Query Learning with Quantum Logic

Alexander Stahl[1,*]

[1]Brandenburg University of Technology Cottbus-Senftenberg

Abstract

Query-by-Example (QBE) is a classical method for formulating database queries by filling out relation templates with example values, which avoids the need for formal query languages. Extensions such as Fuzzy QBE have introduced variants, where users label example tuples as relevant or irrelevant to guide the construction of a query condition. In this paper, we examine how such label-driven query induction can be applied to quantum logic-based query languages, focusing on the Commuting Quantum Query Language (CQQL). We assume the problem to be a form of classification and approach it using Quantum Logic Decision Trees. The resulting conditions are expressed as CQQL formulas and compiled into executable queries using Quantum SQL. We provide illustrative examples and discuss practical considerations.

Keywords

query by example, query learning, classification, quantum logic

1. From Examples to Queries

The gap between information retrieval and structured database querying has motivated many attempts to unify these two paradigms. In [1], van Rijsbergen proposed quantum logic as the foundation for such a unification effort by treating documents and queries as subspaces in a Hilbert space. Quantum logic originates from the mathematical formalism of quantum mechanics, where logical operations correspond to relations between subspaces rather than simple truth values. In contrast to Boolean logic, which restricts truth values to $\{0, 1\}$, quantum logic allows graded truth values in $[0, 1]$. In this paper, we adopt a simplified view, where these evaluations can be understood through semantics similar to probability (e.g., conjunction as product, negation as complement). This is provided in Section 4, while the formal details are accessible in [1, 2]. In line with these works, we restrict ourselves to the Hilbert-space-inspired variant of quantum logic. Other non-classical logics, such as three-valued logics or or temporal dialogical logics, are outside our scope, since our objective is to capture graded relevance rather than to mirror the truth states of SQL. Gradual truth values and probability-like evaluation schemes make quantum logic similar in spirit to fuzzy logic approaches, but with an important difference: fuzzy logic relies on t-norms, t-conorms, and negators that may violate classical logic laws, whereas quantum logic can preserve these to a higher degree. In [2], Schmitt extended van Rijsbergen's concept for a fully-fledged querying framework, introducing quantum query processing mechanics that align with both fact-based and relevance-based retrieval.

Unrelated to these advances, practical methods such as Query by Example (QBE) where developed to make relational database querying more accessible by allowing users to express their information needs through example tuples instead of writing formal SQL statements [3]. QBE reduces the technical barrier to structured querying, which is particularly valuable for non-experts. While the theoretical foundations of quantum logic querying have matured, user-friendly interaction methods like QBE are still lacking in this space. In this paper, we address this gap by introducing a QBE-style strategy that uses quantum logic decision trees (QLDT). Our method enables users to label example tuples for relevance, from which a logical, interpretable query is learned. This introduces example-based querying into the domain of quantum logic.

36th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), September 29 - October 01, 2025, Regensburg, Germany

*Corresponding author.

stahlale@b-tu.de (A. Stahl)
0009-0009-0554-107X (A. Stahl)

2. Related Work and Problem Reformulation

QBE was introduced in the 1970s as a user-friendly interface for relational databases, aimed at users unfamiliar with formal query languages like SQL [3]. In QBE, users fill out a template table by entering example values or conditions into selected fields. These entries are then interpreted as constraints on the corresponding attributes, and the system translates the filled-in template into a formal relational query. The approach allows users to express simple selection, projection, and join queries without writing explicit code. However, classical QBE is limited to Boolean logic, fixed schema interactions, and exact matches, which makes it unsuitable for more expressive logic-based frameworks with gradual relevance values.

A more recent line of work proposes generating queries directly from natural language with techniques from natural language processing, particularly using large language models (LLMs). Examples of this approach can be found in studies such as [4, 5]. These methods allow the translation of information needs into executable query languages with minimal effort by the user. However, while their appeal is clear, the black-box nature and inherent unpredictability of LLMs, despite significant recent advances, may still render them unsuitable for applications where reliability and transparency are essential.

