XPOSE: Bi-directional Engineering for Hidden Query Extraction

Ahana Pradhan Jayant Haritsa

Technical Report
TR-2025-01
(March 2025)

Database Systems Lab
Indian Institute of Science
Bangalore 560012, India

http://dsl.cds.iisc.ac.in

Abstract

Query reverse engineering (QRE) aims to synthesize a SQL query to connect a given database and result instance. A recent variation of QRE is where an additional input, an opaque executable *containing a ground-truth query, is provided, and the goal is to non-invasively extract this specific query through only input-output examples. This variant, called Hidden Query Extraction (HQE), has a spectrum of industrial use-cases including query recovery, database security, and vendor migration.*

The reverse engineering (RE) tools developed for HQE, which are based on database mutation and generation techniques, can only extract flat queries with key-based equi-joins and conjunctive arithmetic filter predicates, making them limited wrt both query structure and query operators. In this paper, we present Xpose*, a HQE solution that elevates the extraction scope to realistic complex queries, such as those found in the TPCH benchmark.*

A two-pronged approach is taken: (1) The existing RE scope is substantially extended to incorporate union *connectors,* algebraic *filter predicates, and* disjunctions *for both values and predicates. (2) The predictive power of LLMs is leveraged to convert business descriptions of the opaque application into extraction guidance, representing "forward engineering" (FE). The FE module recognizes common query constructs, such as* nesting *of sub-queries,* outer joins*, and* scalar functions*. In essence, FE establishes the broad query contours, while RE fleshes out the fine-grained details.*

We have evaluated Xpose *on (a) E-TPCH, a query suite comprising the complete TPCH benchmark extended with queries featuring unions, diverse join types, and sub-queries; and (b) the real-world STACK benchmark. The experimental results demonstrate that its bi-directional engineering approach accurately extracts these complex queries, representing a significant step forward with regard to HQE coverage.*

Contents

1 Introduction 5
2 Mutation Framework of Unmasque [17] 6
2.1 From Clause Extractor 7
2.2 Database Minimizer 7
2.3 Arithmetic Filter Predicate Extractor 7
2.4 Satisfying Values 7
3 Design Overview of Xpose 7
3.1 XRE and XFE: Who does What? 7
3.2 Running Example of Hidden Query 8
3.3 Mutation Framework in XRE 8
3.4 LLM-based Synthesis Framework in XFE 9
4 UNION ALL Extraction 9
4.1 Problem Formulation 9
4.2 Assumptions 9
4.3 Extraction Process Overview 9
4.4 Performance Impacts and the measures 10
4.5 Proof of Correctness 11
5 Algebraic Predicates 11
5.1 Problem Formulation 11
5.2 Extraction Process Overview 12
5.3 Equality (=) Predicate Extraction 12
5.4 Inequality (≤, ≥) Predicate Extraction 13
5.5 Inequality (<, >) Predicate Extraction 14
5.6 Outer Join Predicates 14
5.7 Semi-join Semantics 14
5.8 Generalized Linear Inequality Predicates 14
5.9 Proof of Correctness 14
6 Logical Disjunction in Predicates 15
6.1 Extraction Algorithm 15
6.2 Example 15
7 Forward Engineering in Xpose 15
7.1 Synthesis Scope 15
7.2 Basic Synthesis Prompt 16
7.3 Error Feedback Prompts 17
7.4 Hash-based Result Comparator 17
7.5 Combinatorial Synthesis of Nested Clauses 18
7.6 Equivalence Verification 19
8 Experimental Results 19
8.1 TPCH/ART Extraction 20
8.2 E-TPCH Extraction 20
8.3 Drill-down Analysis (E-TPCH) 21
8.4 STACK Extraction 22
8.5 OL-TPCDS Extraction 22
9 Related Work 22
10 The Road Ahead 23
11 Conclusions 23
References 23
A E-TPCH Queries 25
A.1 E-TPCH Q1 25
A.2 E-TPCH Q2 27
A.3 E-TPCH Q3 28
A.4 E-TPCH Q4 31
A.5 E-TPCH Q5 32
A.6 E-TPCH Q6 34
A.7 E-TPCH Q7 35

A.8 E-TPCH Q8 37
A.9 E-TPCH Q9 38
A.10 E-TPCH Q10 40
A.11 E-TPCH Q11 42
A.12 E-TPCH Q12 43
A.13 E-TPCH Q13 45
A.14 E-TPCH Q14 46
A.15 E-TPCH Q15 47
A.16 E-TPCH Q16 49
A.17 E-TPCH Q17 50
A.18 E-TPCH Q18 50
A.19 E-TPCH Q19 51
A.20 E-TPCH Q20 54
A.21 E-TPCH Q21 56
A.22 E-TPCH Q22 57
A.23 E-TPCH Q23 59
A.24 E-TPCH Q24 60
B STACK Queries 62
B.1 STACK Q7 62
B.2 STACK Q8 62
B.3 STACK Q9 63
B.4 STACK Q10 63

1 Introduction

Query Reverse Engineering (QRE) has been a subject of considerable recent interest in the database community. Here, the standard database equation, namely $Q(D_I) = \mathcal{R}$, where Q is a declarative query, D_I is a database instance, and $\mathcal{R}$ is the result of executing Q on D_I – is inverted. Specifically, the inputs are now D_I and $\mathcal{R}$, and the goal is to identify a candidate SQL query Q_c that satisfies the equation. QRE has a variety of industrial applications, including recreating lost application code and assisting SQL amateurs to formulate complex queries. To meet these objectives, a host of sophisticated software tools (e.g. TALOS [34], Scythe [35], FastQRE [15], REGAL [30, 31], SQUARES [24], PATSQL [29], SICKLE [40], CUBES [4]) have been developed over the past decade.

Hidden Query Extraction

A variant of QRE, called Hidden Query Extraction (HQE), was introduced in Sigmod 2021 [17]. Here, in addition to the generic QRE inputs, D_I and $\mathcal{R}$, a ground-truth query is available, but in a hidden or inaccessible form. For instance, the query may have been encrypted or obfuscated. The goal now is to non-invasively extract the hidden query through "active learning", that is, by observing a series of *input-output examples* produced by this executable on a variety of curated databases. [1]

HQE has several use-cases, including application logic recovery, database security, vendor migration, and query rewriting. For instance, with legacy industrial applications, the original source code is often lost with passage of time [25, 26] (particularly with organizational mergers or outsourcing of projects). However, to understand the application output, we may need to establish the logic connecting the database input to the observed result. Moreover, we may wish to extend or modify the existing application query, and create a new version.

Formally, the HQE problem is defined as: *Given a black-box application $\mathcal{A}$ containing a hidden SQL query $Q_{\mathcal{H}}$, and a sample database instance D_I on which $\mathcal{A}$ produces a populated result $\mathcal{R}_{\mathcal{H}}$ (i.e. $Q_{\mathcal{H}}$ $(D_I) = \mathcal{R}_{\mathcal{H}}$), determine the precise $Q_{\mathcal{H}}$ contained in $\mathcal{A}$.*

At first glance, HQE may appear more tractable than general QRE since the executable could be leveraged to prune the candidate search space. But, conversely, the correctness requirements are *much stricter* since a specific ground-truth query, and not a candidate solution, is expected as the output. So, while the problems are related, their solution frameworks are quite different.

Unmasque *Extractor.* We took a first step towards addressing the HQE problem in [17], by introducing the **UNMASQUE** tool, which is capable of (non-invasively) extracting flat **SPJGAOL** (Select, Project, Join, Group By, Agg, Order By, Limit) queries. The extraction operates in the linear pipeline shown in Figure 1. Here, a clause-by-clause extraction of the hidden query is implemented, starting with From and ending with Limit. This is achieved by analyzing the results of targeted $Q_{\mathcal{H}}$ executions on databases derived from D_I through *database mutation* and *database generation* techniques. To reduce extraction overheads, the pipeline begins by minimizing the sample input D_I to a handful of rows.

[1]This problem is akin to the classical "Chosen Plain-text Attack" in cryptography [3].

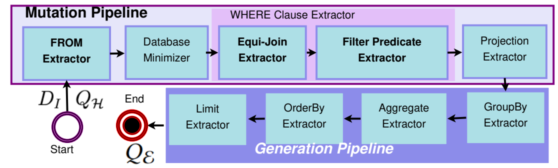


Figure 1: Linear Extraction Pipeline of Unmasque [16, 17]

Bi-directional Engineering

A natural question at this point is whether the extraction scope of Unmasque could be extended to "industrial-strength" queries. For instance, as a tangible milestone, is it feasible to extract the TPCH benchmark queries, most of which go well beyond flat SPJGAOL? Our analysis shows that, by moving from a linear extraction pipeline to a *reentrant* pipeline with looping across modules, the RE-based techniques can be extended to include complex operators. In particular, Unions and Algebraic Predicates (*column op column*) are described later in this paper. But, even so, we also found them to be fundamentally incapable of extracting common query constructs, such as nestings (in From and Where clauses) and scalar functions (e.g. `substring()`).

The above limitations appear to preclude the extraction of realistic queries. However, this negative outcome can be remedied if we assume that with the executable, a high-level *textual description*, TX_Q, of the business logic embedded in the query is also available. Such information is usually present in application design specifications, and therefore likely to be accessible. In fact, industries subject to compliance regulations (e.g. banking, insurance, healthcare) are *mandated* to possess documented business logic for audit purposes.

We can now bring in the remarkable predictive power of LLMs to leverage TX_Q into extraction guidance – that is, "forward engineering" (FE), akin to Text-to-SQL tools [1], specifically for features such as query nesting, outer joins, and scalar functions. And what we show in this paper is that a judicious synergy of forward and reverse engineering is capable of precisely extracting the *entire TPCH benchmark* and more. Moreover, this coverage could *not* have been achieved by either of them in isolation. In a nutshell, FE establishes the broad contours of the query, while RE fleshes out the fine-grained details in the SQL clauses.

The Xpose System

Our new bidirectional approach is implemented in a tool called Xpose, depicted in Figure 2. The inputs are the textual description TX_Q, the opaque executable $\mathcal{A}$ containing $Q_{\mathcal{H}}$, and the sample database D_I. The extraction process starts with the **XRE** *(*Xpose-*RE)* module, which uses $\mathcal{A}$ and a variety of databases derived from D_I to reverse-engineer a *seed query*, $Q_{\mathcal{S}}$. Then $Q_{\mathcal{S}}$ is sent to the **XFE** *(*Xpose-*FE)* module, which contrasts it against TX_Q, and iteratively refines it, using feedback prompting techniques, into $Q_{\mathcal{E}}$, a semantically equivalent version of $Q_{\mathcal{H}}$ through query synthesis. The refinement iterates until one of the following occurs: (1) the synthesized query matches $Q_{\mathcal{H}}$ wrt the execution results on D_I; (2) no new formulations are synthesized; or (3) a threshold number of iterations is exceeded. If the LLM stops due to the latter two cases, which represent failure, we move on to a *combinatorial synthesizer* that searches through all valid combinations of tables and groupings within the nesting structure constructed by the LLM. Finally, although fundamentally limited due to non-availability of the hidden query, we try various tests to check the accuracy of the extraction: First, for queries within its scope, the XData [5] tool is used to generate test databases that "kill mutants" – that is, produce different results if there are discrepancies between the extracted query $Q_{\mathcal{E}}$ and the black-box $Q_{\mathcal{H}}$. Second,the result-equality of $Q_{\mathcal{E}}$ and $Q_{\mathcal{H}}$ is evaluated on multiple randomized databases to probabilistically minimize the likelihood of false positives.

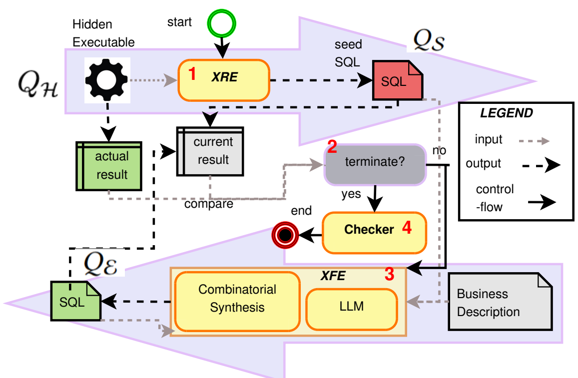


Figure 2: Architecture of XPOSE

Illustrative Extraction

To illustrate the above process, consider TPCH **Q20** shown in Figure 3(a) with encryption. The business description [32] of this so-called "Potential Part Promotion Query", is in Figure 3(b).

The XRE output of XPOSE with the Q20 executable is shown in Figure 3(c). We see here that it correctly extracts the five base tables (LINEITEM, NATION, PART, PARTSUPP, SUPPLIER), equi-joins among these tables, filter predicates (`l_shipdate`, `p_name`, `n_name`), projection columns, and result ordering. On the downside, however, there are several errors (highlighted in red) – for instance, filters on `l_quantity` and `ps_availqty` are *spuriously* reported due to the nested subquery in the WHERE clause, which goes beyond the flat structures covered by XRE. Similarly, the joins within subqueries are incorrectly characterized as global joins. Lastly, semi-joins, constructed using subqueries, are extracted as equi-joins.

The XFE component is now prompted with (i) the business description TX_Q, (ii) the XRE output as a seed query Q_S, and (iii) a set of canonical correction instructions listed in Figure 3(d). These instructions are generic guidelines, based on SQL experience and XRE limitations, to help the LLM synthesize queries that are aligned with Q_S and TX_Q. As a simple instance, the LLM's schematic scope is explicitly restricted to the tables in Q_S since they are known to have been identified correctly. Ensuring compliance with this suite of instructions results in correction of all previously noted errors, as shown in the final extracted query, $Q_{\mathcal{E}}$, listed in Figure 3(e) (the respective roles of XRE and XFE are highlighted in blue and purple colors). Finally, the checker module is invoked to test whether, modulo syntactic differences, $Q_{\mathcal{E}}$ is functionally equivalent to $Q_{\mathcal{H}}$– for Q20, both the XData tool and the result-equivalence tests do not find any discrepancy.

Contributions

In this paper, our contributions are with regard to both forward and reverse engineering:

- Constructing RE algorithms to extract unions, algebraic predicates, and predicate disjunctions, by generalizing the linear pipeline of UNMASQUE to a reentrant directed graph.
- FE using an LLM on a seed query augmented with automated corrective feedback prompts. The prompts are designed to rectify the errors (of both commission and omission) introduced by RE.
- A novel amalgamation of FE and RE to extract complex SQLs, implemented in XPOSE.
- Evaluation of XPOSE on (a) E-TPCH, a query suite featuring the complete TPCH benchmark extended with queries capturing diverse join types and unions of sub-queries; and (b) STACK benchmark queries featuring several joins and disjunction predicates.

Organization

The mutation framework of UNMASQUE is reviewed in Section 2. The high-level design of XPOSE is described in Section 3, followed by the building blocks of XRE and XFE. Detailed XRE extraction strategies for union, algebraic predicates, and disjunction are presented in Sections 4, 5, and 6, respectively. The use of XFE to synthesize from the seed query is detailed in Section 7. The experimental framework and results are presented in Section 8. Our future research avenues are discussed in Section 10, and related work is reviewed in Section 9. Finally, our conclusions are summarized in Section 11.

2 Mutation Framework of UNMASQUE [17]

We review here a few core concepts of the mutation framework employed in UNMASQUE. These concepts are augmented in XPOSE, as described in Section 3. The notations used here and in the rest of the paper are listed in Table 1.

To extract the SPJ-elements (of flat SPJGAOL queries), UNMASQUE employs targeted data updates at different granularities (relation or attribute) to expose clause-specific unique signatures. The underlying assumptions are: (1) The database D_I is NULL-free; (2) $Q_{\mathcal{H}}$ produces a populated result on D_I; (3) $Q_{\mathcal{H}}$ is a conjunctive query.

Figure 4 lists one such SPJ-query.

Symbol	*Meaning*	*Symbol*	*Meaning*
D_I	Initial Database	D^1	Single Row Database
$Q_{\mathcal{H}}$	Hidden Query	$\mathcal{R}_{\mathcal{H}}$	Output of $Q_{\mathcal{H}}$
Q_S	Extracted query by XRE	$\mathcal{R}_S$	Output of Q_S
$Q_{\mathcal{E}}$	Extracted query by XPOSE	$\mathcal{R}_{\mathcal{E}}$	Output of $Q_{\mathcal{E}}$
i_{min}	Min Domain Value	i_{max}	Max Domain Value
SVE	S-value Extractor	GS_{ve}	Group S-value Extractor
T_E	Set of tables in $Q_{\mathcal{H}}$	SVI	SVE output
σ_p	Algebraic Predicates	C_E	Attributes in SVI
$SVI.col.UB$	col's Max s-value	$SVI.col.LB$	col's Min s-value

Table 1: Notations

(a) **Hidden Query:** $Q_{\mathcal{H}}$ **[Encrypted TPCH Q20]**
CREATE PROCEDURE hQ With Encryption BEGIN select s_name, s_address from supplier, nation where s_suppkey in (select ps_suppkey from partsupp where ps_partkey in (select p_partkey from part where p_name like '%ivory%') and ps_availqty > (select 0.5 * sum(l_quantity) from lineitem where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_shipdate >= date '1995-01-01' and l_shipdate < date '1995-01-01' + interval '1' year)) and s_nationkey = n_nationkey and n_name = 'FRANCE' order by s_name **END**;

(b) **Business Description of Q20:** TX_Q
The query identifies suppliers who have an excess of a given part available; an excess is defined to be more than 50% of the parts like the given part that the supplier shipped in the year 1995 for FRANCE. Only parts whose names share the word 'ivory' are considered.

(c) **Output of XRE (Seed Query):** Q_S
Select s_name, s_address From lineitem, nation, part, partsupp, supplier Where l_partkey = p_partkey and p_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_suppkey = s_suppkey and n_nationkey = s_nationkey and n_name = 'FRANCE' and l_quantity <= 9687.99 and l_shipdate between '1995-01-01' and '1995-12-31' and p_name LIKE '%ivory%' and ps_availqty >= 12 Order By s_name asc;

(d) Feedback Prompts to LLM
(1) Only use the tables in the seed query. (2) Validate all the filter predicates in the seed query against the text. (3) Use the valid filter predicates present in the seed query. (4) For the attributes in the invalid filter predicates, validate their use from the business description text. Then formulate predicates with them. (5) Strictly follow the projection order and projection dependencies used in the seed query. (6) A semi-join, implying at least one match, may be incorrectly present as an equi-join in the seed query.

(e) Final Output of Xpose (XRE + XFE): $Q_{\mathcal{E}}$
Select s_name, s_address From supplier Where s_suppkey IN (Select ps_suppkey From partsupp Where ps_availqty > 0.5 * (Select Sum(l_quantity) From lineitem Where l_shipdate BETWEEN '1995-01-01' AND '1995-12-31' and l_partkey = ps_partkey and l_suppkey = ps_suppkey) and ps_partkey IN (Select p_partkey From part Where p_name LIKE '%ivory%')) and s_nationkey = (Select n_nationkey From nation Where n_name = 'FRANCE') Order By s_name ASC;

Figure 3: Extracting TPCH Q20 using Bidirectional Engineering

q_0: Select c_name AS name, c_phone as phone
From customer, orders Where c_custkey = o_custkey AND c_acctbal < 10000

Figure 4: Sample Hidden Query

(a) Minimized Database D^1

Table: D^1.Orders

o_custkey	...
23074	...

Table: D^1.Customer

c_acctbal	*c_custkey*	...
774.84	23074	...

(b) Populated Result Set $\mathcal{R}_{\mathcal{H}}$ obtained by $Q_{\mathcal{H}}$ on D^1:

name	*phone*
Customer#000023074	18-636-637-7498

Figure 5: Showcasing D^1 for q_0 in Figure 5

2.1 From Clause Extractor

The From clause extractor enumerates T_E, the set of tables present in $Q_{\mathcal{H}}$. For this purpose, a simple **Extraction by Error (EbE)** check is sequentially carried out over all tables in the database schema. Specifically, a table is present if renaming it to a dummy name triggers an error from the database engine when parsing $Q_{\mathcal{H}}$. The original database schema is restored after each check is completed. For the $Q_{\mathcal{H}}$ listed in Figure 4, T_E = {Customer, Orders} is determined by the EbE strategy.

An alternative check **Extraction by Voiding (EbV)** states a table is present if its "void" version (i.e. table having no data) causes $Q_{\mathcal{H}}$ output to be empty. This check was intended to be used in Unmasque only for certain special cases where Ebe did not apply, but is routinely invoked in Xpose for extracting Unions.

2.2 Database Minimizer

To avoid long extraction times due to repeated executions of $Q_{\mathcal{H}}$ on a large D_I, the notion of a minimized database D_{min} was introduced in Unmasque. Specifically, a database is said to be minimal if removing a row from any table leads to an empty result. That is, it is the minimal database that provides a populated output, and can be efficiently identified using a recursive halving process [17]. Interestingly, it was proved in [17] that under the aforementioned assumptions, there always exists a D_{min} wherein each table in T_E contains only a *single row*. Such a database is referred to as $\mathbf{D^1}$. In Figure 5(a) and (b), D^1 for the q_0 in Figure 4, and the corresponding result set is shown respectively.

2.3 Arithmetic Filter Predicate Extractor

Arithmetic filter predicates of the form *column op value*, $op \in \{<, \leq, \geq, >\}$, are extracted with the following process: For each attribute of the tables in T_E, a binary search-based mutation on D^1 is carried out over its domain. For instance, let $D^1.col_x = v$, and our goal is to find whether predicate $l \leq col_x \leq r$ exists on attribute col_x. D^1 is first mutated with $col_x = i_{min}$, and $Q_{\mathcal{H}}$ is executed. A populated result implies $i_{min} \leq col_x \leq v$ satisfies the predicate, i.e. $l = i_{min}$. Otherwise, $i_{min} < l < col_x \leq v$. Next, the interval $[i_{min}, v]$ is searched by mutating col_x with the mid-value $(i_{min}+v)/2$, to check whether $Q_{\mathcal{H}}$ produces a populated result. Thus the binary search is used to find the smallest value l in $[i_{min}, v]$ interval satisfying $Q_{\mathcal{H}}$. Similarly, the interval $[v, i_{max}]$ is handled to extract r, the *largest* value in the interval satisfying $Q_{\mathcal{H}}$. From the D^1 in Figure 5(a) for q_0, the s-value interval $[i_{min}, 9999.99]$ for *c_acctbal* extracted in this manner thus infers the predicate *c_acctbal* ≤ 9999.99.

2.4 Satisfying Values

The term "Satisfying values", or *s-values*, was introduced to refer, for each attribute, the range of its values that satisfy $Q_{\mathcal{H}}$ and contribute towards producing a populated result. The predicate filter values determine the s-values for the corresponding attribute. For instance, the s-values for *c_acctbal* is the closed interval $[i_{min}, 9999.99]$.

3 Design Overview of Xpose

In this section, we provide an overview of the Xpose design. We begin with the distribution of work across the XRE and XFE modules, then follow up with how the mutation concepts of Unmasque (Section 2) are adapted in XRE to meet the significantly extended extraction scope requirements of Xpose, and conclude with the synthesis framework of XFE.

3.1 XRE and XFE: Who does What?

An interesting design question is the division of extraction labor between XRE and XFE. For instance, since we already know that

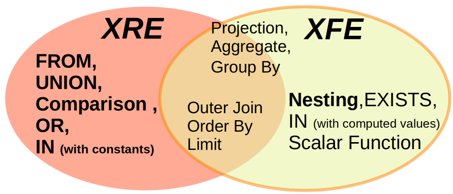


Figure 6: Extraction Scopes of XRE and XFE

XRE has some inherent limits, can we go to the other extreme and carry out most or all of the extraction using XFE? The answer is no, we need a *combination* of XRE and XFE that play to their respective strengths, to achieve precise extraction of complex queries. Specifically, XRE is deterministic and provable, whereas XFE is iterative and probabilistic, requiring XRE as an anchor.

As a case in point, in Figure 3, if we employ XFE to directly carry out the synthesis from only TX_Q, several errors are made – a spurious introduction of the REGION table in the outer query, creation of two instances of LINEITEM, applying temporal predicate on `o_orderdate` instead of `l_shipdate`, using wrong attribute `l_quantity` in the aggregate comparison, etc. However, armed with the XRE-generated seed query of Figure 3(c), XFE is able to restrict its formulation to the precise set of tables and attributes, and converge the synthesis to $Q_{\mathcal{H}}$.

Our division of labor between the XRE and XFE components is shown in Figure 6. Only XRE can guarantee correct extractions of operators performing core associations among the data items, such as UNION, FROM, equality predicates (including JOINS), and disjunctive (OR and IN operator with query constant values) predicates. On the other hand, nested structures, membership operators (such as IN operator with query computed values), and scalar functions can only be handled by XFE. Whereas operators such as SELECT, AGG, and GROUP BY require a *combined* effort – for instance, in projection functions, XRE identifies the variables (attributes) and XFE formulates the function (such as polynomial and conditional computations) linking these variables. Finally, the comparison operators, OUTER JOIN, ORDER BY and LIMIT clauses, can all be extracted by *either* XRE or XFE. For these operators, the advantage of XRE is provable correctness whereas that of XFE is performance benefits (due to immediate derivation from TX_Q). In our implementation, we chose to have comparisons handled by XRE (because JOINS and comparisons are handled uniformly by our algebraic predicate extractor), whereas the remainders are delegated to XFE.

3.2 Running Example of Hidden Query

To explain the working of XPOSE, hereafter we use the hidden query shown, along with its business description, in Figure 7a. This query has two sub-queries, q_1 and q_2, and is designed to highlight the major new extraction extensions provided by XPOSE – nested structures (nested select in q_2), unions of sub-queries (UNION ALL of q_1 and q_2), algebraic predicates (`s_acctbal` $\leq$ `o_totalprice` in q_2), disjunctions (IN in q_2), and outer joins (in q_1).

Business Description TX_Q: List the names and phone numbers of customers and suppliers with low balance. For customers having orders, low balance is determined by a constant threshold whereas for suppliers, the threshold is determined by any order shipped using specified transport modes. Customers having no orders are also included in this list, irrespective of their balance.

