49 GROUP BY L_ORDERKEY,L_TAX,L_DISCOUNT;
SELECT AVG(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_TAX > 0.01 AND
   ↪ L_SHIPMODE = 'AIR';
```

W8

```
SELECT AVG(L_DISCOUNT) FROM LINEITEM WHERE L_COMMITDATE > '1992-12-31'
   ↪  AND L_COMMITDATE < '1993-01-01';
SELECT SUM(L_EXTENDEDPRICE),L_ORDERKEY,L_PARTKEY FROM LINEITEM WHERE
   ↪ L_DISCOUNT = 0 AND L_TAX > 0.01 AND L_QUANTITY > 49 GROUP BY
   ↪ L_ORDERKEY,L_PARTKEY;
```

W9

```
SELECT L_SHIPINSTRUCT,L_SHIPMODE,L_COMMITDATE FROM LINEITEM WHERE
   ↪ L_QUANTITY > 49 AND L_TAX > 0.099 GROUP BY L_SHIPINSTRUCT,
   ↪ L_SHIPMODE,L_COMMITDATE;
SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_DISCOUNT = 0 AND
   ↪ L_RECEIPTDATE = '1991-01-01';
SELECT L_PARTKEY FROM LINEITEM WHERE L_SHIPDATE = '1993-01-01' GROUP
   ↪ BY L_PARTKEY;
```

W10

```
SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_TAX > 0.01;
SELECT L_ORDERKEY,L_QUANTITY FROM LINEITEM WHERE L_RECEIPTDATE =
   ↪ '1990-04-23' GROUP BY L_ORDERKEY,L_QUANTITY HAVING L_QUANTITY >
   ↪ 49;
SELECT AVG(L_DISCOUNT) FROM LINEITEM WHERE L_COMMITDATE >=
   ↪ '1992-12-31' AND L_COMMITDATE < '1993-01-01';
```