To overcome the limitations of classical QBE in handling vague or graded relevance, Moreau et al. proposed Fuzzy Query-by-Example (FQBE) [6]. Their approach allows users to label a small set of tuples as relevant or irrelevant, and the system then infers a fuzzy characterization of the relevant examples. In particular, the method identifies attribute intervals or value patterns that are common among the positive examples and uncommon among the negatives, and constructs a fuzzy selection condition that best captures this contrast. The resulting fuzzy query is expressed using linguistic quantifiers and graded predicates, which provides a retrieval method that tolerates imprecision. While FQBE extends classical QBE with graded relevance and soft matching, it is not conceptually compatible with quantum logic-based approaches. As mentioned above, fuzzy logic, as used in FQBE, is based on numerical aggregation (e.g. t-norms and membership functions) rather than projection-based semantics or subspace logic. More importantly, fuzzy logic does not universally satisfy several core logical properties that are preserved in quantum logic! In particular, the law of the excluded middle and the law of non-contradiction hold only under specific combinations of negation and aggregation operators, and are violated in many common fuzzy systems [7]. These differences make FQBE unsuitable as a foundation for querying in quantum-logical frameworks, where the logic structure itself must be preserved throughout the inference process.

However, a closer examination of FQBE reveals that the problem it addresses is structurally equivalent to that of a classification task. The user, by labeling tuples as relevant or irrelevant, effectively provides a training set, and the system learns a selection condition that distinguishes the relevant examples from the rest. This is much like how a classifier learns a decision boundary. In this light, QBE can be viewed as an instance of logic-based supervised learning, where the goal is to infer an interpretable predicate that approximates user intent. This places the problem in the domain of query learning, where queries are induced from labeled examples [8].

The approach of viewing query construction as a classification problem based on labeled examples has been adopted in other works as well. For example, [9] propose Libra, a system that synthesizes relational selection queries by learning decision trees from input-output tuple examples. Even though it operates strictly in the classical Boolean context, its methodology is very similar to that of Fuzzy QBE. Both treat the user's labeling as a form of supervision, and both induce a condition that distinguishes relevant from irrelevant tuples. Notably, Libra does not reference earlier fuzzy QBE approaches, but can be seen as continuing the same fundamental idea of query learning. The works of these authors show that classification can be used as a tool to learn queries that represent user wishes, which motivates our approach. This perspective motivates our approach: in the following chapters, we explore how classification models can be used to learn such queries directly from labeled examples.

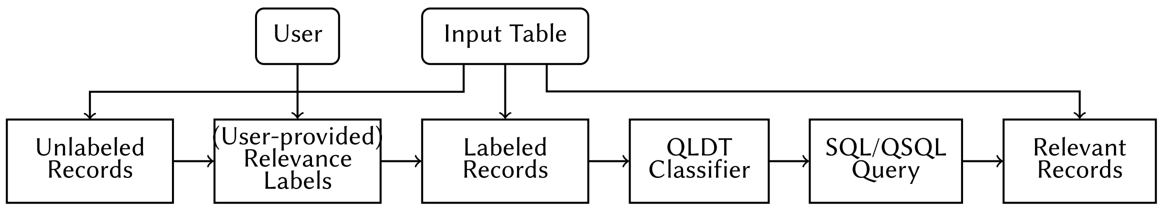

Figure 1: Process of query induction from sample records

Table 1
Example relation with ball attributes

ID	Color	Size	Label
1	Red	Large	1
2	Red	Small	0
3	Blue	Large	0
4	Red	Large	1

3. Query Induction via Classification

As an initial step towards our logic-based formulation and to motivate the next steps, we first consider a simplified version using classical decision trees (DT) and Boolean logic, with a methodology similarly to the approaches in [9, 6]. This example provides a more intuitive foundation for the quantum logic-based method introduced in the next sections.

A DT is a classification model that partitions the input space by recursively applying conditions on attribute values, resulting in a tree structure where each internal node represents a decision based on an attribute, and each leaf corresponds to a classification outcome. It is a classical and widely used model, particularly valued for its interpretability, as each decision path from root to leaf can be translated into a Boolean expression that explains why a particular classification was made [10]. For binary classification, a decision tree can be represented by a single Boolean formula that captures the conditions under which an input is assigned the positive class. Aside from XAI aspects, this property is also useful in database contexts, as the resulting Boolean expression can be directly embedded into an SQL query, which would effectively transform a learned classifier into a selection predicate.

In practice, the process proceeds as follows. We begin by collecting labels from the user, who marks a selection of randomly chosen tuples from the input relation as either relevant or not relevant. Each labeled tuple is treated as a feature vector for training, where the relevance label serves as the class, and each column in the relation corresponds to an attribute. A classification algorithm, such as CART [11], is then applied to learn a DT. The resulting tree is transformed into a Boolean expression that represents the selection condition for the relevant class. This expression can be inserted into the `WHERE` clause of a prepared SQL query. The process of this approach is illustrated in Figure 1. While we focus here on single-table selections for simplicity, this method can be extended to support `JOIN` and more complex queries.