$Q_{\mathcal{H}}$:
```
SELECT * FROM (
q1   ( SELECT c_name AS name, c_phone AS phone
         FROM CUSTOMER LEFT OUTER JOIN ORDERS
              ON c_custkey = o_custkey
              WHERE c_acctbal <= 10000 OR o_orderkey IS NULL )
     UNION ALL
q2   ( SELECT s_name AS name, s_phone AS phone FROM supplier
         WHERE s_suppkey IN ( SELECT l_suppkey FROM orders, lineitem
           WHERE l_orderkey = o_orderkey AND s_acctbal <= o_totalprice
           AND l_commitdate = l_receiptdate
           AND l_shipmode IN ('AIR', 'TRUCK')) )
  ) AS people GROUP BY name, phone
```

(a) Hidden Query $Q_{\mathcal{H}}$ with Business Description

$Q_{\mathcal{S}}$:
```
( SELECT c_name AS name, c_phone as phone
      FROM customer, orders WHERE c_custkey = o_custkey
      AND c_acctbal <= 10000.00 OR o_orderkey IS NULL
      GROUP BY c_name, c_phone )
UNION ALL
   ( SELECT s_name AS name, s_phone as phone
      FROM lineitem, orders, supplier WHERE l_orderkey = o_orderkey
      AND s_suppkey = l_suppkey AND s_acctbal <= o_totalprice
      AND l_commitdate = l_receiptdate
      AND l_shipmode IN ('AIR', 'TRUCK')
   GROUP BY s_name, s_phone )
```

(b) Seed Query $Q_{\mathcal{S}}$ (output by XRE)

$Q_{\mathcal{E}}$:
```
SELECT name, phone FROM (
  (SELECT c_name AS name, c_phone AS phone
    FROM customer c LEFT JOIN orders o ON c.c_custkey = o.o_custkey
    WHERE (c.c_acctbal <= 10000.00 OR o.o_orderkey IS NULL))
UNION ALL
  (SELECT s_name AS name, s_phone AS phone
    FROM supplier s WHERE s.s_suppkey IN (
        SELECT l_suppkey FROM lineitem l JOIN orders o
        ON l.l_orderkey = o.o_orderkey WHERE s.s_acctbal <= o.o_totalprice
    AND l.l_commitdate = l.l_receiptdate AND l.l_shipmode IN ('AIR','TRUCK')))
) as customer_supplier GROUP BY name, phone
```

(c) Final Synthesized Query $Q_{\mathcal{E}}$

Figure 7: Example Hidden Query Extraction with XPOSE

3.3 Mutation Framework in XRE

We now explain how the mutation framework outlined in Section 2 is adapted to support the extended scope of XPOSE.

3.3.1 From Clause Extractor. The EbE (Extraction-by-Error) strategy is sufficient for extracting $T_{\mathcal{H}}$, the set of tables in $Q_{\mathcal{H}}$. However, for Unions, we need this information at the granularity of *sub-queries*. This requires bringing the **EbV (Extraction-by-Voiding)** strategy also into play, as described later in Section 4.

3.3.2 FIT-results. In XPOSE, we consider the possibility of $Q_{\mathcal{H}}$ having outer joins. Due to their presence, attributes in the result set $\mathcal{R}_{\mathcal{H}}$ may have NULL values even if the original input database D_I is completely NULL-free. This has repercussions for our extraction procedure – for instance, the binary search for arithmetic predicates is no longer viable. Therefore, we need to make the definition of a populated result more nuanced than simply checking for the presence of tuples in the output.

Specifically, we introduce the notion of a **fully instantiated tuple (FIT)**, a tuple not containing any NULL values. With this notion, a FIT-result is a query result that features *at least one* FIT row, whereas a UNFIT-result has no such rows. A FIT result implies that the input database could provide one or more pairs of matching tuples, and such tuples become candidates for the join predicate extraction (Section **??**). Consequently, for $Q_{\mathcal{H}}$ to be extractable, $\mathcal{R}_{\mathcal{H}}$ must be a FIT-result – our modified minimizer and the filter predicate extractor ensure retention of this characteristic in the recursive halving and binary search procedure, respectively.

3.3.3 S-Value Extractor. When $Q_{\mathcal{H}}$ is restricted to arithmetic predicates, the s-value bounds correspond, as discussed in Section 2.4, to the predicate constants in the query, i.e. they are *static*. However, this is no longer the case if $Q_{\mathcal{H}}$ features algebraic predicates such as $col_x \le col_y$. Consider $D^1.col_x = X$, for some $X \in [i_{min}, i_{max}]$. Then, col_y has s-value interval $[X, i_{max}]$, a *floating interval* due to its dependency on X. Such floating dependencies are identified, as detailed in Section 5, by mutating attribute values in D_{min} and *iteratively* applying the static s-value extractor used for arithmetic predicates. We denote this modified s-value extractor by **SVE**.

3.3.4 Reentrant Pipeline. The XRE module features a generalized mutation pipeline, with multiple outgoing edges from a node, as well as looping among the extraction modules. This re-entrant structure facilitates progressive construction of $Q_{\mathcal{S}}$, the seed query.

3.3.5 Seed Query Output. With the above augmented mutation framework, the $Q_{\mathcal{S}}$ output by XRE is shown in Figure 7b. It captures the Union All, From and Where clause predicates correctly.

3.4 LLM-based Synthesis Framework in XFE

Since XRE's extraction technique is based on the single-row D^1, it only extracts a minimal conjunctive flat query in $Q_{\mathcal{S}}$. We now need to add the nested structure, outer join, and disjunctive IN operator, and pull up the common grouping attributes from the sub-queries to the outer query.

XFE infers the nested structure using the LLM on the business description TX_Q. For instance, the phrase "any order" in TX_Q (Figure 7a) maps to the IN operator, and the nested structure is inferred from it. As elaborated later in Section 7, XFE has a set of guidelines for such query synthesis through automated iterative prompting that ensures eventual convergence. An important subset of the guidelines instructs the LLM to remain aligned to $Q_{\mathcal{S}}$ while performing its synthesis. For instance, in the above example, retaining the join, comparison and disjunctive predicates is crucial. Also, specific guidelines instruct the LLM on how to handle result mismatches between $Q_{\mathcal{S}}$ and $Q_{\mathcal{H}}$ on D_I. For example, moving the Group By attributes to the outer query is driven by the guidelines.

Finally, $Q_{\mathcal{E}}$, the extracted query post-XFE intervention, is shown in Figure 7c, and it is clearly semantically identical to $Q_{\mathcal{H}}$.

In the following sections, we cover the internals of the Union, Algebraic Predicate, and Disjunction extractors.

4 UNION ALL Extraction

The key step of union extraction is to isolate each subquery in terms of the tables in their respective From clauses. Now we discuss how a fine-grained **EbV** is devised to isolate the subqueries.

4.1 Problem Formulation

A hidden union query $Q_{\mathcal{H}}$ is a compound union of an unknown number of SPJGAOL subqueries. Formally, $Q_{\mathcal{H}} = \bigcup_{i=1}^{n} q_i$ for an unknown n. Let $\texttt{FROM}(q_i)$ denote the set of tables present in subquery q_i. Thus, $T_{\mathcal{H}} = \bigcup_{i=1}^{n} \texttt{FROM}(q_i)$. The tables that appear in *all the subqueries* are referred to as the *common tables*, set $\texttt{COMMON}(Q_{\mathcal{H}})$. Therefore, set difference $T_{\mathcal{H}} - \texttt{COMMON}(Q_{\mathcal{H}})$ obtains the *set of tables that appear in some of the subqueries in $Q_{\mathcal{H}}$, but not in all.* These tables are the *auxiliary tables*, set $\texttt{AuxTables}(Q_{\mathcal{H}})$. Note that the auxiliary tables of subquery q_i, i.e. $\texttt{AuxTables}(q_i)$ are obtained as $\texttt{FROM}(q_i) - \texttt{COMMON}(Q_{\mathcal{H}})$, and $\texttt{AuxTables}(q_i) \subseteq \texttt{AuxTables}(Q_{\mathcal{H}})$. Therefore, to uniquely extract $\texttt{FROM}(q_i)$, sets $\texttt{AuxTables}(q_i)$ and $\texttt{COMMON}(Q_{\mathcal{H}})$ need to be determined.

- ***<u>Extraction Goals</u>***: for a given union query $Q_{\mathcal{H}}$,

(1) Compute the set of common tables $\texttt{COMMON}(Q_{\mathcal{H}})$.
(2) Isolate each subquery by partitioning the set of auxiliary tables
– each partition corresponds to a unique subquery:
$\forall i, 0 \le i \le n$, $\texttt{AuxTables}(q_i) \cup \texttt{COMMON}(Q_{\mathcal{H}}) = \texttt{FROM}(q_i)$.
– all the partitions together cover the set $\texttt{AuxTables}(Q_{\mathcal{H}})$:
$\texttt{AuxTables}(Q_{\mathcal{H}}) = \bigcup_{i=1}^{n} \texttt{AuxTables}(q_i)$.
(3) For each subquery q_i, extract the unique SPJGAOL clauses.

4.2 Assumptions

(1) Each subquery q_i produces a FIT-result on D_I.
(2) The sets of tables in any two subqueries are not subsets of each other. This assumption is satisfied usually in practice (16 out of 18 union queries in the TPC-DS [33] benchmark), such as in data integration queries (union over disjoint fact tables from different locations).

4.3 Extraction Process Overview

4.3.1 Compute Common Tables. The set of all tables in $Q_{\mathcal{H}}$, $T_{\mathcal{H}}$ is extracted using the EbE (Section 2.1). $\texttt{COMMON}(Q_{\mathcal{H}})$ is identified using EbV (Section 2.1): A table T is made *void*, and then $Q_{\mathcal{H}}$ is executed. An UNFIT-result implies that none of the subqueries produced a FIT-result. A union of conjunctive queries can produce an UNFIT-result due to a void table T only if the absence of data in T prevents *every* subquery from obtaining *any satisfying tuple*. Therefore, T is present in all subqueries. The common tables are identified by iterating this technique over all tables in the schema.

4.3.2 Compute Auxiliary Tables (q_i). $\texttt{AuxTables}(Q_{\mathcal{H}})$ is given by $T_{\mathcal{H}} - \texttt{COMMON}(Q_{\mathcal{H}})$. Our goal now is to choose subsets from $\texttt{AuxTables}(Q_{\mathcal{H}})$ so that each subset corresponds to a subquery of $Q_{\mathcal{H}}$, ensuring that every subquery has a matching subset. For this, we enumerate the power set of $\texttt{AuxTables}(Q_{\mathcal{H}})$ and test each member (barring the null set and the entire set) with regard to whether it maps to a *single subquery*. The test is discussed next, capturing its logic in Algorithm 1.

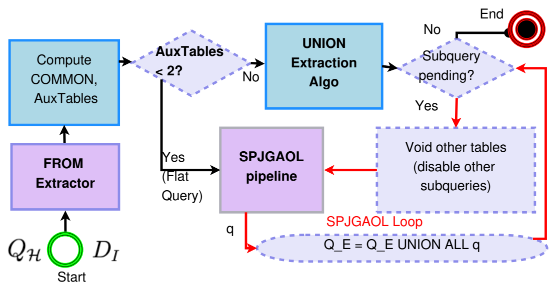


Figure 8: Flow of Union Detection and Extraction

$Q_{\mathcal{H}}$ = (SELECT ... FROM *orders*, *customer* WHERE ...) ∪ (SELECT ... FROM *supplier* WHERE ... (SELECT ... FROM *lineitem*, *orders* WHERE ...))	
$T_{\mathcal{H}}$ = {ORDERS(o), SUPPLIER(s), CUSTOMER(c), LINEITEM(l)}	
COMMON($Q_{\mathcal{H}}$) = $\{o\}$	AuxTables($Q_{\mathcal{H}}$) = $\{s, c, l\}$
$U = (\{s, c\}, \{s, l\}, \{c, l\}, \{s\}, \{c\}, \{l\})$	
CoreTables = $\{\{s, l, c\}, \{s, c\}, \{l, c\}\}$	SideTables = $\{\{s, l\}, \{s\}, \{c\}, \{l\}\}$
Max-SideTables = $\{\{s, l\}, \{c\}\}$	Aux = $\{\{c\}, \{s, l\}\}$
Froms = $\{\{c, o\}, \{s, l, o\}\}$	
FROM(q_1) = {CUSTOMER, ORDERS}, FROM(q_2) = {SUPPLIER, LINEITEM, ORDERS}	

Table 2: Partitioning (Algorithm 1) for $Q_{\mathcal{H}}$ in Figure 7a

Algorithm 1: Union Extraction Algorithm: *Partitioning the Set of Auxiliary Tables*

```
Input    : Q_H, D_I, AuxTables(Q_H), COMMON(Q_H)
1  CoreTables ← AuxTables(Q_H)
2  SideTables, Max-SideTables, Aux, Froms ← Empty sets
3  U ← PowerSet(AuxTables(Q_H)) − {∅, AuxTables(Q_H)}
     // exclude ⊤ and ⊥ from the lattice
4  for u ∈ U do
5      if a subset of u is already in CoreTables then
6          include u in CoreTables
7          continue // skip checking for u
8      if a superset of u is already in SideTables then
9          include u in SideTables
10         continue // skip checking for u
11     D′ ← void all tables in u in D_I
12     if Q_H (D′) is UNFIT then include u in CoreTables
13     else include u in SideTables
14     Revert Mutations done by Line 11
15 Max-SideTables ← {c|c ∈ SideTables, ∀t ∈ AuxTables(Q_H) -
     {c}, c ∪ {t} ∈ CoreTables}
16 for each c ∈ Max-SideTables do
17     include set AuxTables(Q_H) − c in Aux
18 for each p ∈ Aux do
19     from_i ← p ∪ COMMON(Q_H)
20     include from_i in Froms
21 return Froms
```

4.3.3 Union Extraction: Partitioning Algorithm. With the common tables intact, a void auxiliary table fails to satisfy the related predicates in the corresponding subquery. Thus, AuxTables(q_i) is a set of tables from AuxTables($Q_{\mathcal{H}}$), any one of which when voided, gets q_i to produce a UNFIT-result. To identify such sets, we consider the power set of AuxTables($Q_{\mathcal{H}}$), and observe the result of $Q_{\mathcal{H}}$ upon voiding the tables from the enumeration. Algorithm 1 is designed to identify the corresponding set of auxiliary tables of the subqueries. First, it takes a member set u from the power set and *voids* all the tables in it. If $Q_{\mathcal{H}}$ produces an UNFIT-result, u is included in collection CoreTables. The set CoreTables represents all the *voided states* of the database where no subquery is satisfied. Next, we get the set SideTables, where voiding each member set still causes $Q_{\mathcal{H}}$ to produce FIT-result. They capture the *voided* database states where at least one subquery is satisfied.

In Table 2, these sets are enumerated for the running example of $Q_{\mathcal{H}}$ given in Figure 7a. For instance, $\{s, c\}$ ∈ CoreTables because voiding SUPPLIER and CUSTOMER together discards the satisfying tuples for both the subqueries in $Q_{\mathcal{H}}$. On the other hand, $\{s, l\}$ ∈ SideTables because voiding SUPPLIER and LINEITEM together discards the satisfying tuples only for the second subquery.

Next, we identify the *maximal members* of SideTables. Adding any table from AuxTables($Q_{\mathcal{H}}$) to these members gets a set already in CoreTables. E.g. set $\{c\}$ is in Max-SideTables because when CUSTOMER table is void, $Q_{\mathcal{H}}$ produces a FIT-result, and if any other table is voided, say l (LINEITEM), the result of $Q_{\mathcal{H}}$ becomes UNFIT (set $\{c, l\}$ belongs to CoreTables). Thus, the construction of Max-SideTables isolates the individual subqueries. When one member set of Max-SideTables is void, exactly *one subquery is active.*

4.3.4 Candidate Pruning. For a set u in CoreTables, its supersets are also in CoreTables. If voiding tables in u obtain an UNFIT-result for $Q_{\mathcal{H}}$, more void tables cannot produce FIT-tuples. This fact is leveraged to reduce the iterations of voiding, by checking elements from the power set lattice bottom-up. Lines 5-7 of the algorithm capture it.

On the flip side, if a set u is in SideTables, all its subsets must be in SideTables. If voiding tables in u could not produce UNFIT-result for $Q_{\mathcal{H}}$, then voiding lesser number of tables cannot produce UNFIT result. This fact is leveraged to reduce the iterations of voiding, by checking elements from the power set lattice top-down. Lines 8-10 of the algorithm capture it.

4.3.5 Iterative Subquery Extraction. After AuxTables(q_i) is computed, the subqueries of $Q_{\mathcal{H}}$ are extracted in a loop (SPJGAOL loop in Figure 8). In each iteration i, all tables other than FROM(q_i) are voided, to ensure that execution of $Q_{\mathcal{H}}$ produces results only from q_i. Then the UNMASQUE pipeline is used to extract the SPJGAOL-subquery. The overall flow is shown in Figure 8. In this context, note that a nested subquery is extracted as a flat query by Algorithm 1 (similar to what happened for q_2 in Figure 7b).

4.3.6 Mutation Overheads. Executing $Q_{\mathcal{H}}$ over the power set enumerated in Algorithm 1 could, in principle, be computationally highly expensive. However, in practice, the number of auxiliary tables is often limited – for instance, it could correspond to the fact-tables in a warehouse schema, which is usually a small value – so even the power set may not be unviably large. Further, execution overheads could be reduced by various optimizations, such as constructing a rich set of column indexes, as explained below.

4.4 Performance Impacts and the measures

Algorithm 1 could incur significant overheads if D_I is large. Therefore, XRE employs the following techniques to reduce bottlenecks.

4.4.1 Correlated Sampling [38]. It is a technique that makes use of the schema join graph in the sampling process. This results in a higher probability of the sampled data satisfying the join predicates. It is used before the database minimization step to obtain a smaller D_I that produces a FIT-result.

4.4.2 Using Index. In our physical schema, we build index on all the attributes of all the tables so that $Q_{\mathcal{H}}$ executes as fast as possible. Index maintenance is not of concern since (1) most of our database operations are on the metadata, and (2) write operations are on D^1, except for the database minimization, as handled below.

4.4.3 View-based Database Minimization. We employ a minimization technique based on *virtual views*, which does not require copying the records of a table during the binary halving process. The views are created on the base table by utilizing system-generated tuple identifiers, which give the physical location of a row in the table – for instance, in PostgreSQL, this identifier is called `ctid` and consists of a block number and a record number within that block. The `ctid` of the first record of a table is (0, 1). The number of records in a block, n_b, is computable from the schema, based on which we can estimate the `ctid` of the middle row. The following queries create a view containing roughly the upper half of table T:

```
Alter Table T Rename to T_dummy;
Create View T as Select * From T_dummy
    Where ctid between '(0,1)' and '(|T_dummy|/2n_b,1)';
```

If a FIT-result is obtained, the view creation continues recursively with the upper half; if not, it shifts to a virtual view on the lower half. This reduction continues until a D^1 is achieved.

4.5 Proof of Correctness

Lemma 1 proves that the From clause of each of the *n* subqueries are identified uniquely (injection). Lemma 2 proves that no subquery is left (surjection). Therefore, Algorithm 1 is correct (bijection).

Lemma 1. *For any $from_i \in Froms$ returned by Algorithm 1, voiding all the tables not in $from_i$ keeps only one subquery in $Q_{\mathcal{H}}$ active.*

Proof. Since the algorithm constructs set *Froms* by including COMMON($Q_{\mathcal{H}}$) to each members of set Aux, we prove the following: For any $p \in$ Aux, voiding all the table in $\overline{p}$ (i.e. $\overline{p}$ = AuxTables($Q_{\mathcal{H}}$) $- p$) on D_I, exactly one of the subqueries in $Q_{\mathcal{H}}$ produces FIT-result.

- Assume that no subquery in $Q_{\mathcal{H}}$ gives a FIT-result, i.e. $\overline{p} \in$ CoreTables. Since $p \in$ Aux, $\overline{p} \in$ Max-SideTables, obtaining a contradiction. Therefore, at least one subquery produces FIT-result.
- Assuming the contradiction, let q_1 and q_2 be two distinct subqueries that produce FIT-results when the $\overline{p}$ tables are void. Now we void some more tables, pd_{12}: the auxiliary tables that are in q_1 but not in q_2. Therefore, now only q_2 produces a FIT-result. Thus, $\overline{p} \cup pd_{12} \in$ SideTables. But $\overline{p} \in$ Max-SideTables since $p \in$ Aux is given. Therefore, $\overline{p} \cup pd_{12}$ must belong to CoreTables, which is a contradiction. So, two subqueries cannot produce FIT-results.
- The above contradiction extends for $q_1, q_2, \ldots, q_m$, with $m > 2$. As a result, there must be only one subquery producing FIT-results. Consequently, p has the auxiliary tables of a unique subquery.

Lemma 2. *$\forall i, 1 \le i \le n$, FROM(q_i) is a member of set Froms.*

Proof. Let q_0 be an omitted subquery. So, AuxTables(q_0) is not included in set Aux. Therefore, $\overline{\text{AuxTables}}(q_0) \notin$ Max-SideTables. So, it is either (1) in CoreTables; or (2) in SideTables, and some superset of it in Max-SideTables. Case (1) implies voiding all the auxiliary tables except the ones in q_0 obtains UNFIT-result from $Q_{\mathcal{H}}$, which is impossible. For case (2), the only way to form a superset of $\overline{\text{AuxTables}}(q_0)$ is to include at least one member from AuxTables(q_0). When tables of such a set are voided, $Q_{\mathcal{H}}$ produces UNIT-result. Therefore, Case (2) also is a contradiction. Consequently, q_0 is not left out from the coverage of Aux.

We can generalize the above contradiction considering m omitted subqueries $q_1, q_2, \ldots, q_m$. Consequently, no subquery is omitted, i.e., every subquery has an associated mapping in Aux. Formally, AuxTables(q_i) $\in$ Aux, $\forall i, 1 \le i \le n$. Therefore, following the construction of the set *Froms* in the algorithm, FROM(q_i) = AuxTables(q_i) $\cup$ COMMON($Q_{\mathcal{H}}$) is in set *Froms*, $\forall i, 1 \le i \le n$.

5 Algebraic Predicates

In Xpose, we consider predicates comparing an attribute to either (1) a value, or (2) another attribute. While the predicate of type (1) is termed as arithmetic filter predicates in Unmasque, we use a general term *algebraic predicates* to refer to both types. We now discuss how to extract them using database mutation from the signatures available in D^1.

5.1 Problem Formulation

Let σ_p denote the set of join and filter predicates that appear in *a conjunction* in $Q_{\mathcal{H}}$. Let us refer to the individual predicates by σ_i, for some integer i, $1 \le i \le n$ in total n predicates.

- ***Extraction Goal*** Determine individual σ_i in σ_p,

(1) When σ_i is an arithmetic predicate *column op value*. Formally, $\sigma_i = (col_x, op, v_x)$, attribute col_x belongs to a relation in FROM($Q_{\mathcal{H}}$), v_x is a constant value from the domain of col_x, op belongs to the comparison operator set $\{=, \le, \ge\}$

(2) When σ_i is an algebraic predicate *column op column*. Formally, $\sigma_i = (col_x, op, col_y)$, col_x, col_y are attributes of the same domain, which belong to the same or different relations in FROM($Q_{\mathcal{H}}$), op belongs to the operator set $\{=, \le, \ge, <, >\}$

5.1.1 Operators. The operator set $\{=, \le, \ge, <, >, \ne\}$ is exhaustive wrt binary comparison predicates when the attributes belong to ordered domain such as numbers and dates. Note that the operator set for the arithmetic predicates does not include $<$ and $>$, whereas they are included for defining the algebraic predicates. The reason is, $col_x < 5$ is equivalent to $col_x \le 4$, considering col_x as an integer. Consequently, $\le$ and $\ge$ operators are sufficient to cover $<$ and $>$ respectively. Operator $\ne$ is not explicitly handled in XRE since in case of its rare occurrences, the text description captures it explicitly, and so it is derived by XFE.

5.1.2 The Predicate Attributes. The above definition of σ_p includes a comparison relationship between any two attributes, which allows us to treat non-key joins in addition to the key-based joins, and comparison between the attributes of the same relation uniformly.

(a) Minimized Database D^1

Table: D^1.LINEITEM

l_orderkey	*l_commitdate*	*l_receiptdate*	*l_suppkey*	*l_shipmode*
1448519	1992-09-23	1992-09-23	2032	'AIR'

Table: ORDERS

o_orderkey	*o_totalprice*
1448519	197740.95

Table: SUPPLIER

s_acctbal	*s_suppkey*
8968.42	2032

(b) Extracted S-value intervals from D^1

Attribute	s-value interval	Attribute	s-value interval
l_orderkey, *o_orderkey*	[1448519, 1448519]	*s_acctbal*	[i_{min}, 197740.95]
l_suppkey, *s_suppkey*	[2032,2032]	*l_commitdate*, *l_receiptdate*	[1992-09-23, 1992-09-23]

Figure 9: Showcasing D^1 and SVI for q_2 in Figure 7a

5.2 Extraction Process Overview

After obtaining D^1 that produces a FIT-result from $Q_{\mathcal{H}}$, s-value extractor SVE is run once to get the s-value intervals of all the attributes, denoted by SVI. In Figure 9(a), D^1 for the q_2 subquery of Figure 7a is listed. SVI obtained from this database is listed in figure(b).

The interval of $[l, r]$ for col_x is expressed as (col_x, $\geq$, l), and (col_x, $\leq$, r). With D^1, and SVI as inputs, the σ_p-extraction algorithms are executed. We have devised two different algorithms to extract the equality and inequality predicates SVE is used in multiple rounds by these algorithms to discover the floating s-value intervals, and the dependent attributes. Note that $\leq$ also covers $\geq$ ($col_1 \leq col_2 \implies col_2 \geq col_1$). This process picks out some members from the input set SVI. The remaining s-value intervals in SVI are finalized as the arithmetic predicates (col_x, op, v_x).