For a simple example, consider the toy dataset in Table 1, which describes colored balls of differing size. The column "Label" refers to the user-provided relevance evaluation and encodes "irrelevant" as 0 and "relevant" as 1. Training a DT classifier on this dataset could result in a simple DT such as depicted in Figure 2.

This DT can be expressed as a Boolean expression, which expresses the truth value of the statement: "The record is relevant". Such an expression would be:

$$(\mathit{Color} = \text{'Red'}) \land (\mathit{Size} = \text{'Large'})$$

Incorporating the expression into an SQL statement, would yield the query in the following Listing 1:

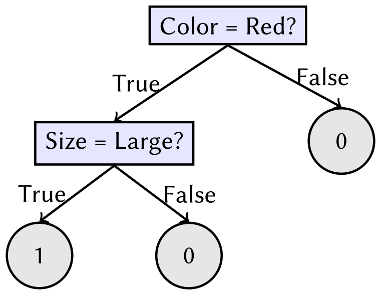

Figure 2: DT for ball example

Listing 1: SQL query with condition from the DT

```
SELECT * FROM Balls
WHERE Color = 'Red' AND Size = 'Large';
```

This example demonstrates how a classification model, in this case a DT, can be used to derive a logical query predicate from user-labeled data. The resulting condition is interpretable, executable, and mirrors the user's intent. In the following sections, we extend this approach beyond Boolean logic by explaining and applying a quantum logic-based classifier instead of the DT, which adds support for graded relevance.

4. The QLDT Model

A QLDT [12] is a classification model for tabular data that uses quantum-inspired logic at its core. Note that the usage of quantum logic does not, by itself, imply any connection to quantum computing. Although quantum logic borrows mathematical ideas from quantum mechanics, it operates on classical computers and does not involve concepts such as superposition, entanglement, or quantum circuits.

A QLDT consists of a CQQL condition on normalized attribute values and a threshold. CQQL refers to Commuting Quantum Query Language [2]. It allows for logic conditions that are syntactically equivalent to propositional logic and preserve many rules of Boolean algebra (such as commutativity and De Morgan's rules), while the truth values are in $[0, 1]$ instead of $\{true, false\}$.

For brevity, we do not describe the full formalism of CQQL here and instead refer to [13] for a simplified explanation based on probability theory. In this interpretation, the evaluation of a condition e against a feature vector o, which is notated as $[e]^o$, works as follows. Atomic conditions a denote the event that a given attribute of an object o "is high" and their evaluation $[a]^o$ returns the corresponding attribute value of o normalized to $[0, 1]$. Disjunctions such as $e_{or} = a_1 \lor a_2$ can be evaluated as a sum:

$$[e_{or}]^o = [a_1]^o + [a_2]^o,$$

assuming all normalized attribute values a_i represent statistically independent events. Conjunctions like $e_{and} = a_1 \land a_2$ are evaluated as a product:

$$[e_{and}]^o = [a_1]^o \cdot [a_2]^o,$$

analogously. Finally, negation is expressed as

$$[e_{not}]^o = 1 - [a]^o,$$

mirroring the complement of a probability value. This evaluation scheme remains valid even if the terms of the logical expression are themselves non-atomic, as long as they remain statistically independent. This is typically the case when the overall expression is in disjunctive normal form (DNF), where each conjunction can be treated as an independent unit. It should be noted that full distributivity does not hold in general quantum logic. To address this, CQQL expressions are always evaluated in DNF.

Table 2
Example relation with automobile attributes and QLDT evaluation

ID	Safety	Emissions	Label	Score
1	0.9	0.1	1	0.81
2	0.8	0.8	0	0.16
3	0.6	0.2	0	0.48
4	1	0.05	1	0.95
5	0.75	0.15	1	0.6375
6	0.55	0.05	0	0.5225

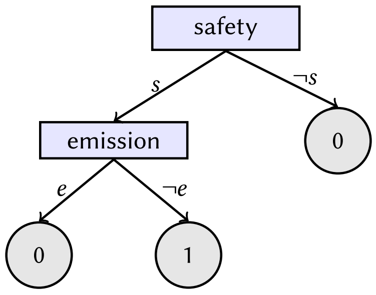

Figure 3: QLDT for emissions example

When trained with the established methods, the CQQL condition of a QLDT is in fact learned in DNF. Like with the DT, it is possible to display this condition as a tree structure. Here, however, we do not require this visualization except for explanations within this article. The evaluation of CQQL conditions generates a score $[e]^o \in [0,1]$ as their output. In order to enable classification, a suitable threshold value τ is chosen, which is used to compare and map the score to $\{0,1\}$.