5.3 Equality (=) Predicate Extraction

The attributes involved in an equality predicate may or may not belong to the same relation. When they are from different relations, the predicate implies *equi-joins*.

5.3.1 **Matched Attributes.** Matched Attributes or an *MA-set* is a set of attributes having the same value v in D^1, and the same s-value interval $[v, v]$. In Figure 5(c), it can be observed that *l_orderkey* and *o_orderkey* form an MA-set, with their only s-value 1448519, and *l_commitdate* and *l_receiptdate* form another MA-set with only s-value DATE '$1992-09-23$'. One MA-set captures the possibility of an algebraic equality. E.g. predicates $col_1 = col_2$ and $col_2 = col_3$ has a single equality relationship among col_1, col_2, col_3 (expressed as two equality predicates).

One MA-set is a possible candidate for an equality relationship. E.g. $Q_{\mathcal{H}}$ with WHERE clause $col_1 = col_2$ and $col_2 = col_3$ has a single equality relationship among col_1, col_2, col_3 (it is expressed as two equality predicates). However, when an MA-set has more than 3 attributes, there may be more than one equality relationship. E.g. $col_1 = col_2$ and $col_3 = col_4$. Due to their coincidental same values in D^1, they form a single MA-set. Section 5.3 elaborates how the equality predicates are refined from the MA-sets.

5.3.2 **Group S-value Extractor.** While the notion of MA-sets is introduced to identify a group of *interesting* attributes, XPOSE needs to mutate them *together* to extract the equality relationships among them. Consequently, we have *group mutation* technique (the same binary search-based technique as discussed in Section 2.3), to extract a common s-value interval for the MA-set under consideration. We use notation GS_{ve} to denote this component.

Algorithm 2: Equality Predicate Extraction

Input : $Q_{\mathcal{H}}$, D^1, SVE, GS_{ve}, SVI
1 **Pre-processing**
2 $Part_{eq} \leftarrow$ Empty Set
3 **for** *each s-value interval [lb, ub]* $\in SVI$ **do**
4 Ignore the bounds which are domain boundaries
5 **for** *each s-value equality interval [B, B]* $\in SVI$ **do**
6 $M_A \leftarrow$ Set of all col_x which have such s-value intervals
7 Include MA-set (B, M_A) in $Part_{eq}$ as a member tuple
8 **The Main Algorithm**
9 $\sigma_{eq}, Done \leftarrow$ Empty Sets
10 **for** *each MA-set* $(B, M_A) \in Part_{eq}$ **do**
11 **if** $|M_A| = 1$ **then** // The only attribute in M_A has equality predicate of B
12 include the predicate in σ_{eq}
13 **else if** $|M_A| \leq 3$ // My size ≤ 3
14 *or every possible subset* S_e *of* M_A *has entry* (B, S_e) *in Done* // All of my subsets have been tried out
15 **then**
16 Run GS_{ve} to extract S-value interval $[LB, UB]$ for M_A
17 **if** $LB = UB = B$ **then**
18 **continue** // invalid Partition, Go to Line 10
19 **else** // The attributes in M_A are in algebraic equality relationship, with arithmetic upper and lower bounds at LB and UB respectively
20 include the equality predicates in σ_{eq}
21 include (B, M_A) in set $Done$
22 **else**
23 **for** $i \leftarrow 2$ *to* $|M_A| - 2$ **do**
24 Create M_A-Subsets of size i that is not present in $Part_{eq}$ and in $Done$
25 include the above sets in $Part_{eq}$
26 **continue** // Go to next element in Line 10
27 remove the attributes of M_A from all MA-sets in $Part_{eq}$
28 **return** σ_{eq}

5.3.3 **Basic Group Mutation.** The group s-value extractor GS_{ve} mutates the attributes in an MA-set over $[i_{min}, i_{max}]$ interval *together* and finds out the common s-value interval. This is used for $n \leq 3$ to extract predicates of the form $l \leq col_1 = col_2 = \cdots = col_n \leq r$. For instance, in the D^1 shown in Figure 5(a), attributes *l_orderkey* and *o_orderkey* from are mutated together by GS_{ve} to extract the algebraic relationship $i_{min} \leq l\_orderkey = o\_orderkey \leq i_{max}$. Similarly, $i_{min} \leq l\_commitdate = l\_receiptdate \leq i_{max}$ equality is also extracted. Note that, $[i_{min}, i_{max}]$ is integer domain in the former case, whereas it is the date domain in the latter case. Handling of MA-set of larger size needs further handling as they may be encapsulating multiple equality relationship coincidentally. In such cases, every possible subset of size 2 needs to be mutated together by GS_{ve} while keeping the other attributes in the MA-set the same.

5.3.4 **Larger Group Mutation .** For MA-sets of having more than 3 attributes, we need to ensure that the different equality relationships are separated. Therefore, we partition one MA-set

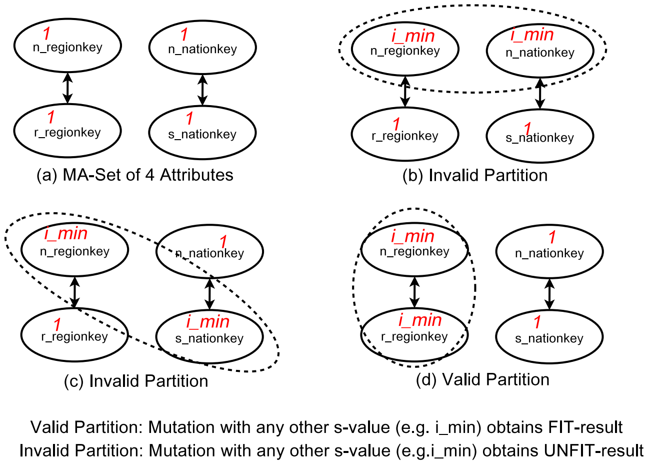


Figure 10: Partition MA-set ($n = 4$) into size i and $n - i$, for $i = 2$.

into two and apply GS_{ve} to *mutate any one* over $[i_{min}, i_{max}]$. $Q_{\mathcal{H}}$ producing FIT-results (implies that all equalities are satisfied) on such mutations over interval $[l, r]$ implies a correct separation, for $i_{min} \leq l, r \leq i_{max}, l \neq v$ or $r \neq v$. Since an equality predicate requires at least two attributes, we partition an MA-set of size n into i and $n-i$-sized sets, $2 \leq i \leq n-2$. For each i, we enumerate all i-sized sets. Figure 10 depicts the partitioning for the running example, where n is 4. If GS_{ve} extracts a s-value interval other than $[v, v]$ for any of these partitions, it is a separate equality relationship. The separation continues until all MA-set of size > 3 are verified.

5.4 Inequality ($\leq, \geq$) Predicate Extraction

Attributes involved in inequality relationships have different values in D^1, and different s-value bounds. XRE makes pairs of possible inequality candidates and then validates them. An inequality predicate $col_x \leq col_y$ is represented as an edge $col_x \longrightarrow col_y$, from vertex col_x to col_y. Multiple such pairs together construct one or more chains of attributes of the form $col_1 \rightarrow col_2 \rightarrow \cdots \rightarrow col_n$. Due to a cyclical join graph, such chains may form circles. We are interested in DFS orders of such chains, with the traversal terminated at an already visited attribute. Algorithm 3 processes all such possible DFS paths to determine each $\leq$ relationship among the attributes.

5.4.1 Enumerate Inequality Candidates (Edge-Set E).

With inputs D^1 and SVI, we construct edge set E capturing all possible $\leq$ relationships among pairs of attributes. E is initiated from the static s-value bounds SVI, as identified by SVE initially. In particular, for s-value interval $[l, r]$ for col_x, we add edges $l \longrightarrow col_x$ and $col_x \longrightarrow r$ into E. If $D^1.col_x \leq SVI.col_y$.LB, $col_x \longrightarrow col_y$ edge is also added into E (for $col_x \leq col_y$ possibility). The goal is to refine E into σ_p. From Figure 5(a) and (c), we have $D^1.s\_acctbal = 8968.42$, and LB of s-value interval of $o\_totalprice = 8968.42$. So, $s\_acctbal \leq o\_totalprice$ is a possibility. Algorithm 4 lists this construction.

Algorithm 3: Inequality Predicate Extraction

Input :SVE, $Q_{\mathcal{H}}$, D^1, C_E and SVI produced by SVE on D^1

$E \leftarrow$ `Build_Edge_Set_E`$(C_E,\ SVI)$ (Section 5.4.1)
for *each path in the DFS paths of E* **do**
  $seq \leftarrow$ from source to sink sequence of $path$
  for *i from 1 to* $|seq|$ **do**
    $col_{src} \leftarrow seq[i]$
    $col_{snk} \leftarrow seq[i+1]$ **if** $i < |seq|$ **else** $NULL$ // may be $col_{src} \leq col_{snk}$
    if col_{snk} *is not NULL* **then**
      E, absorbed bounds $\leftarrow$ `Confirm_LEQ`$(Q_{\mathcal{H}},\ D^1,\ E,$ SVE, $col_{src},\ col_{snk})$ (Section 5.4.2)
    $E \leftarrow$ `Extract_Static_LB`$(D^1,\ col_{src},$ SVE, $E)$ (Section 5.4.3)
  $E \leftarrow$ `Extract_Static_UB`$(D^1,\ seq,$ SVE, $E)$ (Section 5.4.3)
return E

Algorithm 4: Build Edge-Set E

Function `Build_Edge_Set_E`$(C_E,\ SVI)$
  $E \leftarrow$ Empty set
  for *each s-value interval* $[lb, ub]$ *of* $col_x \in SVI$ **do**
    include (col_x, ub) and (lb, col_x) in E // $lb \leq col_x$, and $col_x \leq ub$
  for $col_x \in C_E$ **do**
    for *each* $col_y \in C_E$ *that is different from* col_x **do**
      if $D^1.col_x \leq SVI.col_y.LB$ // may be $col_x \leq col_y$
      then include (col_x, col_y) in E
  Remove all transitive paths from E
  return E

5.4.2 Mutate to Float the S-value Bounds.

Now, for each $col_x \rightarrow col_y$ in E, it is to determine whether any mutation of col_x impacts the lower bound of col_y. So, col_x is mutated with a value ub (its own UB), and then SVE is used on col_y to extract its bounds. If $SVI.col_y$.LB also changes now to be ub, we confirm the case. Consequently, the static UB of col_x and LB of col_y can be removed from E because they floated. For the running example, we represent the possibility $col_x \leq col_y$ as a dashed directed edge from col_x to col_y in Figure 11(a). In Figure 11(b), $D^1.s\_acctbal$ is mutated with its own UB 197740.95, which caused $o\_totalprice$ to have a new LB of 197740.95. Therefore, $s\_acctbal \leq o\_totalprice$.

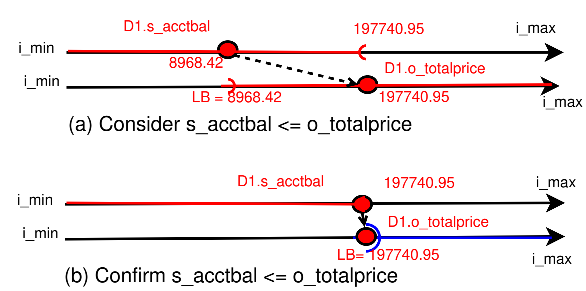


Figure 11: Identifying dependency between two attributes

5.4.3 Interfering Static Bounds.

With additional static bounds on the attributes, in effect, $col_x \leq col_y$ may not hold for the entire

Algorithm 5: Confirm LEQ

1 **Function** Confirm_LEQ($Q_{\mathcal{H}}$, D^1, E, SVE, col_{src}, col_{snk})
2 $\quad prev\_lb_{snk} \leftarrow lb$, where $(lb, col_{snk}) \in E$ // LB of col_{snk}
3 $\quad prev\_ub_{src} \leftarrow ub$, where $(col_{src}, ub) \in E$ // UB of col_{src}
4 $\quad$ Mutate $D^1.col_{src}$ with value ub
5 $\quad$ **if** *$Q_{\mathcal{H}}$ obtains FIT-result* **then**
6 $\quad\quad nw\_lb_{snk} \leftarrow$ LB of col_{snk} extracted by SVE // LB may float due to mutation
7 $\quad\quad$ **if** $prev\_lb_{snk} \neq nw\_lb_{snk}$ **then**
$\quad\quad\quad$ // LB of col_{snk} floated because col_{src} was mutated
8 $\quad\quad\quad$ Absorb (lb, col_{snk}) and (col_{src}, ub) // because $col_{src} \leq col_{snk}$ is confirmed
9 $\quad$ Revert mutation done by Line 4
10 $\quad$ **if** *No bound was absorbed in Line 8* **then**
11 $\quad\quad lb_{src} \leftarrow$ LB of col_{src} extracted by SVE
12 $\quad\quad$ Do 4-9 using mutation value lb_{src}.
13 $\quad$ Remove the absorbed bounds from E
14 $\quad$ **return** E

Algorithm 6: Extract Static Lower Bounds

Function Extract_Static_LB(D^1, col_{src}, SVE, E)
$\quad mut\_lb_{src} \leftarrow$ LB of col_{src} extracted by SVE // Actual LB of col_{src}
$\quad min\_val_{src} \leftarrow val$ **if** $(val, col_{src}) \in E$ **else** i_{min} // Ideal LB of col_{src} as per E
$\quad$ **if** $mut\_lb_{src} \neq min\_val_{src}$ **then**
$\quad\quad$ // There is a cut at the actual LB
$\quad\quad$ include $(mut\_lb_{src}, col_{src})$ in E
$\quad\quad$ Mutate $D^1.col_{src}$ with value mut_lb_{src} // mutate with LB permanently, so that in the next iteration, the next column can be checked in this manner
$\quad$ **return** E

Algorithm 7: Extract Static Upper Bounds

Function Extract_Static_UB(D^1, seq, SVE, E)
$\quad$ **for** *i from $|seq|$ down to* 1 **do**
$\quad\quad col_c \leftarrow seq[i]$
$\quad\quad mut\_ub_c \leftarrow$ UB of col_c extracted by SVE // Actual UB of col_c
$\quad\quad max\_val_c \leftarrow val$ **if** $(col_c, val) \in E$ **else** i_{max} // Ideal UB of col_c as per E
$\quad\quad$ **if** $mut\_ub_c \neq max\_val_c$ **then**
$\quad\quad\quad$ // There is a cut at the actual UB
$\quad\quad\quad$ include (col_c, mut_ub_c) in E
$\quad\quad$ Mutate $D^1.col_c$ with value mut_ub_c // mutate with UB permanently so that in the next iteration, the previous column can be checked in this manner
$\quad$ **return** E

$[i_{min}, i_{max}]$ interval. To identify whether such cuts exist in the $col_x \leq col_y$ relationship, we check the bound behaviors in the *same directions*. When col_x is mutated with its LB, what is the LB of col_y; when col_y is mutated with its UB, what is the UB of col_x. If bounds of the same direction do not match, it confirms a cut. For the generalization to a chain $col_1 \rightarrow col_2 \rightarrow \cdots \rightarrow col_n$, the DFS order is followed to confirm the LBs, and the reverse DFS order is followed for the UBs. In our running example, such predicates do not exist.

Algorithms 6 and 7 are for extracting the lower and upper bounds respectively.

5.5 Inequality (<, >) Predicate Extraction

When a mutation of col_x with value b results in the new LB of col_y of $b + \Delta$, the $col_x < col_y$ case is confirmed (refer to Δ in Section 5.1.1). This small addition to the technique in Section 5.4.2 can be maintained separately than in E to keep track of the $<$ predicates.

5.6 Outer Join Predicates

Even though the above logic is not devised to extract outer joins, it identifies the corresponding equality join predicates. In the above mutation-based technique, the satisfaction of $Q_{\mathcal{H}}$ is wrt FIT results. To produce such a result, the join attributes need matching values. Result tuples with NULL values produced by $Q_{\mathcal{H}}$ due to outer join are thus ignored. So, all join predicates are identified (i.e. no equality predicate is missed), and they are extracted as equi-joins. Predicate $c\_custkey = o\_custkey$ in subquery q_1 in Figure 7a is thus extracted, as shown in Figure 7b.

5.7 Semi-join Semantics

In the case of at least one matching, in the D^1, since they are the only tuples, they are extracted as equality predicate (i.e. all semantics) by the above logic. Due to this, $l\_suppkey = s\_suppkey$ in Figure 7b was extracted.

5.8 Generalized Linear Inequality Predicates

Section 5 formalized algebraic predicates as col_x op col_y. However, generalized predicates include constants co-efficients in the inequality, e.g. $col_x \leq a * col_y + b$, for constants a and b. Here we have the following equality relationship: $col_x.UB = a * col_y + b$. Therefore, we require two mutation of col_y in D_{min} (one with $y_1 = col_y.LB$, and the other with $y_2 = col_y.UB$), and extract the respective $col_x.UB$s (let us refer to them as x_1 and x_2 respectively). Since col_x and col_y are in linear dependency, their *rate of changes* are directly proportional, which is given by $a = (x_2 - x_1) / (y_2 - y_1)$. b can be computed next when we know a.

5.9 Proof of Correctness

5.9.1 *Equality: Algorithm 2 is correct.*

Proof. We prove that $l \leq col_1 = col_2 = \cdots = col_n \leq r$ is extracted correctly. Let us refer to the attributes together as equality set C_n. Assuming the contradiction, let Algorithm 2 extract some other lower and upper s-value bounds l_k and r_k for a partition of size k (i.e. $C_k = \{col_1, col_2, \ldots, col_k\}$), for some $k < n, l_k \neq l, r_k \neq r$. Therefore, a mutation where col_n has one value and the attributes in C_k have a different value, gets a FIT-result from $Q_{\mathcal{H}}$. Either of l and r can serve as this value, when C_k is mutated within interval $[l_k, r_k]$. This is possible only when $col_i \neq col_n$ satisfies $Q_{\mathcal{H}}$ for any $1 \leq i \leq k$. This violates the fact the $Q_{\mathcal{H}}$ has equality within C_n. So, the algorithm does not extract a wrong inequality.

5.9.2 Inequality: Algorithm 3 is correct. Combining Lemmas 3-6, we prove the correctness.

Lemma 3. *If $Q_{\mathcal{H}}$ has a hidden algebraic predicate chain of the form $l \rightarrow col_1 \rightarrow col_2 \rightarrow \cdots \rightarrow col_n \rightarrow r$, where $i_{min} \leq l \leq r \leq i_{max}$, and $\rightarrow$ is either $\leq$ or $<$, and no other predicate involving col_i $\forall i, 1 < i < n$ exists in $Q_{\mathcal{H}}$, Algorithm 3 extracts it correctly.*

Proof. Apart from $col_i \rightarrow col_{i+1}$, the only predicates in $Q_{\mathcal{H}}$ involving col_i is $l + \delta_2 * \Delta \rightarrow col_i$, and involving col_{i+1} is $col_{i+1} \rightarrow r - \delta_3 * \Delta$, where δ_2 is the number of preceding $<$ operators of col_i in σ_{chain}, and δ_3 is the number of succeeding $<$ operators of col_{i+1} in σ_{chain}. Therefore, $SVI.col_i.LB = l + \delta_2 * \Delta$, $SVI.col_{i+1}.UB = r - \delta_3 * \Delta$. Assume the contradiction, $col_i \rightarrow col_{i+1} \notin E$ when Algorithm 3 terminated. Let $D^1.col_i = v_i$, $D^1.col_{i+1} = v_{i+1}$ for some $l + \delta_2 * \Delta \leq v_i \leq v_{i+1} \leq r - \delta_3 * \Delta$, which is bound to be true for the given $Q_{\mathcal{H}}$. Since no other predicate involves col_i and col_{i+1}, $SVI.col_i.UB = v_{i+1}$, $SVI.col_{i+1}.LB = v_i$, which ensures that Algorithm 3(pre-processing) includes $col_i \rightarrow col_{i+1}$ in E. It can only be removed from E if mutation of col_i does not impact the LB of col_{i+1}. Thus, when col_i is mutated with v_{i+1}, LB of col_{i+1} still remains v_i. The same holds when the mutation value is $l+\delta_2*\Delta$. So, $l + \delta_2 * \Delta \leq v_{i+1} \leq v_i$, which is possible only if $l + \delta_2 * \Delta = v_i = v_{i+1}$. It obtains the same LB and UB for col_i. The UB must be static to match the static LB of $l + \delta_2 * \Delta$. It contradicts col_i not having any more predicates.

Lemma 4. *If $Q_{\mathcal{H}}$ has a hidden algebraic predicate chain of the form $l \rightarrow col_1 \rightarrow col_2 \rightarrow \cdots \rightarrow col_n \rightarrow r$, where $i_{min} \leq l \leq r \leq i_{max}$, and $\rightarrow$ is either $\leq$ or $<$, and for any $col_i \rightarrow col_{i+1}$ pair in the predicate chain, at least one of the static bounds $LB_{i+1} \rightarrow col_{i+1}$ and $col_i \rightarrow UB_i$ exists, Algorithm 3 extracts the predicate chain correctly.*

Proof. $UB_i < LB_{i+1}$ implies $col_i \rightarrow col_{i+1}$ is a redundant predicate. Therefore, we prove that Algorithm 3 extracts $col_i \rightarrow col_{i+1}$ when $LB_{i+1} \rightarrow UB_i$. Now, it is given that $SVI.col_{i+1}.LB = LB_{i+1}$, and $SVI.col_i.UB = UB_i$. Let $D^1.col_i = v_i$, $D^1.col_{i+1} = v_{i+1}$.
(1) Let $v_i, v_{i+1} \in [LB_{i+1}, UB_i]$. We also have $v_i \rightarrow v_{i+1}$ due to σ_{chain}. The given static LB of col_{i+1} can only happen in the presence of σ_{chain} if $v_i \rightarrow LB_{i+1}$. Due to our initial assumption, $v_i = LB_{i+1}$. Therefore, when $D^1.col_i$ is mutated with a higher value UB_i, $SVI.col_{i+1}.LB$ becomes UB_i (increases), i.e. gets impacted. Therefore, the algorithm extracts $col_i \rightarrow col_{i+1}$.
(2) Let $v_i \in [i_{min}, LB_{i+1})$, $v_{i+1} \in (UB_i, i_{max}]$. So, when col_i gets mutated with a higher value UB_i, following the construction in the earlier case, $SVI.col_{i+1}.LB$ gets impacted, extracting $col_i \rightarrow col_{i+1}$.

Lemma 5. *Algorithm 3 does not extract a predicate $col_x \rightarrow col_y$ that is absent in $Q_{\mathcal{H}}$. (The proof is skipped due to its triviality.)*

Lemma 6. *If $Q_{\mathcal{H}}$ has a hidden algebraic predicate chain of the form $l \rightarrow col_1 \rightarrow col_2 \rightarrow \cdots \rightarrow col_n \rightarrow r$, where $i_{min} \leq l \leq r \leq i_{max}$, and $\rightarrow$ is either $\leq$ or $<$, Algorithm 3 extracts all the arithmetic predicates $(col_i, \leq, UB_i)$ and $(col_i, \geq, LB_i)$ correctly, $\forall i, 1 \leq i \leq n$.*

Proof. Assuming a contradiction, the algorithm obtained $col_i \leq col_i$, and $lb_i \leq col_i$, such that $lb_i \neq v_i$. $v_i \leq col_i$ is given to be in $Q_{\mathcal{H}}$. Since $Q_{\mathcal{H}}$ produces UNFIT-result on $lb_i < v_i$, our assumption leads to $v_i < lb_i$. This implies $SVI.col_i.LB = lb_i$ when col_{i-1} has mutated value of its own lower bound. So, col_{i-1} has a minimum possible value satisfying $Q_{\mathcal{H}}$ is lb_i, i.e. $lb_i \leq col_{i-1}$. This is a contradiction, given that $v_i \leq col_i$. Similarly, the $\geq$ can also be proved.

6 Logical Disjunction in Predicates

Now we turn our attention to extracting the logical disjunctions present in the Where clause. We assume the Where clause is *conjunction of disjunctions of multiple predicates.*

6.1 Extraction Algorithm

The initial pass of filter predicate extraction (by Section 2) extracts a set of arithmetic predicates in conjunction. Then, for each predicate p_i, we create a $\widehat{D_I}$, which is the original D_I but *eliminating* the rows that satisfy p_i. The database minimization and predicate extraction loop is repeated now, to obtain the set of predicates that are in disjunction with p_i. Such iterations continue, progressively reducing the $\widehat{D_I}$ contents until $Q_{\mathcal{H}}$ does not produce a FIT-result on the current $\widehat{D_I}$. This technique covers the extraction of IN operator with constant values.

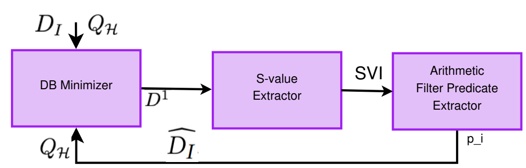


Figure 12: Iterative Extraction of Disjunctive Predicates

6.2 Example

In Figure 9(a), `l_shipmode` the value 'AIR' in D^1. XRE extracts `l_shipmode = 'AIR'` from it. Then, $\widehat{D_I}$ is created as follows:

```
Alter Table Lineitem Rename to Lineitem_dummy;
Create Table Lineitem as
  Select * From Lineitem_dummy Where l_shipmode != 'AIR';
```

Query q_2 listed in Figure 7a still produces FIT-result on the above database. The database minimizer minimizes $\widehat{D_I}$ to be the following D^1. It has `l_shipmode = 'TRUCK'`. The predicate extractor extracts `l_shipmode = 'TRUCK'` from it. Later, while assembling all the extracted predicates together to form the Where clause in $Q_{\mathcal{S}}$, `l_shipmode = 'AIR'` or `l_shipmode = 'TRUCK'` are combined using the IN operator, forming predicate `l_shipmode` IN ('AIR', 'TRUCK').

Table: D^1.Lineitem

l_orderkey	*l_commitdate*	*l_receiptdate*	*l_suppkey*	*l_shipmode*
2171687	1992-03-27	1992-03-27	2793	'TRUCK'

7 Forward Engineering in Xpose

We now turn our attention to XFE, where the predictive abilities of LLMs are used on the textual description TX_Q in conjunction with the grounding provided by $Q_{\mathcal{S}}$, the seed query output by XRE.