5. Query Induction with QLDT

With the methodology from Section 3 established, we now adapt it to the quantum logic setting by integrating the Quantum Logic Decision Tree (QLDT) introduced in Section 4. As before, the goal is to induce a query condition from labeled tuples by extracting a logical expression that best separates relevant from irrelevant examples. In this case, however, we aim to learn a CQQL condition instead. This condition can then be used to construct a query, just as in the Boolean case.

A key difference is that standard SQL does not support quantum logic operators, similarity-based predicates, or graded relevance evaluation. To address this, we propose using the Quantum SQL (QSQL) dialect [14], which is specifically designed to support such semantics. Since QSQL is not widely supported in commercial systems, we optionally translate the resulting QSQL query into standard SQL-99. This is possible, as long as the required similarity functions are defined in the database system for the relevant attributes, for example using user-defined functions in a procedural SQL dialect like PL/SQL.

To illustrate this concept, we provide a second example relation and apply the method on this data. Table 2 shows examples from a relation that stores automobile data. Every tuple contains a numerical value for safety and emission level, where a possible consumer would prefer a high safety and a low emission level. Furthermore, the table shows the the label that was provided by a user and a score that we will discuss below.

Training a QLDT classifier on this data yields the decision tree shown in Figure 3. In this figure, the safety attribute is abbreviated as s and the emission attribute as e, purely for compactness of presentation. (This differs from the rest of the paper, where e denotes a condition!)

Table 3
Functions for QSQL translation for the example

Function	Purpose
`SAFE_TO_SCORE(attr, 1)`	Similarity to *safety*
`EM_TO_SCORE(attr, 1)`	Similarity to *emission*
`LAND(x, y)`	Conjunction $x \cdot y$
`LNOT(x)`	Negation $1 - x$

The condition expressed by this tree structure is

$$e = safety \wedge (\neg emission).$$

While it is syntactically equivalent to a Boolean algebra expression, this is a CQQL condition and therefore evaluated accordingly. Therefore, the evaluation formula of this condition is as follows:

$$[e]^o = [safety]^o \cdot (1 - [emission]^o).$$

Keep in mind that for the evaluation of an atomic attribute (e.g. *emission*), the attribute value is taken from the feature vector of the evaluated object o. Incorporating this condition into QSQL could result in the query in Listing 2:

Listing 2: QSQL query expressing the CQQL condition

```sql
SELECT *
FROM automobiles
WHERE (safety ~ 1)
  AND NOT (emission ~ 1 )
ORDER BY scoreval DESC;
```

In QSQL, the operator ~ denotes similarity to a reference value, here 1, evaluated according to CQQL semantics. This is equivalent to the condition that the value "is high" since 1 is the upper limit of its domain. Conjunction and negation are evaluated as described in Section 4.

As discussed earlier, this QSQL code can be converted into normal SQL. For the actual mechanisms of parsing from QSQL to SQL, we refer to [14]. The SQL version of the previous query can be found in Listing 3 below:

Listing 3: SQL translation of the QSQL query

```sql
SELECT *,
  LAND(
    SAFE_TO_SCORE(safety, 1),
    LNOT(EM_TO_SCORE(emission, 1))
  ) AS scoreval
FROM automobiles
ORDER BY scoreval DESC;
```

This, of course, requires the used functions to be defined in the system beforehand. A short list with descriptions of the functions in this example is provided in Table 3. It should be emphasized that in this SQL version, the condition is not placed in the `WHERE` clause but computed in the SELECT clause. This is because its gradual fulfillment is expressed as the attribute `scoreval`, which must be part of the result relation. Thresholding can be added in `WHERE` by referencing `scoreval`, but the intended query includes all tuples and orders them by their degree of fulfillment.

This demonstrates how a QLDT can be used to derive interpretable, graded query conditions in the form of CQQL expressions, which can then be translated into executable QSQL and SQL statements.

6. Conclusion

This work contributes a conceptual step toward bringing example-based query formulation into the setting of quantum logic. Our main outcome is to show that query learning, when framed as a classification problem, can be extended beyond Boolean logic by employing Quantum Logic Decision Trees (QLDT). This allows queries to be induced from labeled tuples in a way that produces interpretable conditions, which can then be executed as Commuting Quantum Query Language (CQQL) expressions and compiled into Quantum SQL (QSQL) or SQL.

Our method generates clean, Boolean algebra-compliant logic expressions, supports fuzzy decision boundaries and gradual relevance without being confined to classical Boolean logic, and inherits the classification performance of QLDTs, which have been shown to perform well in prior studies. As for the limitations of the method, it relies on sufficient and representative labeled data, which may be challenging to obtain from the user.

The generated CQQL conditions require evaluation in QSQL, for which no current implementation exists, or must be mapped manually to SQL-99 with user-defined similarity functions. Additionally, QLDT currently supports only simple similarity cases (conditions like "is high" or "is low") and lacks more flexible or attribute-specific similarity semantics. This could be partially mitigated by replacing the QLDT with a BBQ-Tree, a variant that includes more types of possible splits [15].

This paper is primarily conceptual and does not include an empirical evaluation of user effectiveness. Nonetheless, the underlying components (QBE, DT and QLDT) have been validated individually in prior research. Future work could investigate the practical usability of this method through user studies, system implementation, and comparative benchmarks. Furthermore, strategies such as Active Learning or weak supervision could be explored to reduce the labeling effort required from users.

In summary, we have shown that query induction from labeled examples is feasible even in the context of quantum logic. This contributes a small but necessary step toward supporting example-based query formulation in systems that rely on quantum logic.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT-4 in order to: Grammar and spelling check and preparation of TikZ code for figures. After using this service, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] C. J. van Rijsbergen, The Geometry of Information Retrieval, Cambridge University Press, 2004.

[2] I. Schmitt, QQL: A DB&IR query language, The VLDB Journal 17 (2008) 39–56. doi:10.1007/s00778-007-0070-1.

[3] M. M. Zloof, Query-by-example: A data base language, IBM Systems Journal 16 (1977) 324–343. doi:10.1147/sj.164.0324.

[4] A. Kharade, Generation of sql query using natural language processing, SSRN preprint, 2023. doi:10.2139/ssrn.4515801.

[5] M. Uma, V. Sneha, G. Sneha, J. Bhuvana, B. Bharathi, Formation of SQL from natural language query using NLP, in: 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), 2019, pp. 1–5. doi:10.1109/ICCIDS.2019.8862080.

[6] A. Moreau, O. Pivert, G. Smits, Fuzzy query by example, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 688–695.

[7] S. Demir Sağlam, G. Karadeniz Gözeri, Some properties of boolean-like laws in fuzzy logic, Symmetry 17 (2025). URL: https://www.mdpi.com/2073-8994/17/4/548. doi:10.3390/sym17040548.

[8] H. Chen, G. Shankaranarayanan, L. She, A. Iyer, A machine learning approach to inductive query

by examples: An experiment using relevance feedback, id3, genetic algorithms, and simulated annealing, Journal of the American Society for Information Science 49 (1998) 693–705.

[9] A. Naik, A. Thakkar, A. Stein, R. Alur, M. Naik, Relational query synthesis ⋈ decision tree learning, Proc. VLDB Endow. 17 (2023) 250–263. URL: https://doi.org/10.14778/3626292.3626306. doi:10.14778/3626292.3626306.

[10] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Global Edition, Pearson Education, London, 2021.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, CRC Press, New York, 1984. doi:10.1201/9781315139470.

[12] I. Schmitt, QLDT: A decision tree based on quantum logic, in: S. Chiusano, T. Cerquitelli, R. Wrembel, K. Nørvåg, B. Catania, G. Vargas-Solar, E. Zumpano (Eds.), New Trends in Database and Information Systems, Springer International Publishing, Cham, 2022, pp. 299–308.

[13] I. Schmitt, Logic interpretations of ann partition cells, 2024. URL: https://arxiv.org/abs/2408.14314. arXiv:2408.14314.

[14] S. Lehrack, I. Schmitt, QSQL: Incorporating logic-based retrieval conditions into SQL, in: H. Kitagawa, Y. Ishikawa, Q. Li, C. Watanabe (Eds.), Database Systems for Advanced Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 429–443.

[15] A. Stahl, I. Schmitt, BBQ-Tree – a decision tree with boolean and quantum logic decisions, in: Lecture Notes in Computer Science, volume 14918, 2024.