7.1 Synthesis Scope

Due to confining attention to the single-row D^1, $Q_{\mathcal{S}}$ will always be either a single flat query, or a union of flat subqueries. However, the original query may incorporate nesting, and we need additional machinery to recover this structure. For tractability, we currently assume that there is, at most, *one-level nesting* in $Q_{\mathcal{H}}$. We also support unions of such one-level nested subqueries, effectively providing a second level of nesting. Accordingly, the synthesis prompts in XFE are designed to address this class of queries.

7.2 Basic Synthesis Prompt

XFE initiates the synthesis task using the prompt template listed in Table 3(a) – this "initial prompt" (IP) comprises the (i) textual description TX_Q, (ii) schema specification, (iii) seed query $Q_{\mathcal{S}}$ produced by XRE, (iv) cardinalities of $\mathcal{R}_{\mathcal{H}}$ and $\mathcal{R}_{\mathcal{S}}$, and (v) general guidelines on synthesizing from $Q_{\mathcal{S}}$ (Table 3a).

The LLM typically finds it easy to infer complex constructs such as nested structures, outer joins, and existential operators, since they are usually expressed directly as such in the business description. As a case in point, TPCH 'Customer Distribution Query' Q13 specifies listing customers and their orders, *including those who have no order* – this clearly maps to outer join. Consequently, even though XRE extracts an outer join as an equi-join predicate (Section 5.6), XFE is able to refine it correctly.

(a) Initial Prompt [IP]
You are an expert in formulating SQL queries from high-level textual business descriptions.
Formulate SQL query for the following description:<TX_Q >
Use the following schema to formulate SQL:<Schema DDL>
Use the following SQL as a seed query. You should refine the seed query to produce the final SQL:<$Q_{\mathcal{S}}$ from XRE>
Follow the refinement guidelines mentioned below:<Guidelines>
(b) Result-Correction Prompt [RCP]
(Query aligned with $Q_{\mathcal{S}}$, but result does not match $Q_{\mathcal{H}}$)
[RCP.v1]
You formulated the following SQL:<Last returned SQL $Q_{\mathcal{E}}$ >
It produces the following number of rows:|<$Q_{\mathcal{E}}$ Result>|
Below is the actual result cardinality:|<$Q_{\mathcal{H}}$ Result>|
The results do not match. Fix the query.
[RCP.v2]
You formulated the following SQL:<Last returned SQL $Q_{\mathcal{E}}$ >
Its result does not match with actual result. Fix the query.
(c) Clause-Correction Prompt [CCP]
(Query is syntactically incorrect or misaligned with $Q_{\mathcal{S}}$)
You formulated the following SQL:<Last return SQL $Q_{\mathcal{E}}$ >
Fix its <incorrect clause> as per $Q_{\mathcal{S}}$ (repeat for relevant clauses)

Table 3: Query Synthesis Prompts in XFE

However, for several other operators, including semi-joins and membership operators, the LLM has a tendency to occasionally go off the rails and produce spurious constructs. To minimize this possibility, we include the detailed guidelines shown in Figure 13 which put in explicit guardrails to make the LLM produce relevant SQL. The guidelines range from the obvious (G1.: "Do not formulate syntactically incorrect SQL") to compliance with $Q_{\mathcal{S}}$ for its provably correct aspects (G5.: "Strictly use only the tables given in the seed query") to more subtle aspects such as not having multiple Count aggregations (G13.: "A subquery may have at most one COUNT() aggregation.") and checking the validity of $Q_{\mathcal{S}}$ predicates (G9.: "Validate all the predicates in the seed query against the textual descriptions."). We have found these guidelines sufficient to handle the queries investigated in our study, but it is, of course, possible that a few more may have to be added for other scenarios.

Basic Guidelines:

G1. Do not formulate syntactically incorrect SQL.
G2. Do not repeat any previously formulated incorrect SQL.
G3. Do not use redundant join conditions or redundant nesting.
G4. Do not use any predicates with place holder parameters.

Guidelines to align synthesis with $Q_{\mathcal{S}}$:

G5. Strictly use the tables given in $Q_{\mathcal{S}}$.
G6. If $Q_{\mathcal{S}}$ has a multi-instance table in its FROM clause, keep all the table instances in your query.
G7. Do not use join predicates absent from $Q_{\mathcal{S}}$.
G8. Strictly reuse the order, attribute dependencies, and aliases of the projections from $Q_{\mathcal{S}}$.

Guidelines to align synthesis with TX_Q:

G9. Validate all the predicates in the seed query against TX_Q. Include all the valid predicates in your query.
G10. For the attributes in the invalid filter predicates, validate their use from TX_Q.
G11. A semi-join, implying at least one match, maybe incorrectly present as an equi-join in $Q_{\mathcal{S}}$.

Guidelines to synthesize compact and meaningful queries:

G12. A subquery used more than once should be a CTE with alias.
G13. A subquery may have at most one COUNT() aggregation.

Guidelines to address result mismatch:

G14. If $\mathcal{R}_{\mathcal{S}}$ has more rows than the actual output, consider performing UNION ALL before GROUP BY.
G15. If $\mathcal{R}_{\mathcal{S}}$ has fewer rows as compared to the actual output, consider either adding more GROUP BY attributes or having more GROUP BY clauses through nestings.

Figure 13: Guideline Instructions for Query Synthesis by XFE

For the example $Q_{\mathcal{S}}$ in Figure 7b, G11. brings in the nested structure and IN operator to rewrite the *l_suppkey = s_suppkey* of $Q_{\mathcal{S}}$ into a semi-join. Further, moving the GROUP BY operator to the outer query is triggered by G14.. In this manner, the prompts direct the LLM towards an accurate extraction.

The following query was produced at first with the basic prompt from the seed query listed in Figure 3(c).

```
SELECT s_name, s_address
FROM supplier
WHERE s_suppkey IN (
    SELECT ps_suppkey
    FROM partsupp
    WHERE ps_partkey IN (
        SELECT l_partkey
        FROM lineitem
       WHERE l_shipdate BETWEEN '1995-01-01' AND '1995-12-31'
          AND l_quantity > 0.5 * (
              SELECT SUM(l_quantity)
              FROM lineitem
         WHERE l_shipdate BETWEEN '1995-01-01' AND '1995-12-31'
                AND l_partkey = partsupp.ps_partkey
                AND l_suppkey = partsupp.ps_suppkey
          )
    )
```

```
    AND ps_availqty > 0.5 * (
        SELECT SUM(l_quantity)
        FROM lineitem
      WHERE l_shipdate BETWEEN '1995-01-01' AND '1995-12-31'
         AND l_partkey = partsupp.ps_partkey
         AND l_suppkey = partsupp.ps_suppkey
    )
)
AND s_nationkey = (
    SELECT n_nationkey
    FROM nation
    WHERE n_name = 'FRANCE'
)
AND EXISTS (
    SELECT 1
    FROM part
    WHERE p_partkey = partsupp.ps_partkey
      AND p_name LIKE '%ivory%'
)
ORDER BY s_name ASC;
```

As can be noted, it has syntax error of using partsupp.ps_partkey without having partsupp in its outer FROM clause. This needs further prompts to fix it, as discussed next.

7.3 Error Feedback Prompts

Feedback prompting is triggered when the synthesized query from the initial prompt differs wrt $Q_{\mathcal{H}}$ in terms of the results on D_I. Specifically, a "result-correction" prompt (RCP), shown in Table 3(b), is submitted to the LLM asking for query correction to address the mismatch. This prompting is repeated until one of the following occurs: (1) the results match, signaling a successful termination; (2) the results of the synthesized query do not change wrt the previous incorrect formulations; (3) the number of unsuccessful trials exceed a threshold.

Apart from result mismatch, it is also possible that the synthesis may introduce new elements not compliant with the provably correct components of $Q_{\mathcal{S}}$. To address such errors, a "clause-correction" prompt (CCP), shown in Table 3(c) is submitted to the LLM.

These feedback prompts are iteratively exercised.

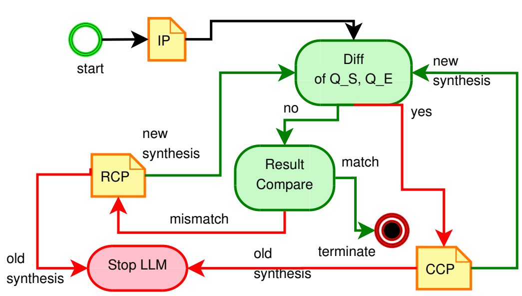


Figure 14: Prompting Flow in XFE

The overall prompting flow in XFE is shown in Figure 14. However, it is possible that, despite several prompt-based efforts, we fail to achieve a satisfactory extraction. For such cases (which only occurred rarely in our evaluation in Section 8), we carry out an enumerative combinatorial synthesis, described next.

Continuing with our example of Q20 synthesis, for the formulated query listed before, prompts (1. CCP, 2. RCP v1) were used in sequence, after which the final correct SQL was achieved. The CCP is shown below.

(1)
```
You formulated the following SQL: <last query>
It has syntax error: ERROR:  missing FROM-clause entry for table "partsupp"
LINE 34:     WHERE p_partkey = partsupp.ps_partkey
                               ^

SQL state: 42P01
Character: 903

Fix the query.
```

The following query was produced which matched $Q_{\mathcal{H}}$ in terms of execution result on D_I.

```
SELECT s_name, s_address
FROM supplier
WHERE s_suppkey IN (
    SELECT ps_suppkey
    FROM partsupp
    WHERE ps_availqty > 0.5 * (
        SELECT SUM(l_quantity)
        FROM lineitem
       WHERE l_shipdate BETWEEN '1995-01-01' AND '1995-12-31'
        AND l_partkey = partsupp.ps_partkey
        AND l_suppkey = partsupp.ps_suppkey
    )
    AND ps_partkey IN (
        SELECT p_partkey
        FROM part
        WHERE p_name LIKE '%ivory%'
    )
)
AND s_nationkey = (
    SELECT n_nationkey
    FROM nation
    WHERE n_name = 'FRANCE'
)
ORDER BY s_name ASC;
```

It can be noted that, this query still has redundant nesting even if the guidelines instructs not to do so. Therefore, it cannot be guaranteed that the LLM follows all the guidelines together at once. However, synthesis is terminated due to matching result.

7.4 Hash-based Result Comparator

The result comparator aims to efficiently verify the equality of $\mathcal{R}_{\mathcal{H}}$ and $\mathcal{R}_{\mathcal{E}}$, the results of $Q_{\mathcal{H}}$ and synthesized query respectively over D_I. The direct but slow technique is to explicitly compute, using the EXCEPT command, the difference between the results in both directions – i.e. $\mathcal{R}_{\mathcal{H}}$ EXCEPT $\mathcal{R}_{\mathcal{E}}$ and $\mathcal{R}_{\mathcal{E}}$ EXCEPT $\mathcal{R}_{\mathcal{H}}$, and verify that both are zero. XFE also takes help of the LLM to determine whether the business description specifies to order the result, and then employs the following hash-based result comparators instead. While PostgreSQL has several options for hash functions, we use the HashText hash function since it works across datatypes.

7.4.1 Global Hash Method: This method is used if either the business description specifies to order the output or $Q_{\mathcal{E}}$ has GROUP BY attributes. In such a case, considering the extracted ORDER BY vector in $Q_{\mathcal{E}}$ as $\vec{O}$, the comparator sorts $\mathcal{R}_{\mathcal{H}}$ and $\mathcal{R}_{\mathcal{E}}$ on remaining projection attributes wrt $\vec{O}$, then computes hashes on each result table. If they are equal, the results are the same.

7.4.2 Rolling Hash Method: This method is used if the text mentions no physical ordering. Here, we calculate a hash value for a relation by applying a hash function to each tuple in the relation and then aggregating the results. In the PostgreSQL database, to get the rolling hash of the tuples present in $\mathcal{R}_{\mathcal{H}}$, we use:

Select SUM(rh_hashes.hashtext)
From (Select hashtext($\mathcal{R}_{\mathcal{H}}$::TEXT) From $\mathcal{R}_{\mathcal{H}}$) as rh_hashes;

The same evaluation is done for $\mathcal{R}_{\mathcal{E}}$. Comparing the two hashes confirms or denies the unordered set equality of $\mathcal{R}_{\mathcal{H}}$ and $\mathcal{R}_{\mathcal{E}}$.

7.5 Combinatorial Synthesis of Nested Clauses

XFE keeps the previously synthesized nesting structure constant while combinatorial synthesis. It tries valid re-distributions of clause elements between the outer and inner queries one by one, until a successful outcome is reached or the candidate pool is exhausted – signaling extraction failure. In particular, the following two re-distributions are attempted within the nested structure: First, the tables in the outer and inner From clauses are redistributed, along with their associated predicates, in each synthesized candidate. Second, we know that XRE correctly identifies all the Group By attributes from the *base tables*. Further, that two layers of Group By is reasonable only when the outer layer is formed by the projections of the inner subquery. Based on this fact, we enumerate all possible candidates for outer Group By, Select, and Order By clauses. It may be noted that, such a synthesis mechanism is prone to generate a large number of candidates, depending on the tables, projections, and the predicates of the query. However, in our experiments, we observed favorable cases where the number of syntheses remains within 10.

7.5.1 Table Redistribution in From *clauses.* For a nested query, the tables in their respective From clauses are redistributed among them in each synthesized candidate. Such distributions of the tables are synthesized using the following two rules: We can either (1) *Push down* a table from the outer query inside the nesting; (2) or *drop* a table from a subquery to the outer query.

The strategies of pull-up and push-down are listed below:

- We do not change the identified nested structure.
- If any pull-up operation leaves no table inside nesting, then only we remove the nesting. Such a pull-up is permitted only if the resulting query is still a nested query, not the same flat query as the seed query.
- Table Pull-up (without duplication): Pulls a table to the outer query, thus pulls up the corresponding filter predicate also, and leaves the join as a dependent join in the inner query.
- Table Pull-up (with duplication): Adds another instance of the table to the outer query, along with the corresponding filter predicate. Therefore, the inner query has an independent join for the inner instance of the table.
- Table Push-down (with duplication): Adds another instance of a table down inside an inner query, by creating its independent new instance. Thus, it requires duplicating the filter predicate in the inner query. It converts a dependent join into an independent join.
- Table Push-down (without duplication): Pushes a table inside an inner query, along with its join and filter predicates, thus making an independent join a dependent one.
- No table displacement is done if the table does not have any extracted join predicate with any other table of the same level.
- If any table inside a nesting has aggregation in its projection, do not pull it up.
- First try to pull up (to avoid unnecessary syntactic nesting structures). Then push down.
- Continue the above possibilities until there are no more nesting structure left.

An example of such synthesis is shown below:

Select . . . From Customer
<connecting clause/operator> (Select . . . From Orders . . .)
. . .
(a) Initial nesting structure

Select . . . From Customer
<connecting clause/operator> (Select . . . From Customer, Orders . . .)
. . .
(b) Synthesis from (a) after push-down (with duplication)

Select . . . From Customer, Orders
<connecting clause/operator> (Select . . . From Orders . . .)
. . .
(c) Synthesis from (a) after pull-up (with duplication)

7.5.2 Nesting of Group By *clause.* When a nested query is formulated that has one SPJ-element (in the inner subquery) but two Group By clauses, i.e. one in the inner subquery and one in the outer, their correctness may be difficult to achieve using the LLM. To revise them correctly, we rely on the fact that, XRE correctly identifies all the Group By attributes from the *base tables*. On the other hand, two layers of Group By is reasonable only when the outer layer is formed by the projections of the inner subquery. Therefore, they are most likely to be missed in the seed query. Based on this fact, we enumerate all possible candidates for outer Group By, projection, and Order By clauses. The synthesis rules are as follows:

- The Group By attributes from the seed query are placed as the Group By attributes of the inner subquery.
- The projections $\vec{P}$ of the inner subquery are chosen to create the outer Group By clause. Their power set enumeration is taken for this purpose. i.e. each subset $G \subseteq P$ forms the outer Group By clause in a candidate synthesis. $|G|$ cannot exceed $|F|$, where $\vec{F}$ is the projection aliases in $\mathcal{R}_{\mathcal{H}}$.
- For each outer Group By clause G, the corresponding Select clause is synthesized as follows: Since $|G| < |F|$, the attributes in G are placed in the outer Select clause. The sets of size $|F - G|$ from remaining attributes $P - G$ are tried for aggregated projections. COUNT(*) is also tried in such aggregates. The aggregation functions are taken from those that are present in the incorrectly synthesized candidates by the LLM.
- If the seed query has Order By clause, then the synthesized candidates also need to adhere to it. Therefore, for each candidate of a different outer Select clause, all possible ordering of the projections are enumerated for verification.

Depending on the query complexity, the computational overheads of the above trial-and-error exercise could, in principle, be highly expensive. However, in our experiments, we found that a successful outcome was reached within a few iterations.

7.6 Equivalence Verification

The XRE-XFE pipeline may end with either what appears to be a successful extraction or a failure. In the former case, there still is the possibility of a false positive, where equivalence between the extracted query and the hidden query is incorrectly claimed. Therefore, we need to implement checks to either eliminate or, at least, reduce such possibilities. At first glance, an obvious verification mechanism is to use *logic-based* query equivalence tools (e.g. QED [36], SQLSolver [7]) to compare $Q_{\mathcal{H}}$ and $Q_{\mathcal{E}}$ – the problem here is that $Q_{\mathcal{H}}$ is not available in our framework. This non-availability also rules out use of *model-based* tools such as LLMs for evaluating equivalence.

A more practical alternative is to use *data-based* equivalence tools such as XData [5], where carefully curated databases are created that elicit differences in the results between an "instructor version" (in our case, $Q_{\mathcal{E}}$) and a potentially incorrect "student version" (in our case, $Q_{\mathcal{H}}$). However, such tools have limited scope as yet – for instance, they cannot accommodate computed column functions.

A final option is to use *result-based* equivalence tools where several randomized databases are created on which $Q_{\mathcal{H}}$ and $Q_{\mathcal{E}}$ are run. The results are compared via set difference, and a non-zero outcome indicates an extraction error. Of course, result-based equivalence is only probabilistic, and not deterministic, wrt the validity of its outcomes.

8 Experimental Results

In this section, we quantitatively evaluate the extraction performance of XPOSE. The experiments are carried out on the PostgreSQL database engine hosted on an Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz, 32 GB 2666 MHz DDR4, Ubuntu 22.04 LTS platform. The popular GPT-4o [23] LLM, configured with 0 temperature to minimize hallucinations, is the synthesis agent in the XFE module.

We present results for the following complex query suites:

1. TPCH: The standard decision support benchmark [32], which models a data warehousing environment and features 22 analytic queries labeled Q1 through Q22. The business descriptions are taken from the official documentation [32].

2. ART: A pertinent question that could be asked here is whether the extraction performance is an *artifact* of GPT-4o being previously trained on TPCH, which is publicly available. To assess this concern, we created a *different* database using the benchmark schema and data to capture *art dealing* scenario. Specifically, the mappings listed in Table 4 are used to convert a TPCH database into ART. The text descriptions also used the text mappings, as listed in Section **??**.

3. E-TPCH: Only key-based equi-joins are modeled in TPCH, and there are no Unions of sub-queries. However, as highlighted in [6, 18], many-to-many joins, non-equi-joins, and unions are commonplace in contemporary applications such as data mediators and integrators. Therefore, we have extended the basic TPCH schema and its suite of queries as follows: Union is included in some of the existing queries (by replacing LINEITEM with WEB_LINEITEM and STORE_LINEITEM tables, representing data from online and offline retail, respectively), and non-key-based joins are brought in via two new queries, Q23 and Q24, similar to those created in [6] – the full details are in Section A. This extended benchmark is referred to as E-TPCH, and its textual inputs were created by mildly augmenting the corresponding TPCH descriptions. Figure 15 lists Q7 from this query suit.

The Query finds, for two given nations, the gross discounted revenues derived from line items in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. The query lists the supplier nation, the customer nation, the year, and the revenue from shipments that took place in that year. The query orders the answer by Supplier nation, Customer nation, and year (all ascending).

```
SELECT supp_nation, cust_nation, l_year, sum(volume) as revenue
FROM (SELECT n1.n_name AS supp_nation, n2.n_name AS cust_nation,
        los.l_year AS l_year, los.volume AS volume
        FROM (
          (SELECT Extract(year FROM wl_shipdate) AS l_year,
            wl_extendedprice * ( 1 - wl_discount ) AS volume,
            s_nationkey, o_custkey FROM supplier, web_lineitem, orders
           WHERE s_suppkey = wl_suppkey AND o_orderkey = wl_orderkey
            AND wl_shipdate BETWEEN '1995-01-01' AND '1996-12-31')
          UNION ALL
          ( SELECT Extract(year FROM sl_shipdate) AS l_year,
            sl_extendedprice * ( 1 - sl_discount ) AS volume,
            s_nationkey, o_custkey FROM supplier, store_lineitem, orders
            WHERE s_suppkey = sl_suppkey AND o_orderkey = sl_orderkey
            AND sl_shipdate BETWEEN '1995-01-01' AND '1996-12-31')
        ) AS los, customer, nation n1, nation n2
        WHERE c_custkey = los.o_custkey AND los.s_nationkey = n1.n_nationkey
        AND c_nationkey = n2.n_nationkey
        AND ((n1.n_name = 'GERMANY' AND n2.n_name = 'FRANCE')
              OR (n1.n_name = 'FRANCE' AND n2.n_name = 'GERMANY'))
      ) AS shipping GROUP BYsupp_nation, cust_nation, l_year
ORDER BY supp_nation, cust_nation, l_year;
```

Figure 15: E-TPCH Q7 Business description and SQL

TPCH Table	*ART Table*	*TPCH word*	*ART word*
Customer	Patron	Line	Series
Supplier	Art Dealer	Extended	Sales
Part	Artwork	Ship	Transport
Lineitem	Painting	Clerk	Agent
Orders	Requests	Manufacturer	Artisan
Nation	Association	Comment	Remark
Region	Locality	Revenue	Turnover
Partsupp	Artwork deal	supp	deal

Table 4: Translation from TPCH to ART

4. STACK: A real-world benchmark from StackExchange [21] with 16 query templates representing questions and answers from experts. A random instance of each template is taken. Since the benchmark does not provide textual summaries, we used the LLM to create draft versions and then manually refined the descriptions.

5. OL-TPCDS: TPCDS is a decision support benchmark modeling a multi-channel large retail scenario [33] features a suite of 99 analytic queries. From this benchmark, we evaluate the one-level nesting queries within our synthesis scope, as defined in Section 7.1 – we refer to this set as OL-TPCDS. It covers a substantial fraction of the benchmark, amounting to **40** queries. Further, since the TPCDS schema has a complex structure – **3x** relations and **5x** attributes wrt

E-TPCH, we also use it to evaluate scaling of the Xpose algorithms wrt schema complexity.

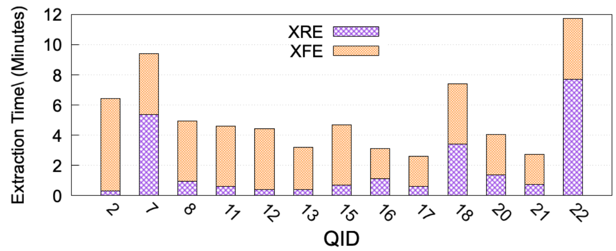
We focus here only on the benchmark queries not fully extractable by XRE or XFE in isolation – that is, where both modules had to work *together* to produce a successful extraction. With this restriction, the number of "bi-directional" queries is **13** for TPCH, **23** for E-TPCH, and **4** for STACK.

8.1 TPCH/ART Extraction

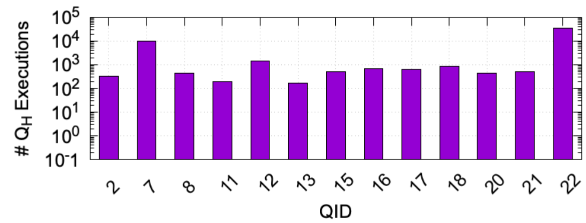
We begin by executing Xpose on encrypted (via a plugin) versions of the 13 TPCH queries. Apart from our own manual verification, the accuracy of each extraction was also checked against the techniques discussed in Section 7.6, and these results are shown in Table 5. We observe that 6 queries – **Q2, Q13, Q15, Q16, Q20, Q21** – could be successfully verified by XData [5], our best choice from a deployment perspective. For queries that were outside its scope, the Result-equivalence-based techniques were invoked and no false positives or negatives were observed. Finally, as a matter of abundant caution, we also used the logic-based tools, SQL Solver and QED, for queries in their coverage (of course, as mentioned before, these tools cannot be used in deployment due to non-availability of hidden query). We see that QED and SQL Solver additionally confirm **Q18** and **Q22**, respectively, beyond those verified by XData.

Equivalence Checker	*TPCH Query ID*
XData [5]	2, 13, 15, 16, 20, 21
QED [36]	18, 21
SQLSolver [7]	2, 21, 22

Table 5: Checkers for Extraction Accuracy (TPCH/ART)



(a) Extraction Times



(b) $Q_{\mathcal{H}}$ Invocations

Figure 16: Extraction Overheads (TPCH)

8.1.1 *Extraction Overheads.*

We now turn our attention to the time overheads incurred by the extractions. These results are shown in Figure 16(a) and we find that the extractions are typically completed in less than *ten minutes* – the sole exception is Q22, which has several disjunctions (7 each in two tables), and even this "hard-nut" query is drawn out in about 12 minutes. These overheads appear reasonable given that HQE is expected to be typically invoked in an *offline* environment.

The graph also shows the time-split across the XRE and XFE modules, and we find that the distribution is query-specific – some queries (e.g. Q7, Q22) have XRE dominating, whereas in others (e.g. Q15), XFE takes the lion's share.

The larger duration of XRE is caused by disjunction predicates, especially those with many string literals, such as in Q22. Extracting each literal requires one round of database minimization, shooting up the extraction time. On the other hand, XFE takes a longer time than XRE when the nesting structure of the query is complex. For instance, Q15 has two levels of nesting, which required multiple synthesis trials upon result mismatch.

8.1.2 *Application Invocations.*

As mentioned previously, Xpose is based on generating a curated series of input-output examples by repeated invocations of the opaque executable. To quantify this notion, the number of invocations are shown (on a log-scale) in Figure 16(b). We observe here that most of the queries take *several hundred* invocations, and a few (Q7, Q12, and Q22) go well beyond even this mark – in fact, Q22 is more than *ten thousand*!

From a conceptual perspective, these results demonstrate that (a) HQE for industrial-strength queries is a challenging problem, requiring numerous examples to achieve the goal of precision extraction, but (b) thanks to database minimization, the overheads are kept in practical check despite the numerous invocations.

8.1.3 **ART Database**.

On the ART database, Text-to-SQL encounters more mistakes than on the standard TPCH database. Only Q6 was correctly formulated by the LLM, while the remaining formulations have several mistakes that cannot be corrected only based on the result observations, as listed in Table 6. The good news is that, despite these material changes, all the ART queries continued to be extracted correctly. However, they required a couple more iterations of the clause-correction prompts to reach extraction closure.

8.1.4 *Text-to-SQL.*

To address the question as to whether contemporary Text-to-SQL tools would be directly able to extract the above queries, we also evaluate the performance of Infly-RL-SQL-32B [14]. This program is currently the highest-ranked tool (with an accessible code link) on the single-model leader-board of the BIRD benchmark [19].

We highlighted in Table 8 that Infly-RL-SQL-32B could not extract most of the E-TPCH queries. Nevertheless, a pertinent question that could still be asked is "how *far* was its output from the ground-truth"? This question is addressed in Table 7, where the errors in the incorrect extractions are enumerated. Note that there are errors across all the major query clauses, and in fact, even syntactically erroneous SQL is output for some queries.

8.2 E-TPCH Extraction

We now turn our attention to the E-TPCH query suite which features unions of sub-queries, additional nesting in the From clause, and a rich set of join types. Here, 7 extraction cases (the same

QID	*Mistake (wrt TPCH database queries)*
1	filter predicate range on `l_shipdate` (link)
2	= 'Brass' instead of LIKE '%Brass' for filter predicate on `p_type`
3	missed Customer table and related predicates
4	missed semi-join implemented by EXISTS operator
5, 7, 8	spurious Partsupp table and related joins
7	unknown filter constants on `n_name` attribute
8	missed duplicated instance of Nation
9	missed Orders table and related joins. Unknown filter predicate constant.
10	different projection order, missed `c_custkey` from projection, unknown filter constants for `o_orderdate`
11	missed Nation table, missed SUM aggregation in projection
12	incorrect constants used in the projection
13	incorrect sequence of join and grouping, incorrect projections
14	spurious Partsupp table, unknown filter constants for `l_shipdate`
15	spurious instance of Lineitem table in the outer query, missed projections
16	missed projections
17	unknown filter constants, spurious filter predicates in the inner query
18	incorrect key for semi-join, and aggregation in the inner query
19	unknown filter predicate constants
20	Spurious Customer and Orders tables and related joins, missed semi-joins
21	missed aggregate in projection, spurious HAVING clause in inner query
22	missed projection, missed filter predicates in the inner Customer table

Table 6: Text-to-SQL by GPT-4o for ART queries

Remarks on Correctness	*QID*
Correct	1, 6, 10, 14, e1
Syntactically erroneous query	2, 3, 12, 20, 21, e3, e7, e12
Missed/Incorrect projection	8, 11, 13, e13, 15, e15, 16, 21
Missed/Incorrect filter	2, 7, e7, 9, e9, 16, 19
Missed Group By	8, 13, e13, 15, e15, 16, 18, 22
Missed/Incorrect aggregation	11, 18, 22
Missed Join/Semi-join	4, e4, 5, e5, 21
Missed nested logic	2, 11, 17
Missed Union All/subquery	e3, e4, e5, e6, e9, e10, e12, e14, e15

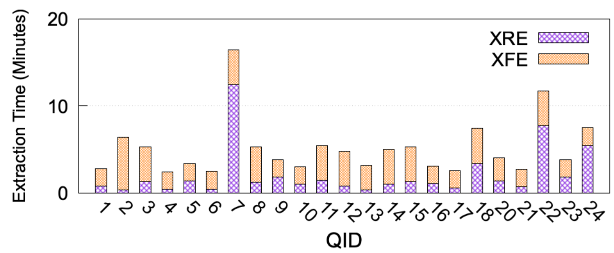
Table 7: Text-to-SQL Performance by Infly-RL-SQL-32B

Query Extraction Approach	*QID Coverage*
Unmasque [17]	1, 3, 5, 6, 10
Infly-RL-SQL-32B (Text-to-SQL) [14]	1, 6, 10, 14, e1
Xpose	All 35

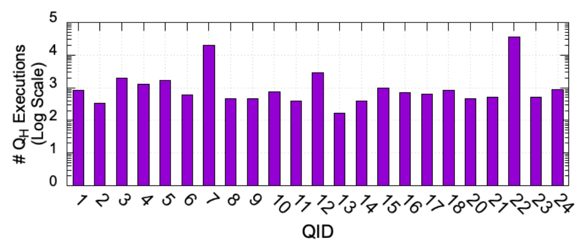
Table 8: Extraction Coverage on E-TPCH Benchmark

queries reported in Table 5) were in the scope of XData and the formal checkers, which confirmed their equivalence. For the remaining 17 "bi-directional" queries, apart from our manual verification, result-equivalence tests confirmed extraction accuracy.

8.2.1 Extraction Overheads. The extraction times for the E-TPCH queries are shown in Figure 17a. We observe that most of the queries are extracted within 5 minutes, while the remaining few are completed within 15 minutes. Again, as in TPCH, Q7 and Q22 take the maximum time to execute $Q_{\mathcal{H}}$ due to several string disjunctions and multiple instances of tables. We also observe that the time-split ratio between XRE and XFE is query-specific and covers a large range of values. Finally, with regard to invocations, shown in Figure 17b, we see here too that they are in the several hundreds to thousands, with most being higher than their corresponding avatars in TPCH due to the increased query complexities.



(a) Extraction Times



(b) $Q_{\mathcal{H}}$ Invocations

Figure 17: Extraction Overheads (E-TPCH)

8.3 Drill-down Analysis (E-TPCH)

The E-TPCH scenario foregrounds the need for bi-directional engineering, as shown in Table 9, where the work done by XRE and XFE is shown on a clause-by-clause basis. We observe that while XRE does do the majority of the work, XFE also plays a significant role in taking the extractions to closure.

8.3.1 XFE Prompts. Drilling down into XFE, the prompt sequences leading to successful extraction are shown in Table 10 on a per-query basis, along with the overall token counts. We find that all of the 20 queries shown in rows 1 and 2 of the table, are completed within 4 prompts. In the remaining 3 queries, Q20 required additional clause-corrections, whereas Q2 and Q13 proved to be "feedback-prompt-resistant" after nesting structures were synthesized by the initial prompt, requiring invocation of the computationally heavy Combinatorial Synthesis step as a last resort for extraction closure – specifically, Q2 required redistribution of tables

Clause/Operator Type	*XRE*	*XFE*
$T_{\mathcal{H}}$ (23)	23	0
Union All (11)	11	0
Semantic Preserving Join Predicates (21)	16	5
Algebraic Inequality Predicates (5)	3	2
Disjunctive attributes and literals (4)	4	0
Projection Dependencies (23)	14	9
Semantic Preserving Group By(15)	3	12

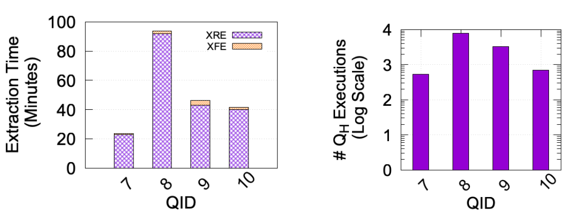
Table 9: Extraction Distribution of XRE and XFE (E-TPCH)

in the From clauses, whereas Q13 required redistribution of Group By attributes. All queries were refined with less than 4000 tokens. Given the current GPT-4o pricing of \$2.5/*million* input tokens [23], even these “hard-nut” cases cost less than a cent apiece.

Prompt Sequence	*QID*	*#Tokens*
IP, CCP, RCP	1, 4, 5, 6, 9, 10, 16, 17, 21, 23, 24	< 4k
IP, CCP, RCP, RCP	3, 7, 8, 11, 12, 14, 15, 18, 22	
IP, CCP, RCP, CCP, CCP	20	
IP, CCP, RCP, CCP, RCP, RCP, **CS**	2, 13	

Table 10: Prompt Sequences for Query Synthesis (E-TPCH)

8.3.2 Effectiveness of Guidelines. We observed that the LLM often deviates from the synthesis guidelines, especially with regard to complying with $Q_{\mathcal{S}}$ on its provable extractions. Hence, feedback prompts were generated based on the output query to forcefully instantiate guidelines. For instance, the $Q_{\mathcal{S}}$ for Q17 has tables Part, Web_Lineitem w1, Web_Lineitem w2. However, the LLM-refinement uses two instances of Part, and requires an explicit prompt (*Strictly use the tables as per* $Q_{\mathcal{S}}$*:* Part table once, Web_Lineitem table twice) to prevent it from continuing to do so. The opposite phenomenon was seen for Q24, where multi-instance tables in $Q_{\mathcal{S}}$ appeared only once in the synthesized From clause. However, despite such issues, repeated feedback prompting was sufficient to eventually resolve all 23 queries.



(a) Extraction Times (b) $Q_{\mathcal{H}}$ invocations

Figure 18: Extraction Overheads (STACK)

8.4 STACK Extraction

Our next experiment is on the 4 bi-directional STACK queries. All these extractions could be verified using XData.

The overheads incurred in the extractions are shown in Figure 18. We observe that the extraction times are now much larger, with Q8 going up to almost two hours. The reason for this extended time is that the STACK database is large (~100 GB) and therefore minimization itself, rather than extraction per se, becomes an expensive exercise. Moreover, multiple rounds of such minimization are required to extract all the disjunction predicates (there are 5 constants in Q8’s IN predicate).

We also observe that in this benchmark, virtually all the time is taken XRE relative to XFE. This is because XRE handles the self-joins and disjunctions which take the primary computational effort. Whereas XFE’s role is confined to nested structures connected by existential operators, which were found within a couple of feedback prompting iterations.

8.5 OL-TPCDS Extraction

We now turn our attention to the OL-TPCDS benchmark, featuring 40 queries with up to one-level of nesting.

8.5.1 Extraction Correctness. We found that 31 of the 40 queries were successfully extracted by Xpose. Of these, 17 were completely in-scope for XRE, while the remaining 14 required bidirectional engineering. The 9 failure cases arose primarily due to presence of specialized constructs such as self-joins and semi-joins. This might appear to be a limitation of Xpose, but we found that, in fact, the failures were due to *inadequate and ambiguous text* in the TPCDS documentation. By just adding *one* additional line to each TX_Q, all these problematic queries were accurately extracted.

8.5.2 Extraction Scalability. Moving on to a performance perspective, the ability of Xpose to extract the OL-TPCDS queries demonstrates its scalability wrt schema complexity. For instance, Q17 and Q25 feature 10 joins apiece, while Q72 has as many as 13 joins. Despite these sizeable join-graphs, they were all extracted in less than 5 minutes. This efficiency is thanks to the D^1-based algebraic predicate extractor.

When we consider Union-based queries, recall that Algorithm 1 has, in principle, a power-set dependency on $|\texttt{AuxTables}(Q_{\mathcal{H}})|$. So, we took Q71 which has a union of 3 subqueries with 3 tables each in its original version, and created an artificially inflated query with 13 subqueries, each having as many as 6 tables. Despite this steep jump in breadth and depth, the Union module completed its extraction in less than 10 minutes! The surprising efficiency is due to the strong pruning of the candidate space (Section 4.3.4), which ensured that on average, only about 30 percent of the power set was actually evaluated, with the median being just 19 percent. Note that the union complexity we modeled here far exceeds the maximum complexity in the TPCDS benchmark, which is 6.

9 Related Work

Classical query reverse engineering (QRE) [34], [35], [15], [30, 31], [24], [29], [40], [4] uses only query synthesis, and hints on the ground truth [4]. They are instances of the program-by-example paradigm [13] hosted in the SQL-world. However, due to the large enumerated search space, the scalability of such systems is weak [17]. Moreover, synthesizing a query that matches the user intention (i.e. business logic) is not achieved in most cases [4], and requires significant human interaction. Lastly, synthesizing complex OLAP queries such as the TPCH benchmark, has not been previously achieved in the literature.

Query forward engineering is gaining pace in automation using the contemporary technology of LLM-based Text-to-SQL tools [1]. However, the reach of such tools is limited to simple cases [8, 10, 22]. Moreover, they require unambiguous text, and expect restricted schema structures [10]. The Text-to-SQL benchmarks such as Bird [2] and Spider [39] comprise queries that correspond to user questions posed to get immediate answers from the database, i.e. they lack complexity. In fact, there is recent evidence that applying such tools to engineer OLAP queries is unlikely to be successful [20], and it is concluded that a human-in-the-loop workflow is required when AI is used as the synthesis agent. Our own experience also bears this out – only 5 TPCH benchmark queries were correctly formulated by GPT-4o [23] directly from their respective business descriptions, despite prior training on the same benchmark.

10 The Road Ahead

At this stage, we have shown that it is indeed feasible, with a judicious combination of reverse and forward engineering, to extract substantively complex hidden SQL queries. But there remains ample opportunities for taking these ideas further, as discussed below.

Multi-level Nesting. Our current extraction scope primarily covers queries with two levels of nesting. This is mainly due to GPT-4o's bias toward synthesizing unnested queries. Moreover, business logic requiring more nesting levels are often abstract, as in TPC-DS, which hampers XFE in deriving such structures. We are exploring whether it would be feasible to extend XRE from its current flat avatar to handle at least one level of nesting, thereby enhancing Xpose's overall scope for multi-level query hierarchies.

Multi-block Queries. Our focus here was on single-block queries, but many queries in the TPC-DS benchmark [33] pose novel challenges to Xpose. This is because they have multi-block structures constructed via multiple CTEs, instantiated many times, which makes identifying their signatures difficult as compared to base tables. Further, we found their high-level descriptions, as per the official documentation, to be insufficient for XFE to recognize the block boundaries. In our future work, we intend to study these issues with new strategies – for instance, chain-of-thought prompting [37] may help to address the identification of block boundaries.

Multi-query Applications. Thus far, we had only discussed applications with monolithic SQL queries embedded within them. But enterprise applications may comprise several queries. Xpose can be used in such multi-query scenarios if the individual queries are wrapped within separate functions [17]. In practice, modular application codebases typically have such wrappers. Moreover, using utilities such as *GDB* [11] or *objdump* [12], individual functions can be isolated from the application binary [28]. They can then be independently executed from their machine codes [9, 27]. Therefore, we expect that Xpose could be made viable for such real-world black-box applications.

11 Conclusions

Hidden Query Extraction is a QRE variant with several industrial use-cases. We presented Xpose, a non-invasive system for extracting hidden queries of the complexity seen in popular OLAP benchmarks. Xpose features a unique bi-directional engineering architecture, wherein forward engineering establishes the query skeleton while reverse engineering fleshes out the details. Moreover, our evaluation showed that many queries could only be extracted by harnessing together their complementary abilities. We also found that HQE for industrial-strength queries is a complex learning task requiring powerful LLM models and reasoning power.

The XRE module deterministically extracts the core operators of the opaque application to generate a rich seed query featuring unions, algebraic predicates and disjunctions. Subsequently, XFE uses an LLM and combinatorial query synthesis to refine the seed to match the hidden query. The extraction overheads were found to be reasonable for modestly-sized database instances, and in our future work, we plan to look into scalability improvements.

References

[1] Text-to-SQL - Papers With Code. `https://paperswithcode.com/task/text-to-sql`, 2024. Accessed: 2024-12-16.

[2] Bird-Bench: A Benchmark for Bird Image Recognition. `https://bird-bench.github.io/`, 2025. Accessed: 2025-01-10.

[3] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., USA, 1st edition, 2001.

[4] R. Brancas, M. Terra-Neves, M. Ventura, V. Manquinho, and R. Martins. Towards reliable SQL synthesis: Fuzzing-based evaluation and disambiguation. In D. Beyer and A. Cavalcanti, editors, *Fundamental Approaches to Software Engineering - 27th International Conference, FASE 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings*, volume 14573 of *Lecture Notes in Computer Science*, pages 232–254. Springer, 2024.

[5] B. Chandra, B. Chawda, S. Shah, S. Sudarshan, and A. Shah. Extending xdata to kill SQL query mutants in the wild. In V. R. Narasayya and N. Polyzotis, editors, *Proceedings of the Sixth International Workshop on Testing Database Systems, DBTest 2013, New York, NY, USA, June 24, 2013*, pages 2:1–2:6. ACM, 2013.

[6] B. Ding, S. Chaudhuri, J. Gehrke, and V. R. Narasayya. DSB: A decision support benchmark for workload-driven and traditional database systems. *Proc. VLDB Endow.*, 14(13):3376–3388, 2021.

[7] H. Ding, Z. Wang, Y. Yang, D. Zhang, Z. Xu, H. Chen, R. Piskac, and J. Li. Proving query equivalence using linear integer arithmetic. *Proc. ACM Manag. Data*, 1(4), Dec 2023.

[8] A. Floratou, F. Psallidas, F. Zhao, S. Deep, G. Hagleither, W. Tan, J. Cahoon, R. Alotaibi, J. Henkel, A. Singla, A. V. Grootel, B. Chow, K. Deng, K. Lin, M. Campos, K. V. Emani, V. Pandit, V. Shnayder, W. Wang, and C. Curino. NL2SQL is a solved problem... not! In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org, 2024.

[9] Frodox. Execute Machine Code from Memory - GitHub Repository. `"https://github.com/Frodox/execute-machine-code-from-memory/blob/master/README.md"`, 2025. Accessed: 2025-03-11.

[10] J. Fürst, C. Kosten, F. Nooralahzadeh, Y. Zhang, and K. Stockinger. Evaluating the data model robustness of text-to-sql systems based on real user queries. In A. Simitsis, B. Kemme, A. Queralt, O. Romero, and P. Jovanovic, editors, *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 158–170. OpenProceedings.org, 2025.

[11] GNU Project. *GDB: The GNU Project Debugger*, 2024. Accessed: 2025-03-11.

[12] GNU Project. *objdump - GNU Binary Utilities*, 2024. Accessed: 2025-03-11.

[13] S. Gulwani and P. Jain. Programming by Examples: PL meets ML. In B. E. Chang, editor, *Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings*, volume 10695 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2017.

[14] infly. inf-rl-qwen-coder-32b-2746. `https://huggingface.co/infly/inf-rl-qwen-coder-32b-2746`, 2025. Accessed: 2025-06-01.

[15] D. V. Kalashnikov, L. V. S. Lakshmanan, and D. Srivastava. FastQRE: Fast Query Reverse Engineering. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 337–350. ACM, 2018.

[16] K. Khurana and J. R. Haritsa. UNMASQUE: A hidden SQL query extractor. *Proc. VLDB Endow.*, 13(12):2809–2812, 2020.

[17] K. Khurana and J. R. Haritsa. Shedding light on opaque application queries. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 912–924. ACM, 2021.

[18] M. Lenzerini. Data integration: A theoretical perspective. In L. Popa, S. Abiteboul, and P. G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002.

[19] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024. `https://bird-bench.github.io/`.

[20] L. Ma, K. Pu, Y. Zhu, and W. Taylor. Comparing large language models for generating complex queries. *Journal of Computer and Communications*, 13(2):236–249, 2025.

[21] R. Marcus. Stack dataset. `"https://rmarcus.info/stack.html"`, 2025. Accessed: 2025-03-11.

[22] A. Mitsopoulou and G. Koutrika. Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities. In A. Simitsis, B. Kemme, A. Queralt, O. Romero, and P. Jovanovic, editors, *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 199–212. OpenProceedings.org, 2025.

[23] OpenAI. ChatGPT: Language Model for Conversational AI. `https://chat.openai.com`, 2024. Accessed: 2024-12-19.

[24] P. Orvalho, M. Terra-Neves, M. Ventura, R. Martins, and V. M. Manquinho. SQUARES : A SQL synthesizer using query reverse engineering. *Proc. VLDB Endow.*, 13(12):2853–2856, 2020.

[25] Software Heritage. Software is fragile: why we need a universal source code archive. Accessed: 2025-02-24.

[26] Stack Exchange Community. How serious is losing the source code? `"https://softwareengineering.stackexchange.com/questions/13614/how-serious-is-losing-the-source-code"`, 2024. Accessed: 2025-03-20.

[27] Stack Overflow Community. Executing machine code in memory. `"https://stackoverflow.com/questions/2019923/executing-machine-code-in-memory"`, 2025. Accessed: 2025-03-11.

[28] Stack Overflow Community. How to extract a few functions out of a compiled ELF executable (no disassembly)? `"https://stackoverflow.com/questions/14290684/how-to-extract-a-few-functions-out-of-a-compiled-elf-executable-no-disassembly"`, 2025. Accessed: 2025-03-11.

[29] K. Takenouchi, T. Ishio, J. Okada, and Y. Sakata. PATSQL: efficient synthesis of SQL queries from example tables with quick inference of projected columns. *Proc. VLDB Endow.*, 14(11):1937–1949, 2021.

[30] W. C. Tan. Efficient Query Reverse Engineering for Joins and OLAP-Style Aggregations. In Y. Cai, Y. Ishikawa, and J. Xu, editors, *Web and Big Data - Second International Joint Conference, APWeb-WAIM 2018, Macau, China, July 23-25, 2018, Proceedings, Part II*, volume 10988 of *Lecture Notes in Computer Science*, pages 53–62. Springer, 2018.

[31] W. C. Tan, M. Zhang, H. Elmeleegy, and D. Srivastava. REGAL+: reverse engineering SPJA queries. *Proc. VLDB Endow.*, 11(12):1982–1985, 2018.

[32] The Transaction Processing Performance Council. TPC Benchmark™ H (TPC-H). In *TPC Benchmark H (Decision Support)*, 2005.

[33] The Transaction Processing Performance Council. TPC Benchmark™ DS (TPC-DS). In *TPC Benchmark DS (Decision Support)*, 2006.

[34] Q. T. Tran, C. Y. Chan, and S. Parthasarathy. Query reverse engineering. *VLDB J.*, 23(5):721–746, 2014.

[35] C. Wang, A. Cheung, and R. Bodík. Synthesizing highly expressive SQL queries from input-output examples. In A. Cohen and M. T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 452–466. ACM, 2017.

[36] S. Wang, S. Pan, and A. Cheung. QED: A powerful query equivalence decider for SQL. *Proc. VLDB Endow.*, 17(11):3602–3614, 2024.

[37] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[38] F. Yu, W. Hou, C. Luo, D. Che, and M. Zhu. CS2: a new database synopsis for query estimation. In K. A. Ross, D. Srivastava, and D. Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 469–480. ACM, 2013.

[39] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics, 2018.

[40] X. Zhou, R. Bodík, A. Cheung, and C. Wang. Synthesizing analytical SQL queries from computation demonstration. In R. Jhala and I. Dillig, editors, *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, pages 168–182. ACM, 2022.

A E-TPCH Queries

A.1 E-TPCH Q1

A.1.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        (select wl_returnflag as l_returnflag,
        wl_linestatus as l_linestatus,
        wl_quantity as l_quantity,
        wl_extendedprice as l_extendedprice,
        wl_discount as l_discount,
        wl_tax as l_tax
        from web_lineitem where wl_shipdate <= date '1998-12-01' - interval '3' day
        UNION ALL
        select sl_returnflag as l_returnflag,
        sl_linestatus as l_linestatus,
        sl_quantity as l_quantity,
        sl_extendedprice as l_extendedprice,
        sl_discount as l_discount,
        sl_tax as l_tax
        from store_lineitem where sl_shipdate <= date '1998-12-01' - interval '3' day
        ) as lineitem
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

A.1.2 Seed Query produced by XRE.

```
(select
        wl_returnflag,
        wl_linestatus,
        sum(wl_quantity) as sum_qty,
        sum(wl_extendedprice) as sum_base_price,
        sum(wl_extendedprice * (1 - wl_discount)) as sum_disc_price,
        sum(wl_extendedprice * (1 - wl_discount) * (1 + wl_tax)) as sum_charge,
        avg(wl_quantity) as avg_qty,
        avg(wl_extendedprice) as avg_price,
        avg(wl_discount) as avg_disc,
        count(*) as count_order
from web_lineitem where wl_shipdate <= date '1998-11-28'
group by
        wl_returnflag,
        wl_linestatus
order by
        wl_returnflag,
```

```
        wl_linestatus)
UNION ALL
(select
        sl_returnflag,
        sl_linestatus,
        sum(sl_quantity) as sum_qty,
        sum(sl_extendedprice) as sum_base_price,
        sum(sl_extendedprice * (1 - sl_discount)) as sum_disc_price,
        sum(sl_extendedprice * (1 - sl_discount) * (1 + sl_tax)) as sum_charge,
        avg(sl_quantity) as avg_qty,
        avg(sl_extendedprice) as avg_price,
        avg(sl_discount) as avg_disc,
        count(*) as count_order
from store_lineitem where sl_shipdate <= date '1998-11-28'
group by
        sl_returnflag,
        sl_linestatus
order by
        sl_returnflag,
        sl_linestatus);
```

A.1.3 **Business Description Text.** The Query provides a summary pricing report for all lineitems shipped as of a given date. The date is within 3 days of the greatest ship date contained in the database. The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of lineitems in each group is included. 1998-12-01 is the highest possible ship date as defined in the database population.

A.1.4 **Output of XPOSE.**

```
SELECT
    returnflag,
    linestatus,
    SUM(sum_qty) AS sum_qty,
    SUM(sum_base_price) AS sum_base_price,
    SUM(sum_disc_price) AS sum_disc_price,
    SUM(sum_charge) AS sum_charge,
    AVG(avg_qty) AS avg_qty,
    AVG(avg_price) AS avg_price,
    AVG(avg_disc) AS avg_disc,
    SUM(count_order) AS count_order
FROM (
    SELECT
    wl_returnflag as returnflag,
    wl_linestatus as linestatus,
    SUM(wl_quantity) AS sum_qty,
    SUM(wl_extendedprice) AS sum_base_price,
    SUM(wl_extendedprice * (1 - wl_discount)) AS sum_disc_price,
    SUM(wl_extendedprice * (1 - wl_discount) * (1 + wl_tax)) AS sum_charge,
    AVG(wl_quantity) AS avg_qty,
    AVG(wl_extendedprice) AS avg_price,
    AVG(wl_discount) AS avg_disc,
    COUNT(*) AS count_order
    FROM web_lineitem
    WHERE wl_shipdate <= DATE '1998-11-28'
    GROUP BY
    wl_returnflag,
    wl_linestatus
    UNION ALL
```

```
    SELECT
    sl_returnflag as returnflag,
    sl_linestatus as linestatus,
    SUM(sl_quantity) AS sum_qty,
    SUM(sl_extendedprice) AS sum_base_price,
    SUM(sl_extendedprice * (1 - sl_discount)) AS sum_disc_price,
    SUM(sl_extendedprice * (1 - sl_discount) * (1 + tax)) AS sum_charge,
    AVG(sl_quantity) AS avg_qty,
    AVG(sl_extendedprice) AS avg_price,
    AVG(sl_discount) AS avg_disc,
    COUNT(*) AS count_order
    FROM store_lineitem
    WHERE sl_shipdate <= DATE '1998-11-28'
    GROUP BY
    sl_returnflag,
    sl_linestatus
) AS combined
GROUP BY
    returnflag,
    linestatus
ORDER BY
    returnflag,
    linestatus;
```

A.2 E-TPCH Q2

A.2.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from
        part,
        supplier,
        partsupp,
        nation,
        region
where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = 15
        and p_type like '%BRASS'
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'EUROPE'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp,
                        supplier,
                        nation,
```

```
                        region
                where
                        p_partkey = ps_partkey
                        and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey
                        and n_regionkey = r_regionkey
                        and r_name = 'EUROPE'
        )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey limit 100;
```

A.2.2 Seed Query produced by XRE.

```
 Select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
 From nation, part, partsupp, region, supplier
 Where nation.n_nationkey = supplier.s_nationkey
 and nation.n_regionkey = region.r_regionkey
 and part.p_partkey = partsupp.ps_partkey
 and partsupp.ps_suppkey = supplier.s_suppkey
 and part.p_size = 15
 and region.r_name = 'EUROPE'
 and part.p_type LIKE '%BRASS'
 Order By s_acctbal desc, n_name asc, s_name asc, p_partkey asc
Limit 100;
```

A.2.3 **Business Description Text.** The Query finds, in Europe, for each part made of Brass and of size 15, the supplier who can supply it at minimum cost. If several European suppliers offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances. For each supplier, the query lists the supplier's account balance, name and nation; the part's number and manufacturer; the supplier's address, phone number and comment information.

A.2.4 Output of Xpose.

```
SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM nation, part, partsupp, region, supplier
WHERE nation.n_nationkey = supplier.s_nationkey
  AND nation.n_regionkey = region.r_regionkey
  AND part.p_partkey = partsupp.ps_partkey
  AND partsupp.ps_suppkey = supplier.s_suppkey
  AND part.p_size = 15
  AND region.r_name = 'EUROPE'
  AND part.p_type LIKE '%BRASS'
  AND partsupp.ps_supplycost = (
    SELECT MIN(ps_supplycost)
    FROM nation n, region r, partsupp ps, supplier s
    WHERE ps.ps_partkey = part.p_partkey
    AND ps.ps_suppkey = s.s_suppkey
    AND s.s_nationkey = n.n_nationkey
    AND n.r_regionkey = r.r_regionkey
    AND r.r_name = 'EUROPE'
  )
ORDER BY s_acctbal DESC, n_name ASC, s_name ASC, p_partkey ASC
LIMIT 100;
```

A.3 E-TPCH Q3

A.3.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
 WITH combined_data AS (
```

```
    (SELECT
        wl_orderkey AS orderkey,
        wl_extendedprice * (1 - wl_discount) AS l_discounted_price,
        o_orderdate,
        o_shippriority
    FROM
        customer
    JOIN orders ON c_custkey = o_custkey
    JOIN web_lineitem ON wl_orderkey = o_orderkey
    WHERE
        c_mktsegment = 'FURNITURE'
        AND o_orderdate < DATE '1995-01-01'
        AND wl_shipdate > DATE '1995-01-01')

    UNION ALL

    (SELECT
        sl_orderkey AS orderkey,
        sl_extendedprice * (1 - sl_discount) AS l_discounted_price,
        o_orderdate,
        o_shippriority
    FROM
        customer
    JOIN orders ON c_custkey = o_custkey
    JOIN store_lineitem ON sl_orderkey = o_orderkey
    WHERE
        c_mktsegment = 'FURNITURE'
        AND o_orderdate < DATE '1995-01-01'
        AND sl_shipdate > DATE '1995-01-01'
    )
)
SELECT
    o_shippriority,
    SUM(l_discounted_price) AS revenue,
FROM
    combined_data GROUP BY
        orderkey, o_orderdate, o_shippriority
ORDER BY
    revenue DESC
LIMIT 10;
```

A.3.2 Seed Query produced by XRE.

```
(select wl_orderkey, sum(wl_extendedprice * (1 - wl_discount)) as revenue, o_orderdate, o_shippriority
from customer, orders, web_lineitem
where customer.c_mktsegment = 'FURNITURE'
and customer.c_custkey = orders.o_custkey
and web_lineitem.wl_orderkey = orders.o_orderkey
and orders.o_orderdate <= date '1994-12-31'
and web_lineitem.wl_shipdate >= date '1995-01-02'
group by wl_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate limit 10)
UNION ALL
(select sl_orderkey, sum(sl_extendedprice * (1 - sl_discount)) as revenue, o_orderdate, o_shippriority
from customer, orders, store_lineitem
where customer.c_mktsegment = 'FURNITURE'
and customer.c_custkey = orders.o_custkey
and store_lineitem.sl_orderkey = orders.o_orderkey
```

```
and orders.o_orderdate <= date '1994-12-31'
and store_lineitem.sl_shipdate >= date '1995-01-02'
group by sl_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate limit 10);
```

A.3.3 **Business Description Text.** The Query retrieves the shipping priority and potential revenue, defined as the sum of extended price * (1-discount), of the orders having the largest revenue among those that had not been shipped as of a given date. Orders are listed in decreasing order of revenue. If more than 10 unshipped orders exist, only the 10 orders with the largest revenue are listed.

A.3.4 **Output of Xpose.**

```
SELECT
    orderkey,
    SUM(extendedprice * (1 - discount)) AS revenue,
    o_orderdate,
    o_shippriority
FROM (
    SELECT
        wl_orderkey AS orderkey,
        wl_extendedprice AS extendedprice,
        wl_discount AS discount,
        o_orderdate,
        o_shippriority
    FROM
        customer,
        orders,
        web_lineitem
    WHERE
        c_mktsegment = 'FURNITURE'
        AND c_custkey = o_custkey
        AND wl_orderkey = o_orderkey
        AND o_orderdate <= DATE '1994-12-31'
        AND wl_shipdate >= DATE '1995-01-02'
    UNION ALL
    SELECT
        sl_orderkey AS orderkey,
        sl_extendedprice AS extendedprice,
        sl_discount AS discount,
        o_orderdate,
        o_shippriority
    FROM
        customer,
        orders,
        store_lineitem
    WHERE
        c_mktsegment = 'FURNITURE'
        AND c_custkey = o_custkey
        AND sl_orderkey = o_orderkey
        AND o_orderdate <= DATE '1994-12-31'
        AND sl_shipdate >= DATE '1995-01-02'
) AS combined_orders
GROUP BY
    orderkey,
    o_orderdate,
    o_shippriority
ORDER BY
    revenue DESC
LIMIT 10;
```

A.4 E-TPCH Q4

A.4.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
SELECT o_orderpriority,
       Count(*) AS order_count
FROM   orders
WHERE  o_orderdate >= DATE '1995-01-01'
AND    o_orderdate <  DATE '1995-01-01' + interval '3' month
AND    EXISTS
       (
              SELECT *
              FROM   (
                     (
                            SELECT sl_commitdate  AS l_commitdate,
                                   sl_receiptdate AS l_receiptdate,
                                   sl_orderkey    AS l_orderkey
                            FROM   store_lineitem)
              UNION ALL
                        (
                               SELECT wl_commitdate  AS l_commitdate,
                                      wl_receiptdate AS l_receiptdate,
                                      wl_orderkey    AS l_orderkey
                               FROM   web_lineitem)) AS lineitem
           WHERE     l_orderkey = o_orderkey
           AND       l_commitdate < l_receiptdate) GROUP BY o_orderpriority ORDER BY o_orderpriority;
```

A.4.2 Seed Query produced by XRE.

```
(Select o_orderpriority, Count(*) as order_count
 From orders, web_lineitem
 Where orders.o_orderkey = web_lineitem.wl_orderkey
 and web_lineitem.wl_commitdate < web_lineitem.wl_receiptdate
 and orders.o_orderdate between '1995-01-01' and '1995-03-31'
 Group By o_orderpriority
 Order By o_orderpriority asc)
 UNION ALL
 (Select o_orderpriority, Count(*) as order_count
 From orders, store_lineitem
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and store_lineitem.sl_commitdate < store_lineitem.sl_receiptdate
 and orders.o_orderdate between '1995-01-01' and '1995-03-31'
 Group By o_orderpriority
 Order By o_orderpriority asc);
```

A.4.3 **Business Description Text.** The Query counts the number of orders ordered in a given quarter of 1995 in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority sorted in ascending priority order.

A.4.4 Output of Xpose.

```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM (
    SELECT o_orderpriority, orders.o_orderkey
    FROM orders
    WHERE orders.o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'
    AND orders.o_orderkey IN (
        SELECT wl_orderkey
        FROM web_lineitem
        WHERE wl_commitdate < wl_receiptdate
    )
```

```
    UNION
    SELECT o_orderpriority, orders.o_orderkey
    FROM orders
    WHERE orders.o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'
    AND orders.o_orderkey IN (
        SELECT sl_orderkey
        FROM store_lineitem
        WHERE sl_commitdate < sl_receiptdate
    )
) AS combined
GROUP BY o_orderpriority
ORDER BY o_orderpriority ASC;
```

A.5 E-TPCH Q5

A.5.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        n_name,
        sum(los.l_extendedprice * (1 - los.l_discount)) as revenue
from
        customer,
        ( select
            wl_extendedprice as l_extendedprice,
            wl_discount as l_discount,
            wl_suppkey as l_suppkey,
            wl_orderkey as l_orderkey,
            s_nationkey,
            o_custkey
        from web_lineitem, orders, supplier
        where
            o_orderdate >= date '1995-01-01'
            and o_orderdate < date '1995-01-01' + interval '1' year
            and wl_orderkey = o_orderkey
            and wl_suppkey = s_suppkey
        UNION ALL
         select
            sl_extendedprice as l_extendedprice,
            sl_discount as l_discount,
            sl_suppkey as l_suppkey,
            sl_orderkey as l_orderkey,
            s_nationkey,
            o_custkey
        from store_lineitem, orders, supplier
        where
            o_orderdate >= date '1995-01-01'
            and o_orderdate < date '1995-01-01' + interval '1' year
            and sl_orderkey = o_orderkey
            and sl_suppkey = s_suppkey
        ) as los,
        nation,
        region
where
        c_custkey = los.o_custkey
        and c_nationkey = los.s_nationkey
        and los.s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'ASIA'
```

```
group by
        n_name
order by
        revenue desc;
```

A.5.2 **Seed Query produced by XRE.**

```
(Select n_name, Sum(wl_extendedprice*(1 - wl_discount)) as revenue
 From customer, nation, orders, region, supplier, web_lineitem
 Where customer.c_custkey = orders.o_custkey
 and customer.c_nationkey = nation.n_nationkey
 and nation.n_nationkey = supplier.s_nationkey
 and orders.o_orderkey = web_lineitem.wl_orderkey
 and nation.n_regionkey = region.r_regionkey
 and supplier.s_suppkey = web_lineitem.wl_suppkey
 and region.r_name = 'ASIA'
 and orders.o_orderdate between '1995-01-01' and '1995-12-31'
 Group By n_name
 Order By revenue desc, n_name asc)
 UNION ALL
 (Select n_name, Sum(sl_extendedprice*(1 - sl_discount)) as revenue
 From customer, nation, orders, region, store_lineitem, supplier
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and store_lineitem.sl_suppkey = supplier.s_suppkey
 and customer.c_custkey = orders.o_custkey
 and customer.c_nationkey = nation.n_nationkey
 and nation.n_nationkey = supplier.s_nationkey
 and nation.n_regionkey = region.r_regionkey
 and region.r_name = 'ASIA'
 and orders.o_orderdate between '1995-01-01' and '1995-12-31'
 Group By n_name
 Order By revenue desc, n_name asc);
```

A.5.3 **Business Description Text.** The Query lists for each nation in Asia the revenue volume that resulted from line item transactions in which the customer ordering parts and the supplier filling them were both within that nation. The query is run in order to determine whether to institute local distribution centers in a given region. The query considers only parts ordered in the year 1995. The query displays the nations and revenue volume in descending order by revenue. Revenue volume for all qualifying line items in a particular nation is defined as sum(extendedprice * (1 - discount)).

A.5.4 **Output of XPOSE.**

```
SELECT n_name, SUM(revenue) AS revenue
FROM (
    SELECT n_name, wl_extendedprice * (1 - wl_discount) AS revenue
    FROM customer, nation, orders, region, supplier, web_lineitem
    WHERE customer.c_custkey = orders.o_custkey
    AND customer.c_nationkey = nation.n_nationkey
    AND nation.n_nationkey = supplier.s_nationkey
    AND orders.o_orderkey = web_lineitem.wl_orderkey
    AND nation.n_regionkey = region.r_regionkey
    AND supplier.s_suppkey = web_lineitem.wl_suppkey
    AND region.r_name = 'ASIA'
    AND orders.o_orderdate BETWEEN '1995-01-01' AND '1995-12-31'
    UNION ALL
    SELECT n_name, sl_extendedprice * (1 - sl_discount) AS revenue
    FROM customer, nation, orders, region, store_lineitem, supplier
    WHERE orders.o_orderkey = store_lineitem.sl_orderkey
    AND store_lineitem.sl_suppkey = supplier.s_suppkey
    AND customer.c_custkey = orders.o_custkey
    AND customer.c_nationkey = nation.n_nationkey
```

```
    AND nation.n_nationkey = supplier.s_nationkey
    AND nation.n_regionkey = region.r_regionkey
    AND region.r_name = 'ASIA'
    AND orders.o_orderdate BETWEEN '1995-01-01' AND '1995-12-31'
) AS combined
GROUP BY n_name
ORDER BY revenue DESC, n_name ASC;
```

A.6 E-TPCH Q6

A.6.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        sum(lineitem.l_extendedprice *(1 - lineitem.l_discount)) as revenue
from
        (select wl_extendedprice as l_extendedprice,
        wl_discount as l_discount
        from web_lineitem
        where wl_shipdate >= date '1993-01-01'
        and wl_shipdate < date '1994-03-01' + interval '1' year
        and wl_discount between 0.06 - 0.01 and 0.06 + 0.01
        and wl_quantity < 24
        UNION ALL
        select sl_extendedprice as l_extendedprice,
        sl_discount as l_discount
        from store_lineitem
        where sl_shipdate >= date '1993-01-01'
        and sl_shipdate < date '1994-03-01' + interval '1' year
        and sl_discount between 0.06 - 0.01 and 0.06 + 0.01
        and sl_quantity < 24) as lineitem;
```

A.6.2 Seed Query produced by XRE.

```
select sum(wl_extendedprice*wl_discount) as revenue
        from web_lineitem
        where web_lineitem.wl_shipdate >= date '1993-01-01'
        and web_lineitem.wl_shipdate <= date '1995-02-28'
        and web_lineitem.wl_discount between 0.05 and 0.07
        and web_lineitem.wl_quantity <= 23.99
        UNION ALL
        select sum(sl_extendedprice*sl_discount) as revenue
        from store_lineitem
        where store_lineitem.sl_shipdate >= date '1993-01-01'
        and store_lineitem.sl_shipdate <= date '1995-02-28'
        and store_lineitem.sl_discount between 0.05 and 0.07
        and store_lineitem.sl_quantity < 23.99;
```

A.6.3 **Business Description Text.** The Query considers all the line items shipped in a given period, with discounts between 0.05 and 0.07. The query lists the amount by which the total revenue would have increased if these discounts had been eliminated for line items with quantity less than 24. Note that the potential revenue increase is equal to the sum of [extendedprice * discount] for all line items with discounts and quantities in the qualifying range.

A.6.4 Output of XPOSE.

```
SELECT SUM(revenue) AS revenue
FROM (
    SELECT wl_extendedprice * wl_discount AS revenue
    FROM web_lineitem
    WHERE wl_shipdate >= DATE '1993-01-01'
    AND wl_shipdate <= DATE '1995-02-28'
    AND wl_discount BETWEEN 0.05 AND 0.07
```

```
    AND wl_quantity < 24
    UNION ALL
    SELECT sl_extendedprice * sl_discount AS revenue
    FROM store_lineitem
    WHERE sl_shipdate >= DATE '1993-01-01'
    AND sl_shipdate <= DATE '1995-02-28'
    AND sl_discount BETWEEN 0.05 AND 0.07
    AND sl_quantity < 24
) AS combined_revenue;
```

A.7 E-TPCH Q7

A.7.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
 SELECT supp_nation,
       cust_nation,
       l_year,
       SUM(volume) AS revenue
FROM   (SELECT n1.n_name  AS supp_nation,
               n2.n_name  AS cust_nation,
               los.l_year AS l_year,
               los.volume AS volume
        FROM   (SELECT Extract(year FROM wl_shipdate)         AS l_year,
                       wl_extendedprice * ( 1 - wl_discount ) AS volume,
                       s_nationkey,
                       o_custkey
                FROM   supplier,
                       web_lineitem,
                       orders
                WHERE  s_suppkey = wl_suppkey
                       AND o_orderkey = wl_orderkey
                       AND wl_shipdate BETWEEN DATE '1995-01-01' AND DATE
                                               '1996-12-31'
                UNION ALL
                SELECT Extract(year FROM sl_shipdate)         AS l_year,
                       sl_extendedprice * ( 1 - sl_discount ) AS volume,
                       s_nationkey,
                       o_custkey
                FROM   supplier,
                       store_lineitem,
                       orders
                WHERE  s_suppkey = sl_suppkey
                       AND o_orderkey = sl_orderkey
                       AND sl_shipdate BETWEEN DATE '1995-01-01' AND DATE
                                               '1996-12-31')
               AS los,
               customer,
               nation n1,
               nation n2
        WHERE  c_custkey = los.o_custkey
               AND los.s_nationkey = n1.n_nationkey
               AND c_nationkey = n2.n_nationkey
               AND ( ( n1.n_name = 'GERMANY'
                       AND n2.n_name = 'FRANCE' )
                      OR ( n1.n_name = 'FRANCE'
                           AND n2.n_name = 'GERMANY' ) )) AS shipping
GROUP  BY supp_nation,
          cust_nation,
```

```
         l_year
ORDER  BY supp_nation,
         cust_nation,
         l_year;
```

A.7.2 Seed Query produced by XRE.

```
(Select n1.n_name as supp_nation, n2.n_name as cust_nation,
wl_shipdate as l_year, wl_extendedprice*(1 - wl_discount) as revenue
 From customer, nation n1, nation n2, orders, supplier, web_lineitem
 Where orders.o_orderkey = web_lineitem.wl_orderkey
 and supplier.s_suppkey = web_lineitem.wl_suppkey
 and customer.c_custkey = orders.o_custkey
 and customer.c_nationkey = n2.n_nationkey
 and n1.n_nationkey = supplier.s_nationkey
 and (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY') OR (n2.n_name = 'FRANCE' and n1.n_name = 'GERMANY')
 and web_lineitem.wl_shipdate between '1995-01-01' and '1996-12-31')
 UNION ALL
 (Select n1.n_name as supp_nation, n2.n_name as cust_nation,
 sl_shipdate as l_year, sl_extendedprice*(1 - sl_discount) as revenue
 From customer, nation n1, nation n2, orders, supplier, store_lineitem
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and supplier.s_suppkey = store_lineitem.sl_suppkey
 and customer.c_custkey = orders.o_custkey
 and customer.c_nationkey = n2.n_nationkey
 and n1.n_nationkey = supplier.s_nationkey
 and (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY') OR (n2.n_name = 'FRANCE' and n1.n_name = 'GERMANY')
 and store_lineitem.sl_shipdate between '1995-01-01' and '1996-12-31');
```

A.7.3 Business Description Text.

The Query finds, for two given nations, the gross discounted revenues derived from line items in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. The query lists the supplier nation, the customer nation, the year, and the revenue from shipments that took place in that year. The query orders the answer by Supplier nation, Customer nation, and year (all ascending).

A.7.4 Output of Xpose.

```
SELECT supp_nation, cust_nation, l_year, SUM(revenue) as revenue
FROM (
    SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
    EXTRACT(YEAR FROM wl_shipdate) as l_year, wl_extendedprice*(1 - wl_discount) as revenue
    FROM customer, nation n1, nation n2, orders, supplier, web_lineitem
    WHERE orders.o_orderkey = web_lineitem.wl_orderkey
    AND supplier.s_suppkey = web_lineitem.wl_suppkey
    AND customer.c_custkey = orders.o_custkey
    AND customer.c_nationkey = n2.n_nationkey
    AND n1.n_nationkey = supplier.s_nationkey
    AND ((n1.n_name = 'FRANCE' AND n2.n_name = 'GERMANY') OR (n2.n_name = 'FRANCE' AND n1.n_name = 'GERMANY'))
    AND web_lineitem.wl_shipdate BETWEEN '1995-01-01' AND '1996-12-31'
    UNION ALL
    SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
    EXTRACT(YEAR FROM sl_shipdate) as l_year, sl_extendedprice*(1 - sl_discount) as revenue
    FROM customer, nation n1, nation n2, orders, supplier, store_lineitem
    WHERE orders.o_orderkey = store_lineitem.sl_orderkey
    AND supplier.s_suppkey = store_lineitem.sl_suppkey
    AND customer.c_custkey = orders.o_custkey
    AND customer.c_nationkey = n2.n_nationkey
    AND n1.n_nationkey = supplier.s_nationkey
    AND ((n1.n_name = 'FRANCE' AND n2.n_name = 'GERMANY') OR (n2.n_name = 'FRANCE' AND n1.n_name = 'GERMANY'))
    AND store_lineitem.sl_shipdate BETWEEN '1995-01-01' AND '1996-12-31'
) AS combined
```

```
GROUP BY supp_nation, cust_nation, l_year
ORDER BY supp_nation, cust_nation, l_year;
```

A.8 E-TPCH Q8

A.8.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
	o_year,
	sum(case
		when nation = 'INDIA' then volume
		else 0
	end) / sum(volume) as mkt_share
from
	(
		select
			extract(year from o_orderdate) as o_year,
			wl_extendedprice * (1 - wl_discount) as volume,
			n2.n_name as nation
		from
			part,
			supplier,
			web_lineitem,
			orders,
			customer,
			nation n1,
			nation n2,
			region
		where
			p_partkey = wl_partkey
			and s_suppkey = wl_suppkey
			and wl_orderkey = o_orderkey
			and o_custkey = c_custkey
			and c_nationkey = n1.n_nationkey
			and n1.n_regionkey = r_regionkey
			and r_name = 'ASIA'
			and s_nationkey = n2.n_nationkey
			and o_orderdate between date '1995-01-01' and date '1996-12-31'
			and p_type = 'ECONOMY ANODIZED STEEL'
	) as all_nations
group by
	o_year
order by
	o_year;
```

A.8.2 Seed Query produced by XRE.

```
Select o_orderdate as o_year, Sum(0) as mkt_share
 From customer, web_lineitem, nation n1, nation n2, orders, part, region, supplier
 Where c_custkey = o_custkey
 and c_nationkey = n1.n_nationkey
 and wl_orderkey = o_orderkey
 and wl_partkey = p_partkey
 and wl_suppkey = s_suppkey
 and n1.n_regionkey = r_regionkey
 and n2.n_nationkey = s_nationkey
 and p_type = 'ECONOMY ANODIZED STEEL'
 and r_name = 'ASIA'
 and o_orderdate between '1995-01-01' and '1996-12-31'
group by o_orderdate;
```

A.8.3 **Business Description Text.** The market share for 'INDIA' within Asian region is defined as the fraction of the revenue, the sum of [extended price * (1-discount)], from the products of 'ECONOMY ANODIZED STEEL' type sold online in that region that was supplied by the Indian suppliers. The query determines this for the years 1995 and 1996 presented in this order.

A.8.4 **Output of XPOSE.**

```
SELECT
    EXTRACT(YEAR FROM o_orderdate) AS o_year,
    SUM(CASE WHEN n2.n_name = 'INDIA'
    THEN wl_extendedprice * (1 - wl_discount) ELSE 0 END) / SUM(wl_extendedprice * (1 - wl_discount)) AS mkt_share
FROM
    customer, web_lineitem, nation n1, nation n2, orders, part, region, supplier
WHERE
    c_custkey = o_custkey
    AND c_nationkey = n1.n_nationkey
    AND wl_orderkey = o_orderkey
    AND wl_partkey = p_partkey
    AND wl_suppkey = s_suppkey
    AND n1.n_regionkey = r_regionkey
    AND n2.n_nationkey = s_nationkey
    AND p_type = 'ECONOMY ANODIZED STEEL'
    AND r_name = 'ASIA'
    AND o_orderdate BETWEEN '1995-01-01' AND '1996-12-31'
GROUP BY
    EXTRACT(YEAR FROM o_orderdate)
ORDER BY
    o_year;
```

A.9 E-TPCH Q9

A.9.1 **Ground Truth Query $Q_{\mathcal{H}}$.**

```
select
        nation,
        o_year,
        sum(amount) as sum_profit
from
        (
                select
                        n_name as nation, p_name,
                        extract(year from o_orderdate) as o_year,
                        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
                from
                        part,
                        supplier,
                        (select
                        wl_extendedprice as l_extendedprice,
                        wl_discount as l_discount,
                        wl_quantity as l_quantity,
                        wl_suppkey as l_suppkey,
                        wl_partkey as l_partkey,
                        wl_orderkey as l_orderkey
                        from web_lineitem
                        UNION ALL
                        select
                        sl_extendedprice as l_extendedprice,
                        sl_discount as l_discount,
                        sl_quantity as l_quantity,
                        sl_suppkey as l_suppkey,
                        sl_partkey as l_partkey,
```

```
                        sl_orderkey as l_orderkey
                        from store_lineitem
                        ) as lineitem,
                        partsupp,
                        orders,
                        nation
                where
                        s_suppkey = l_suppkey
                        and ps_suppkey = l_suppkey
                        and ps_partkey = l_partkey
                        and p_partkey = l_partkey
                        and o_orderkey = l_orderkey
                        and s_nationkey = n_nationkey
                        and p_name like 'co%'
        ) as profit
group by
        nation,
        o_year
order by
        nation,
        o_year desc;
```

A.9.2 Seed Query produced by XRE.

```
(Select n_name as nation, o_orderdate as o_year,
Sum(-ps_supplycost*wl_quantity + wl_extendedprice*(1 - wl_discount)) as sum_profit
 From nation, orders, part, partsupp, supplier, web_lineitem
 Where orders.o_orderkey = web_lineitem.wl_orderkey
 and part.p_partkey = partsupp.ps_partkey
 and partsupp.ps_partkey = web_lineitem.wl_partkey
 and partsupp.ps_suppkey = supplier.s_suppkey
 and supplier.s_suppkey = web_lineitem.wl_suppkey
 and nation.n_nationkey = supplier.s_nationkey
 and part.p_name LIKE 'co%'
 Group By n_name , o_orderdate
 Order By nation asc)
 UNION ALL
 (Select n_name as nation, o_orderdate as o_year,
 Sum(-ps_supplycost*sl_quantity + sl_extendedprice*(1 - sl_discount)) as sum_profit
 From nation, orders, part, partsupp, store_lineitem, supplier
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and part.p_partkey = partsupp.ps_partkey
 and partsupp.ps_partkey = store_lineitem.sl_partkey
 and partsupp.ps_suppkey = store_lineitem.sl_suppkey
 and store_lineitem.sl_suppkey = supplier.s_suppkey
 and nation.n_nationkey = supplier.s_nationkey
 and part.p_name LIKE 'co%'
 Group By n_name , o_orderdate
 Order By nation asc);
```

A.9.3 **Business Description Text.** The Query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation. The profit is defined as the sum of [(extended price*(1-discount)) - (supply cost * quantity)] for all line items describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first).

A.9.4 Output of Xpose.

```
SELECT
    nation,
    EXTRACT(YEAR FROM o_year) AS o_year,
```

```
    SUM(sum_profit) AS sum_profit
FROM (
    SELECT
        n_name AS nation,
        o_orderdate AS o_year,
        SUM(-ps_supplycost * wl_quantity + wl_extendedprice * (1 - wl_discount)) AS sum_profit
    FROM
        nation, orders, part, partsupp, supplier, web_lineitem
    WHERE
        orders.o_orderkey = web_lineitem.wl_orderkey
        AND part.p_partkey = partsupp.ps_partkey
        AND partsupp.ps_partkey = web_lineitem.wl_partkey
        AND partsupp.ps_suppkey = supplier.s_suppkey
        AND supplier.s_suppkey = web_lineitem.wl_suppkey
        AND nation.n_nationkey = supplier.s_nationkey
        AND part.p_name LIKE '%co%'
    GROUP BY
        n_name, o_orderdate

    UNION ALL

    SELECT
        n_name AS nation,
        o_orderdate AS o_year,
        SUM(-ps_supplycost * sl_quantity + sl_extendedprice * (1 - sl_discount)) AS sum_profit
    FROM
        nation, orders, part, partsupp, store_lineitem, supplier
    WHERE
        orders.o_orderkey = store_lineitem.sl_orderkey
        AND part.p_partkey = partsupp.ps_partkey
        AND partsupp.ps_partkey = store_lineitem.sl_partkey
        AND partsupp.ps_suppkey = store_lineitem.sl_suppkey
        AND store_lineitem.sl_suppkey = supplier.s_suppkey
        AND nation.n_nationkey = supplier.s_nationkey
        AND part.p_name LIKE '%co%'
    GROUP BY
        n_name, o_orderdate
) AS combined
GROUP BY
    nation, o_year
ORDER BY
    nation ASC, o_year DESC;
```

A.10 E-TPCH Q10

A.10.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        c_custkey,
        c_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,
        n_name,
        c_address,
        c_phone,
        c_comment
from
        customer,
```

```
        orders,
        (select
                        wl_extendedprice as l_extendedprice,
                        wl_discount as l_discount,
                        wl_returnflag as l_returnflag,
                        wl_orderkey as l_orderkey
                        from web_lineitem
                        UNION ALL
                        select
                        sl_extendedprice as l_extendedprice,
                        sl_discount as l_discount,
                        sl_returnflag as l_returnflag,
                        sl_orderkey as l_orderkey
                        from store_lineitem
                        ) as lineitem,
        nation
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate >= date '1995-01-01'
        and o_orderdate < date '1995-01-01' + interval '3' month
        and l_returnflag = 'R'
        and c_nationkey = n_nationkey
group by
        c_custkey,
        c_name,
        c_acctbal,
        c_phone,
        n_name,
        c_address,
        c_comment
order by
        revenue desc Limit 20;
```

A.10.2 Seed Query produced by XRE.

```
(Select c_custkey, c_name,
Sum(sl_extendedprice*(1 - sl_discount)) as revenue, c_acctbal, n_name, c_address, c_phone, c_comment
 From customer, nation, orders, store_lineitem
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and customer.c_nationkey = nation.n_nationkey
 and customer.c_custkey = orders.o_custkey
 and store_lineitem.sl_returnflag = 'R'
 and orders.o_orderdate between '1995-01-01' and '1995-03-31'
 Group By c_acctbal, c_address, c_comment, c_custkey, c_name, c_phone, n_name
 Order By revenue desc, c_custkey asc, c_name asc,
 c_acctbal asc, c_phone asc, n_name asc, c_address asc, c_comment asc Limit 20)
 UNION ALL
 (Select c_custkey, c_name,
 Sum(wl_extendedprice*(1 - wl_discount)) as revenue, c_acctbal, n_name, c_address, c_phone, c_comment
 From customer, nation, orders, web_lineitem
 Where customer.c_nationkey = nation.n_nationkey
 and orders.o_orderkey = web_lineitem.wl_orderkey
 and customer.c_custkey = orders.o_custkey
 and web_lineitem.wl_returnflag = 'R'
 and orders.o_orderdate between '1995-01-01' and '1995-03-31'
 Group By c_acctbal, c_address, c_comment, c_custkey, c_name, c_phone, n_name
 Order By revenue desc, c_custkey asc, c_name asc,
```

```
c_acctbal asc, c_phone asc, n_name asc, c_address asc, c_comment asc Limit 20);
```

A.10.3 **Business Description Text.** The Query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue. Revenue lost is defined as sum(extendedprice*(1-discount)) for all qualifying line items.

A.10.4 **Output of XPOSE.**

```
SELECT c_custkey, c_name, SUM(revenue) AS revenue,
c_acctbal, n_name, c_address, c_phone, c_comment
FROM (
    SELECT c_custkey, c_name, sl_extendedprice * (1 - sl_discount) AS revenue,
    c_acctbal, n_name, c_address, c_phone, c_comment
    FROM customer, nation, orders, store_lineitem
    WHERE orders.o_orderkey = store_lineitem.sl_orderkey
    AND customer.c_nationkey = nation.n_nationkey
    AND customer.c_custkey = orders.o_custkey
    AND store_lineitem.sl_returnflag = 'R'
    AND orders.o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'
    UNION ALL
    SELECT c_custkey, c_name, wl_extendedprice * (1 - wl_discount) AS revenue,
    c_acctbal, n_name, c_address, c_phone, c_comment
    FROM customer, nation, orders, web_lineitem
    WHERE customer.c_nationkey = nation.n_nationkey
    AND orders.o_orderkey = web_lineitem.wl_orderkey
    AND customer.c_custkey = orders.o_custkey
    AND web_lineitem.wl_returnflag = 'R'
    AND orders.o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'
) AS combined
GROUP BY c_custkey, c_name, c_acctbal, n_name, c_address, c_phone, c_comment
ORDER BY revenue DESC, c_custkey ASC, c_name ASC, c_acctbal ASC, c_phone ASC, n_name ASC, c_address ASC, c_comment ASC
LIMIT 20;
```

A.11 E-TPCH Q11

A.11.1 **Ground Truth Query $Q_{\mathcal{H}}$.**

```
SELECT
    ps_partkey, n_name,
    SUM(ps_supplycost * ps_availqty) AS total_value
FROM
    partsupp, supplier, nation
where
    ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'INDIA'
GROUP BY
    ps_partkey, n_name
HAVING
    SUM(ps_supplycost * ps_availqty) > (
        SELECT SUM(ps_supplycost * ps_availqty) * 0.00001
        FROM partsupp, supplier, nation WHERE
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'INDIA'
    )
ORDER BY
    total_value DESC;
```

A.11.2 **Seed Query produced by XRE.**

```
SELECT ps_partkey, n_name, MIN(partsupp.ps_availqty), MIN(partsupp.ps_supplycost)
       FROM nation, supplier, partsupp
       WHERE supplier.s_nationkey = nation.n_nationkey
       AND supplier.s_suppkey = partsupp.ps_suppkey
       AND nation.n_name = 'INDIA'
       GROUP BY partsupp.ps_partkey, nation.n_name
       HAVING MIN(partsupp.ps_availqty) >= 1 AND MIN(partsupp.ps_supplycost) >= 0.01;
```

A.11.3 **Business Description Text.** The Query finds, from scanning the available stock of suppliers in India, all the parts that represent 0.001% of the total value of all available parts. The query displays the part number and the value of those parts in descending order of value.

A.11.4 **Output of** XPOSE**.**

```
SELECT ps_partkey, n_name, part_value
FROM (
    SELECT partsupp.ps_partkey, nation.n_name,
           SUM(partsupp.ps_availqty * partsupp.ps_supplycost) AS part_value
    FROM nation, supplier, partsupp
    WHERE supplier.s_nationkey = nation.n_nationkey
      AND supplier.s_suppkey = partsupp.ps_suppkey
      AND nation.n_name = 'INDIA'
    GROUP BY partsupp.ps_partkey, nation.n_name
) AS part_values
WHERE part_value >= 0.00001 * (SELECT SUM(ps_availqty * ps_supplycost)
                               FROM nation, supplier, partsupp
                               WHERE supplier.s_nationkey = nation.n_nationkey
                                 AND supplier.s_suppkey = partsupp.ps_suppkey
                                 AND nation.n_name = 'INDIA')
ORDER BY part_value DESC;
```

A.12 E-TPCH Q12

A.12.1 **Ground Truth Query** $Q_{\mathcal{H}}$.

```
select
        l_shipmode,
        sum(case
                when o_orderpriority = '1-URGENT'
                        or o_orderpriority = '2-HIGH'
                        then 1
                else 0
        end) as high_line_count,
        sum(case
                when o_orderpriority <> '1-URGENT'
                        and o_orderpriority <> '2-HIGH'
                        then 1
                else 0
        end) as low_line_count
from
        orders,
        (select
                sl_shipmode as l_shipmode,
                sl_orderkey as l_orderkey,
                sl_commitdate as l_commitdate,
                sl_shipdate as l_shipdate,
                sl_receiptdate as l_receiptdate
                from store_lineitem
                UNION ALL
                select
                wl_shipmode as l_shipmode,
```

```
                wl_orderkey as l_orderkey,
                wl_commitdate as l_commitdate,
                wl_shipdate as l_shipdate,
                wl_receiptdate as l_receiptdate
                from web_lineitem) as lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode IN ('SHIP','TRUCK')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= date '1995-01-01'
        and l_receiptdate < date '1995-01-01' + interval '1' year
group by
        l_shipmode
order by
        l_shipmode;
```

A.12.2 **Seed Query produced by XRE.**

```
(Select sl_shipmode, 0 as high_line_count, Count(*) as low_line_count
 From orders, store_lineitem
 Where orders.o_orderkey = store_lineitem.sl_orderkey
 and store_lineitem.sl_shipdate < store_lineitem.sl_commitdate
 and store_lineitem.sl_commitdate < store_lineitem.sl_receiptdate
 and store_lineitem.sl_shipmode IN ('SHIP', 'TRUCK')
 and store_lineitem.sl_receiptdate between '1995-01-01' and '1995-12-31'
group by sl_shipmode)
 UNION ALL
 (Select wl_shipmode, 0 as high_line_count, Count(*) as low_line_count
 From orders, web_lineitem
 Where orders.o_orderkey = web_lineitem.wl_orderkey
 and web_lineitem.wl_shipdate < web_lineitem.wl_commitdate
 and web_lineitem.wl_commitdate < web_lineitem.wl_receiptdate
 and web_lineitem.wl_shipmode IN ('SHIP', 'TRUCK')
 and web_lineitem.wl_receiptdate between '1995-01-01' and '1995-12-31'
 group by wl_shipmode);
```

A.12.3 **Business Description Text.** The Query counts, by ship mode, for line items actually received by customers in the year 1995, the number of line items belonging to orders for which the receiptdate exceeds the commitdate for two different specified ship modes. Only line items that were actually shipped before the commitdate are considered. The late line items are partitioned into two groups, those with priority URGENT or HIGH, and those with a priority other than URGENT or HIGH.

A.12.4 **Output of Xpose.**

```
SELECT shipmode,
       SUM(high_line_count) AS high_line_count,
       SUM(low_line_count) AS low_line_count
FROM (
    SELECT sl_shipmode AS shipmode,
           SUM(CASE WHEN o_orderpriority IN ('1-URGENT', '2-HIGH') THEN 1 ELSE 0 END) AS high_line_count,
           SUM(CASE WHEN o_orderpriority NOT IN ('1-URGENT', '2-HIGH') THEN 1 ELSE 0 END) AS low_line_count
    FROM orders, store_lineitem
    WHERE orders.o_orderkey = store_lineitem.sl_orderkey
      AND store_lineitem.sl_shipdate < store_lineitem.sl_commitdate
      AND store_lineitem.sl_commitdate < store_lineitem.sl_receiptdate
      AND store_lineitem.sl_shipmode IN ('SHIP', 'TRUCK')
      AND store_lineitem.sl_receiptdate BETWEEN '1995-01-01' AND '1995-12-31'
    GROUP BY sl_shipmode
    UNION ALL
    SELECT wl_shipmode AS shipmode,
```

```
            SUM(CASE WHEN o_orderpriority IN ('1-URGENT', '2-HIGH') THEN 1 ELSE 0 END) AS high_line_count,
            SUM(CASE WHEN o_orderpriority NOT IN ('1-URGENT', '2-HIGH') THEN 1 ELSE 0 END) AS low_line_count
    FROM orders, web_lineitem
    WHERE orders.o_orderkey = web_lineitem.wl_orderkey
      AND web_lineitem.wl_shipdate < web_lineitem.wl_commitdate
      AND web_lineitem.wl_commitdate < web_lineitem.wl_receiptdate
      AND web_lineitem.wl_shipmode IN ('SHIP', 'TRUCK')
      AND web_lineitem.wl_receiptdate BETWEEN '1995-01-01' AND '1995-12-31'
    GROUP BY wl_shipmode
) AS combined
GROUP BY shipmode;
```

A.13 E-TPCH Q13

A.13.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        c_count, c_orderdate,
        count(*) as custdist
from
        (
                select
                        c_custkey, o_orderdate,
                        count(o_orderkey)
                from
                        customer left outer join orders on
                                c_custkey = o_custkey
                                and o_comment not like '%special%requests%'
                group by
                        c_custkey, o_orderdate
        ) as c_orders (c_custkey, c_count, c_orderdate)
group by
        c_count, c_orderdate
order by
        custdist desc,
        c_count desc;
```

A.13.2 Seed Query produced by XRE.

```
Select o_orderdate as c_count, Count(*) as c_orderdate, <unknown> as custdist
 From customer, orders
 Where customer.c_custkey = orders.o_custkey
 Group By o_orderdate, c_custkey
 Order By c_count DESC;
```

A.13.3 **Business Description Text.** This query determines the distribution of customers by the number of orders they have made, including customers who have no record of orders, in past or present. It counts and reports how many customers have no orders, how many have 1, 2, 3, etc. A check is made to ensure that the orders counted do not fall into one of several special categories of orders. Special categories are identified in the order comment column by looking for the pattern '%special%requests%'.

A.13.4 Output of Xpose.

```
SELECT
    c_count,
    custdist AS c_orderdate,
    COUNT(*) as custdist
FROM (
    SELECT
        c.c_custkey as c_custkey,
        o.o_orderdate AS c_count,
 COUNT(o.o_orderkey) AS custdist
```

```
    FROM customer c
    LEFT JOIN orders o
    ON c.c_custkey = o.o_custkey
        AND o.o_comment NOT LIKE '%special%requests%'
    GROUP BY c.c_custkey, o.o_orderdate
) AS subquery
GROUP BY subquery.c_count, subquery.custdist
ORDER BY custdist desc, c_count DESC;
```

A.14 E-TPCH Q14

A.14.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
 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
        (select
                 sl_extendedprice as l_extendedprice,
                 sl_discount as l_discount,
                 sl_partkey as l_partkey,
                 sl_shipdate as l_shipdate
                 from store_lineitem
                 UNION ALL
                 select wl_extendedprice as l_extendedprice,
                 wl_discount as l_discount,
                 wl_partkey as l_partkey,
                 wl_shipdate as l_shipdate
                 from web_lineitem) as lineitem,
                 part
                 where
                 l_partkey = p_partkey
        and l_shipdate >= date '1995-01-01'
        and l_shipdate < date '1995-01-01' + interval '1' month;
```

A.14.2 Seed Query produced by XRE.

```
(Select Sum(0) as promo_revenue
 From part, store_lineitem
 Where part.p_partkey = store_lineitem.sl_partkey
 and store_lineitem.sl_shipdate between '1995-01-01' and '1995-01-31')
 UNION ALL
 (Select Sum(0) as promo_revenue
 From part, web_lineitem
 Where part.p_partkey = web_lineitem.wl_partkey
 and web_lineitem.wl_shipdate between '1995-01-01' and '1995-01-31');
```

A.14.3 **Business Description Text.** The Query determines what percentage of the revenue in a given year and month was derived from promotional parts. The query considers only parts actually shipped in that month and gives the percentage. Revenue is defined as (extended price * (1-discount)).

A.14.4 Output of Xpose.

```
SELECT SUM(promo_revenue) / SUM(total_revenue) * 100 AS promo_revenue_percentage
FROM (
    SELECT
     SUM(CASE WHEN part.p_type LIKE 'PROMO%' THEN store_lineitem.sl_extendedprice * (1 - store_lineitem.sl_discount)
        ELSE 0 END) AS promo_revenue,
```

```
        SUM(store_lineitem.sl_extendedprice * (1 - store_lineitem.sl_discount)) AS total_revenue
    FROM
        part, store_lineitem
    WHERE
        part.p_partkey = store_lineitem.sl_partkey
        AND store_lineitem.sl_shipdate BETWEEN '1995-01-01' AND '1995-01-31'
    UNION ALL
    SELECT
       SUM(CASE WHEN part.p_type LIKE 'PROMO%' THEN web_lineitem.wl_extendedprice * (1 - web_lineitem.wl_discount)
        ELSE 0 END) AS promo_revenue,
        SUM(web_lineitem.wl_extendedprice * (1 - web_lineitem.wl_discount)) AS total_revenue
    FROM
        part, web_lineitem
    WHERE
        part.p_partkey = web_lineitem.wl_partkey
        AND web_lineitem.wl_shipdate BETWEEN '1995-01-01' AND '1995-01-31'
) AS combined_revenue;
```

A.15 E-TPCH Q15

A.15.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
with revenue(supplier_no, total_revenue) as
(select
                l_suppkey,
                sum(l_extendedprice * (1 - l_discount))
        from
                (select
                 sl_extendedprice as l_extendedprice,
                 sl_discount as l_discount,
                 sl_suppkey as l_suppkey,
                 sl_shipdate as l_shipdate
                 from store_lineitem
                 UNION ALL
                 select
                 wl_extendedprice as l_extendedprice,
                 wl_discount as l_discount,
                 wl_suppkey as l_suppkey,
                 wl_shipdate as l_shipdate
                 from web_lineitem
        ) as lineitem
        where
                l_shipdate >= date '1995-01-01'
                and l_shipdate < date '1995-01-01' + interval '3' month
        group by
                l_suppkey)
select
        s_suppkey,
        s_name,
        s_address,
        s_phone,
        total_revenue
from
        supplier,
        revenue
where
        s_suppkey = supplier_no
        and total_revenue = (
```

```
                select
                        max(total_revenue)
                from
                        revenue
        )
order by
        s_suppkey;
```

A.15.2 **Seed Query produced by XRE.**

```
(Select Max(wl_suppkey) as s_suppkey, s_name, s_address, s_phone,
Sum(wl_extendedprice*(1 - wl_discount)) as total_revenue
 From supplier, web_lineitem
 Where supplier.s_suppkey = web_lineitem.wl_suppkey
 and web_lineitem.wl_shipdate between '1995-01-01' and '1995-01-31'
 group by s_name, s_address, s_phone)
 UNION ALL
 (Select Max(sl_suppkey) as s_suppkey, s_name, s_address, s_phone,
 Sum(sl_extendedprice*(1 - sl_discount)) as total_revenue
 From store_lineitem, supplier
 Where store_lineitem.sl_suppkey = supplier.s_suppkey
 and store_lineitem.sl_shipdate between '1995-01-01' and '1995-01-31'
 group by s_name, s_address, s_phone);
```

A.15.3 **Business Description Text.** The Query finds the supplier who contributed the most to the overall revenue for parts shipped during the first quarter of 1995. In case of a tie, the query lists all suppliers whose contribution was equal to the maximum, presented in supplier number order. Revenue is calculated as sum(extended price*(1-discount)).

A.15.4 **Output of XPOSE.**

```
WITH combined_revenue AS (
    SELECT wl_suppkey AS s_suppkey,
           s_name,
           s_address,
           s_phone,
           SUM(wl_extendedprice * (1 - wl_discount)) AS total_revenue
    FROM web_lineitem
    JOIN supplier ON supplier.s_suppkey = web_lineitem.wl_suppkey
    WHERE wl_shipdate BETWEEN '1995-01-01' AND '1995-03-31'
    GROUP BY wl_suppkey, s_name, s_address, s_phone

    UNION ALL

    SELECT sl_suppkey AS s_suppkey,
           s_name,
           s_address,
           s_phone,
           SUM(sl_extendedprice * (1 - sl_discount)) AS total_revenue
    FROM store_lineitem
    JOIN supplier ON supplier.s_suppkey = store_lineitem.sl_suppkey
    WHERE sl_shipdate BETWEEN '1995-01-01' AND '1995-03-31'
    GROUP BY sl_suppkey, s_name, s_address, s_phone
)

SELECT DISTINCT s_suppkey, s_name, s_address, s_phone, total_revenue
FROM combined_revenue
WHERE total_revenue = (
    SELECT MAX(total_revenue)
    FROM combined_revenue
)
```

```
ORDER BY s_suppkey;
```

A.16 E-TPCH Q16

A.16.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        p_brand,
        p_type,
        p_size,
        count(distinct ps_suppkey) as supplier_cnt
from
        partsupp,
        part
where
        p_partkey = ps_partkey
        and p_brand <> 'Brand#23'
    AND p_type NOT LIKE 'MEDIUM POLISHED%'
        and p_size IN (1, 4, 7)
        and ps_suppkey not in (
                select
                        s_suppkey
                from
                        supplier
                where
                        s_comment like '%Customer%Complaints%'
        )
group by
        p_brand,
        p_type,
        p_size
order by
        supplier_cnt desc,
        p_brand,
        p_type,
        p_size;
```

A.16.2 Seed Query produced by XRE.

```
Select p_brand, p_type, p_size, Count(*) as supplier_cnt
 From part, partsupp, supplier
 Where part.p_partkey = partsupp.ps_partkey
 and part.p_size IN (1, 4, 7)
 Group By p_brand, p_size, p_type;
```

A.16.3 **Business Description Text.** The Query counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. The customer is interested in parts of sizes 1, 4, and 7 as long as they are not like 'MEDIUM POLISHED%', not of 'Brand#23', and not from a supplier who has had complaints registered at the Better Business Bureau, identified by pattern '%Customer%Complaints%' in the comments..

A.16.4 Output of XPOSE.

```
SELECT p_brand, p_type, p_size, COUNT(*) AS supplier_cnt
FROM part
JOIN partsupp ON part.p_partkey = partsupp.ps_partkey
JOIN supplier ON partsupp.ps_suppkey = supplier.s_suppkey
WHERE part.p_size IN (1, 4, 7)
AND part.p_brand <> 'Brand#23'
AND part.p_type NOT LIKE 'MEDIUM POLISHED%'
AND supplier.s_comment NOT LIKE '%Customer%Complaints%'
GROUP BY p_brand, p_size, p_type
```

```
ORDER BY supplier_cnt DESC, p_brand ASC, p_type ASC, p_size ASC;
```

A.17 E-TPCH Q17

A.17.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select sum(wl_extendedprice) / 7.0 as avg_yearly
from
        web_lineitem,
        part
where
        p_partkey = wl_partkey
        and p_brand = 'Brand#53'
        and p_container = 'MED BAG'
        and wl_quantity < (
                select
                        0.7 * avg(wl_quantity)
                from
                        web_lineitem
                where
                        wl_partkey = p_partkey
        );
```

A.17.2 Seed Query produced by XRE.

```
Select 0.14*wl_extendedprice as avg_yearly
From part, web_lineitem w1, web_lineitem w1
Where part.p_partkey = w1.wl_partkey
 and w1.wl_partkey = w2.wl1_partkey
 and w1.wl_quantity < w2.wl1_quantity
 and part.p_brand = 'Brand#53'
 and part.p_container = 'MED BAG'
 and w1.wl_quantity <= 1503238553.51
```

A.17.3 **Business Description Text.** The Query considers parts of a given brand and with a given container type and determines the average lineitem quantity of such parts ordered for all orders (past and pending) in the 7-year database. What would be the average yearly gross (undiscounted) loss in revenue if orders for these parts with a quantity of less than 70% of this average were no longer taken?

A.17.4 Output of Xpose.

```
SELECT 0.14 * SUM(w1.wl_extendedprice) AS avg_yearly
FROM part
JOIN web_lineitem w1 ON part.p_partkey = w1.wl_partkey
JOIN (
    SELECT wl_partkey, AVG(wl_quantity) * 0.7 AS threshold_quantity
    FROM web_lineitem
    GROUP BY wl_partkey
) w2 ON w1.wl_partkey = w2.wl_partkey
WHERE part.p_brand = 'Brand#53'
  AND part.p_container = 'MED BAG'
  AND w1.wl_quantity < w2.threshold_quantity;
```

A.18 E-TPCH Q18

A.18.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
 select
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice,
        sum(wl_quantity)
```

```
from
        customer,
        orders,
        web_lineitem
where
        o_orderkey in (
                select
                        wl_orderkey
                from
                        web_lineitem
                group by
                        wl_orderkey having
                                sum(wl_quantity) > 300
        )
        and c_custkey = o_custkey
        and o_orderkey = wl_orderkey
group by
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice
order by
        o_totalprice desc,
        o_orderdate;
```

A.18.2 Seed Query produced by XRE.

```
SELECT <unknown>
        FROM customer, orders, web_lineitem
        WHERE orders.o_orderkey = web_lineitem.wl_orderkey AND customer.c_custkey = orders.o_custkey
        GROUP BY customer.c_custkey, customer.c_name, web_lineitem.wl_orderkey
        HAVING SUM(web_lineitem.wl_quantity) >= 300.01;
```

A.18.3 **Business Description Text.** The Query finds a list of the top 100 customers who have ever placed more than 300 orders online. The query lists the customer name, customer key, the order key, date and total price and the quantity for the order.

A.18.4 Output of Xpose.

```
SELECT c.c_name, c.c_custkey, o.o_orderkey, o.o_orderdate, o.o_totalprice, SUM(wl.wl_quantity) AS total_quantity
FROM customer c
JOIN orders o ON c.c_custkey = o.o_custkey
JOIN web_lineitem wl ON o.o_orderkey = wl.wl_orderkey
WHERE o.o_orderkey IN (
    SELECT wl_orderkey
    FROM web_lineitem
    GROUP BY wl_orderkey
    HAVING SUM(wl_quantity) > 300
)
GROUP BY c.c_name, c.c_custkey, o.o_orderkey, o.o_orderdate, o.o_totalprice
ORDER BY total_quantity DESC
LIMIT 100;
```

A.19 E-TPCH Q19

A.19.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
 select
        sum(wl_extendedprice* (1 - wl_discount)) as revenue
from
```

```
        web_lineitem,
        part
where
        (
                p_partkey = wl_partkey
                and p_brand = 'Brand#12'
                and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                and wl_quantity >= 1 and wl_quantity <= 1 + 10
                and p_size between 1 and 5
                and wl_shipmode in ('AIR', 'AIR REG')
                and wl_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = wl_partkey
                and p_brand = 'Brand#23'
                and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
                and wl_quantity >= 10 and wl_quantity <= 10 + 10
                and p_size between 1 and 10
                and wl_shipmode in ('AIR', 'AIR REG')
                and wl_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = wl_partkey
                and p_brand = 'Brand#34'
                and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
                and wl_quantity >= 20 and wl_quantity <= 20 + 10
                and p_size between 1 and 15
                and wl_shipmode in ('AIR', 'AIR REG')
                and wl_shipinstruct = 'DELIVER IN PERSON'
        );
```

A.19.2 **Seed Query produced by XRE.**

```
select
        sum(wl_extendedprice* (1 - wl_discount)) as revenue
from
        web_lineitem,
        part
where (p_partkey = wl_partkey
                and p_brand = 'Brand#12'
                and p_size between 1 and 5
                and l_shipinstruct = 'DELIVER IN PERSON'
                and l_shipmode = 'AIR'
                and l_quantity between 1 and 11
                and p_container = 'SM CASE')
    OR (p_partkey = wl_partkey
                and p_brand = 'Brand#12'
                and p_size between 1 and 5
                and l_shipinstruct = 'DELIVER IN PERSON'
                and l_shipmode = 'AIR'
                and l_quantity between 1 and 11
                and p_container = 'SM BOX')
    OR (p_partkey = wl_partkey
                and p_brand = 'Brand#12'
                and p_size between 1 and 5
                and l_shipinstruct = 'DELIVER IN PERSON'
```

```
            and l_shipmode = 'AIR'
            and l_quantity between 1 and 11
            and p_container = 'SM PACK')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#12'
            and p_size between 1 and 5
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 1 and 11
            and p_container = 'SM PKG')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#23'
            and p_size between 1 and 10
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 10 and 20
            and p_container = 'MED CASE')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#23'
            and p_size between 1 and 10
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 10 and 20
            and p_container = 'MED BOX')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#23'
            and p_size between 1 and 10
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 10 and 20
            and p_container = 'MED PACK')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#23'
            and p_size between 1 and 10
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 10 and 20
            and p_container = 'MED PKG')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#34'
            and p_size between 1 and 15
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 20 and 30
            and p_container = 'LG CASE')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#34'
            and p_size between 1 and 15
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
            and l_quantity between 20 and 30
            and p_container = 'LG BOX')
OR (p_partkey = wl_partkey
            and p_brand = 'Brand#34'
            and p_size between 1 and 15
            and l_shipinstruct = 'DELIVER IN PERSON'
            and l_shipmode = 'AIR'
```

```
                and l_quantity between 20 and 30
                and p_container = 'LG PACK')
    OR (p_partkey = wl_partkey
                and p_brand = 'Brand#34'
                and p_size between 1 and 15
                and l_shipinstruct = 'DELIVER IN PERSON'
                and l_shipmode = 'AIR'
                and l_quantity between 20 and 30
                and p_container = 'LG PKG');
```

A.19.3 **Business Description Text.** The query finds the gross discounted revenue for all orders for three different types of parts that were shipped by air and delivered in person. Parts are selected based on the combination of specific brands, a list of containers, and a range of sizes.

A.19.4 **Output of XPOSE.**

```
select
    sum(wl_extendedprice * (1 - wl_discount)) as revenue
from
    web_lineitem,
    part
where
    p_partkey = wl_partkey
    and wl_shipinstruct = 'DELIVER IN PERSON'
    and wl_shipmode = 'AIR'
    and (
        (p_brand = 'Brand#12' and p_size between 1 and 5 and wl_quantity between 1 and 11
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG'))
        or (p_brand = 'Brand#23' and p_size between 1 and 10 and wl_quantity between 10 and 20
        and p_container in ('MED CASE', 'MED BOX', 'MED PACK', 'MED PKG'))
        or (p_brand = 'Brand#34' and p_size between 1 and 15 and wl_quantity between 20 and 30
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG'))
    );
```

A.20 E-TPCH Q20

A.20.1 **Ground Truth Query** $Q_{\mathcal{H}}$**.**

```
 select
        s_name,
        s_address
from
        supplier,
        nation
where
        s_suppkey in (
                select
                        ps_suppkey
                from
                        partsupp
                where
                        ps_partkey in (
                                select
                                        p_partkey
                                from
                                        part
                                where
                                        p_name like '%ivory%'
                        )
```

```
                        and ps_availqty > (
                                select
                                        0.5 * sum(wl_quantity)
                                from
                                        web_lineitem
                                where
                                        wl_partkey = ps_partkey
                                        and wl_suppkey = ps_suppkey
                                        and wl_shipdate >= date '1995-01-01'
                                        and wl_shipdate < date '1995-01-01' + interval '1' year
                        )
        )
        and s_nationkey = n_nationkey
        and n_name = 'FRANCE'
order by
        s_name
```

A.20.2 Seed Query produced by XRE.

```
Select s_name, s_address
 From web_lineitem, nation, part, partsupp, supplier
 Where wl_partkey = part.p_partkey
 and part.p_partkey = partsupp.ps_partkey
 and wl_suppkey = partsupp.ps_suppkey
 and partsupp.ps_suppkey = supplier.s_suppkey
 and nation.n_nationkey = supplier.s_nationkey
 and nation.n_name = 'FRANCE'
 and wl_quantity <= 9687.99
 and wl_shipdate between '1995-01-01' and '1995-12-31'
 and part.p_name LIKE '%ivory%'
 and partsupp.ps_availqty >= 12
 Order By s_name asc;
```

Considering the implementation of generalized algebraic predicates, when this module is included in XPOSE, we get the following seed $Q_{\mathcal{E}}$ (instead of the seed listed above:

```
Select s_name, s_address
from supplier, nation, partsupp, web_lineitem
where s_suppkey = ps_suppkey
    and ps_partkey = p_partkey
    and p_name like '%ivory%'
    and 2*ps_availqty - 0.01 >= wl_quantity
    and wl_partkey = ps_partkey
    and wl_suppkey = ps_suppkey
    and wl_shipdate >= date '1995-01-01'
    and wl_shipdate <= date '1995-12-31'
    and s_nationkey = n_nationkey
    and n_name = 'FRANCE'
order by s_name;
```

A.20.3 **Business Description Text.** The query identifies suppliers who have an excess of a given part available; an excess is defined to be more than 50% of the parts like the given part that the supplier shipped in 1995 for France. Only parts made of Ivory available online are considered.

A.20.4 Output of XPOSE.

```
SELECT s_name, s_address
FROM supplier
WHERE s_suppkey IN (
    SELECT ps_suppkey
    FROM partsupp
    WHERE ps_availqty > 0.5 * (
        SELECT SUM(l_quantity)
```

```
        FROM web_lineitem
        WHERE wl_shipdate BETWEEN '1995-01-01' AND '1995-12-31'
        AND wl_partkey = partsupp.ps_partkey
        AND wl_suppkey = partsupp.ps_suppkey
    )
    AND ps_partkey IN (
        SELECT p_partkey
        FROM part
        WHERE p_name LIKE '%ivory%'
    )
)
AND s_nationkey = (
    SELECT n_nationkey
    FROM nation
    WHERE n_name = 'FRANCE'
)
ORDER BY s_name ASC;
```

A.21 E-TPCH Q21

A.21.1 **Ground Truth Query $Q_{\mathcal{H}}$.**

```
 select
        s_name,
        count(*) as numwait
from
        supplier,
        web_lineitem l1,
        orders,
        nation
where
        s_suppkey = l1.wl_suppkey
        and o_orderkey = l1.wl_orderkey
        and o_orderstatus = 'F'
        and l1.wl_receiptdate > l1.wl_commitdate
        and exists (
                select
                        *
                from
                        web_lineitem l2
                where
                        l2.wl_orderkey = l1.wl_orderkey
                        and l2.wl_suppkey <> l1.wl_suppkey
        )
        and not exists (
                select
                        *
                from
                        web_lineitem l3
                where
                        l3.wl_orderkey = l1.wl_orderkey
                        and l3.wl_suppkey <> l1.wl_suppkey
                        and l3.wl_receiptdate > l3.wl_commitdate
        )
        and s_nationkey = n_nationkey
        and n_name = 'ARGENTINA'
group by
        s_name
```

```
order by
        numwait desc,
        s_name;
```

A.21.2 Seed Query produced by XRE.

```
Select s_name, <unknown> as numwait
 From web_lineitem l1, web_lineitem l2, nation, orders, supplier
 Where l1.wl_orderkey = l2.wl_orderkey
 and l2.wl_orderkey = o_orderkey
 and l1.wl_suppkey = s_suppkey
 and n_nationkey = s_nationkey
 and l1.wl_commitdate < l1.wl_receiptdate
 and n_name = 'ARGENTINA'
 and o_orderstatus = 'F'
 Order By s_name asc;
```

A.21.3 **Business Description Text.** The query identifies suppliers, for nation 'ARGENTINA', whose product was part of a multi-supplier online order (with current status of 'F') where they were the only supplier who failed to meet the committed delivery date.

A.21.4 Output of Xpose.

```
SELECT s_name, COUNT(*) AS numwait
FROM supplier, nation, orders, web_lineitem l1
WHERE s_suppkey = l1.wl_suppkey
  AND s_nationkey = n_nationkey
  AND n_name = 'ARGENTINA'
  AND l1.wl_orderkey = o_orderkey
  AND o_orderstatus = 'F'
  AND l1.wl_commitdate < l1.wl_receiptdate
  AND EXISTS (
    SELECT 1
    FROM web_lineitem l2
    WHERE l1.wl_orderkey = l2.wl_orderkey
      AND l1.wl_suppkey <> l2.wl_suppkey
  )
  AND NOT EXISTS (
    SELECT 1
    FROM web_lineitem l3
    WHERE l1.wl_orderkey = l3.wl_orderkey
      AND l1.wl_suppkey <> l3.wl_suppkey
      AND l3.wl_commitdate < l3.wl_receiptdate
  )
GROUP BY s_name
ORDER BY numwait DESC, s_name;
```

A.22 E-TPCH Q22

A.22.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
select
        cntrycode,
        count(*) as numcust,
        sum(c_acctbal) as totacctbal
from
        (
                select
                        substring(c_phone from 1 for 2) as cntrycode,
                        c_acctbal
                from
                        customer
```

```
            where
                    substring(c_phone from 1 for 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 from 1 for 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;
```

A.22.2 Seed Query produced by XRE.

```
Select c1.c_phone as cntrycode, <unknown> as numcust, c1.c_acctbal as totacctbal
 From customer c1, customer c2, orders
 Where c2.c_acctbal < c1.c_acctbal
 and (c1.c_phone LIKE '30%' OR c1.c_phone LIKE '13%' OR c1.c_phone LIKE '31%' OR c1.c_phone LIKE '17%' OR
 c1.c_phone LIKE '18%' OR c1.c_phone LIKE '23%' OR c1.c_phone LIKE '29%')
 and (c2.c_phone LIKE '30%' OR c2.c_phone LIKE '13%' OR c2.c_phone LIKE '31%' OR c2.c_phone LIKE '17%' OR
 c2.c_phone LIKE '18%' OR c2.c_phone LIKE '23%' OR c2.c_phone LIKE '29%')
 and c2.c_acctbal >= 0.01
```

A.22.3 **Business Description Text.** This query counts how many customers within country codes among '13', '31', '23', '29', '30', '18', and '17' have not placed orders for 7 years but who have a greater than average "positive" account balance. It also reflects the magnitude of that balance. Country code is defined as the first two characters of c_phone.

A.22.4 Output of Xpose.

```
SELECT
    SUBSTRING(c1.c_phone FROM 1 FOR 2) AS cntrycode,
    COUNT(DISTINCT c1.c_custkey) AS numcust,
    SUM(c1.c_acctbal) AS totacctbal
FROM
    customer c1
LEFT JOIN
    orders o ON c1.c_custkey = o.o_custkey
WHERE
    SUBSTRING(c1.c_phone FROM 1 FOR 2) IN ('13', '31', '23', '29', '30', '18', '17')
    AND c1.c_acctbal > (
        SELECT AVG(c2.c_acctbal)
        FROM customer c2
        WHERE c2.c_acctbal > 0
          AND SUBSTRING(c2.c_phone FROM 1 FOR 2) IN ('13', '31', '23', '29', '30', '18', '17')
    )
```

```
    AND o.o_orderkey IS NULL
GROUP BY
    cntrycode
ORDER BY
    cntrycode;
```

A.23 E-TPCH Q23

A.23.1 Ground Truth Query $Q_{\mathcal{H}}$.

```
SELECT   RIGHT(c_address, 5) AS city,
         p_brand             AS part_brand
FROM     customer,
         orders o1,
         orders o2,
         store_lineitem,
         web_lineitem,
         part
WHERE    c_custkey = o1.o_custkey
AND      c_custkey = o2.o_custkey
AND      o1.o_orderkey = wl_orderkey
AND      wl_returnflag = 'A'
AND      o2.o_orderkey = sl_orderkey
AND      sl_returnflag = 'N'
AND      wl_partkey = sl_partkey
AND      sl_partkey = p_partkey
AND      o1.o_orderdate < o2.o_orderdate
AND      wl_receiptdate < sl_receiptdate
AND      o1.o_orderdate BETWEEN date '1995-01-01' AND      date '1995-12-31'
AND      o2.o_orderdate BETWEEN date '1995-01-01' AND      date '1995-12-31'
GROUP BY RIGHT(c_address, 5),
         p_brand
ORDER BY city, part_brand;
```

A.23.2 Seed Query produced by XRE.

```
SELECT   c_address AS city,
         p_brand             AS part_brand
FROM     customer,
         orders o1,
         orders o2,
         store_lineitem,
         web_lineitem,
         part
WHERE    c_custkey = o1.o_custkey
AND      c_custkey = o2.o_custkey
AND      o1.o_orderkey = wl_orderkey
AND      wl_returnflag = 'A'
AND      o2.o_orderkey = sl_orderkey
AND      sl_returnflag = 'N'
AND      wl_partkey = sl_partkey
AND      sl_partkey = p_partkey
AND      o1.o_orderdate < o2.o_orderdate
AND      wl_receiptdate < sl_receiptdate
AND      o1.o_orderdate BETWEEN date '1995-01-01' AND      date '1995-12-31'
AND      o2.o_orderdate BETWEEN date '1995-01-01' AND      date '1995-12-31'
GROUP BY c_address,
         p_brand
ORDER BY city, part_brand;
```

A.23.3 **Business Description Text.** Find the cities and part brands where a customer first buys and returns on web, and then buys again from store. City is identified as the last 5 characters of customer's address.

A.23.4 **Output of Xpose.**

```
SELECT   RIGHT(c_address, 5) AS city,
         p_brand             AS part_brand
FROM     customer,
         orders o1,
         orders o2,
         store_lineitem,
         web_lineitem,
         part
WHERE    c_custkey = o1.o_custkey
AND      c_custkey = o2.o_custkey
AND      o1.o_orderkey = wl_orderkey
AND      wl_returnflag = 'A'
AND      o2.o_orderkey = sl_orderkey
AND      sl_returnflag = 'N'
AND      wl_partkey = sl_partkey
AND      sl_partkey = p_partkey
AND      o1.o_orderdate < o2.o_orderdate
AND      wl_receiptdate < sl_receiptdate
AND      o1.o_orderdate BETWEEN date '1995-01-01' AND date '1995-12-31'
AND      o2.o_orderdate BETWEEN date '1995-01-01' AND date '1995-12-31'
GROUP BY RIGHT(c_address, 5),
         p_brand
ORDER BY city, part_brand;
```

A.24 E-TPCH Q24

A.24.1 **Ground Truth Query $Q_{\mathcal{H}}$.**

```
SELECT Right(c_address, 5) AS city
FROM   customer,
       orders o1,
       orders o2,
       store_lineitem,
       web_lineitem w,
       part,
       web_lineitem w1,
       partsupp ps1,
       partsupp ps2
WHERE  c_custkey = o1.o_custkey
       AND c_custkey = o2.o_custkey
       AND o1.o_orderkey = sl_orderkey
       AND sl_returnflag = 'A'
       AND o2.o_orderkey = w.wl_orderkey
       AND w.wl_returnflag = 'N'
       AND w.wl_partkey = sl_partkey
       AND sl_partkey = p_partkey
       AND w1.wl_partkey = p_partkey
       AND sl_receiptdate < w.wl_receiptdate
       AND o1.o_orderdate < o2.o_orderdate
       AND w.wl_suppkey = ps1.ps_suppkey
       AND w1.wl_suppkey = ps2.ps_suppkey
       AND ps2.ps_availqty >= ps1.ps_availqty
       AND o1.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
       AND o2.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
GROUP  BY Right(c_address, 5) ;
```

A.24.2 **Seed Query produced by XRE.**

```
SELECT c_address AS city
FROM   customer,
       orders o1,
       orders o2,
       store_lineitem,
       web_lineitem w,
       part,
       web_lineitem w1,
       partsupp ps1,
       partsupp ps2
WHERE  c_custkey = o1.o_custkey
       AND c_custkey = o2.o_custkey
       AND o1.o_orderkey = sl_orderkey
       AND sl_returnflag = 'A'
       AND o2.o_orderkey = w.wl_orderkey
       AND w.wl_returnflag = 'N'
       AND w.wl_partkey = sl_partkey
       AND sl_partkey = p_partkey
       AND w1.wl_partkey = p_partkey
       AND sl_receiptdate < w.wl_receiptdate
       AND o1.o_orderdate < o2.o_orderdate
       AND w.wl_suppkey = ps1.ps_suppkey
       AND w1.wl_suppkey = ps2.ps_suppkey
       AND ps2.ps_availqty >= ps1.ps_availqty
       AND o1.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
       AND o2.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
GROUP  BY c_address;
```

A.24.3 **Business Description Text.** Find the cities where the customer buys an item from the store and buys it again from web, where the initial purchase could have been made from the web as well. City is identified as the last 5 characters of customer's address.

A.24.4 **Output of** Xpose**.**

```
 SELECT RIGHT(c_address, 5) AS city
FROM   customer,
       orders o1,
       orders o2,
       store_lineitem,
       web_lineitem w,
       part,
       web_lineitem w1,
       partsupp ps1,
       partsupp ps2
WHERE  c_custkey = o1.o_custkey
       AND c_custkey = o2.o_custkey
       AND o1.o_orderkey = sl_orderkey
       AND sl_returnflag = 'A'
       AND o2.o_orderkey = w.wl_orderkey
       AND w.wl_returnflag = 'N'
       AND w.wl_partkey = sl_partkey
       AND sl_partkey = p_partkey
       AND w1.wl_partkey = p_partkey
       AND sl_receiptdate < w.wl_receiptdate
       AND o1.o_orderdate < o2.o_orderdate
       AND w.wl_suppkey = ps1.ps_suppkey
       AND w1.wl_suppkey = ps2.ps_suppkey
```

```
       AND ps2.ps_availqty >= ps1.ps_availqty
       AND o1.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
       AND o2.o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1995-12-31'
GROUP  BY RIGHT(c_address, 5);
```

B STACK Queries

B.1 STACK Q7

B.1.1 TX_Q. The query is counting how many unique users (by display name) have both made a significant contribution and shown community moderation behavior on the Stack Exchange platform – and have shared a valid website URL in their profile.

B.1.2 $Q_{\mathcal{H}}$.

```
select count(distinct account.display_name) from account,
so_user, badge b1, badge b2 where
account.website_url != '' and
account.id = so_user.account_id and

b1.site_id = so_user.site_id and
b1.user_id = so_user.id and
b1.name = 'Famous Question' and

b2.site_id = so_user.site_id and
b2.user_id = so_user.id and
b2.name = 'Constable' and
b2.date > b1.date + '7 months'::interval
```

B.2 STACK Q8

B.2.1 TX_Q. The query is counting the number of unique questions posted on the German Stack Exchange site that are part of a linked conversation involving two questions and meaningful engagement through comments and relevant technical tags.

B.2.2 $Q_{\mathcal{H}}$.

```
 select count(distinct q1.id) from
site, post_link pl, question q1, question q2, comment c1, comment c2,
tag, tag_question tq1, tag_question tq2
where
site.site_name = 'german' and
pl.site_id = site.site_id and

pl.site_id = q1.site_id and
pl.post_id_from = q1.id and
pl.site_id = q2.site_id and
pl.post_id_to = q2.id and

c1.site_id = q1.site_id and
c1.post_id = q1.id and

c2.site_id = q2.site_id and
c2.post_id = q2.id and

c1.date < c2.date and

tag.name in ('jquery', 'android', 'excel', 'iphone', 'sql-server') and
tag.id = tq1.tag_id and
tag.site_id = tq1.site_id and
tag.id = tq2.tag_id and
tag.site_id = tq1.site_id and
```

```
tag.site_id = pl.site_id and

tq1.site_id = q1.site_id and
tq1.question_id = q1.id and
tq2.site_id = q2.site_id and
tq2.question_id = q2.id;
```

B.3 STACK Q9

B.3.1 TX_Q. This query counts the number of unique user accounts on Stack Overflow who posted questions tagged with "perl" after January 1, 2014, which received no answers. It filters for users with high reputation and who have provided a website URL in their profile. The result gives insight into engaged Perl question askers whose questions went unanswered despite their active presence.

B.3.2 $Q_{\mathcal{H}}$.

```
select count(distinct account.id) from
account, site, so_user, question q, post_link pl, tag, tag_question tq where
not exists (select * from answer a where a.site_id = q.site_id and a.question_id = q.id) and
site.site_name = 'stackoverflow' and
site.site_id = q.site_id and
pl.site_id = q.site_id and
pl.post_id_to = q.id and

tag.name = 'perl' and
tag.site_id = q.site_id and

q.creation_date > '2014-01-01'::date and

tq.site_id = tag.site_id and
tq.tag_id = tag.id and
tq.question_id = q.id and

q.owner_user_id = so_user.id and
q.site_id = so_user.site_id and
so_user.reputation > 67 and

account.id = so_user.account_id and
account.website_url != '';
```

B.4 STACK Q10

B.4.1 TX_Q. This query counts how many initial questions on the "aviation" site are part of a three-question chain, where each is linked to the next and all three have received at least one comment. It also ensures that the first question in the chain has a higher score than the last. This helps identify well-received questions that started meaningful discussions across multiple linked posts.

B.4.2 $Q_{\mathcal{H}}$.

```
select count(distinct q1.id) from
site, post_link pl1, post_link pl2, question q1, question q2, question q3 where

site.site_name = 'aviation' and
q1.site_id = site.site_id and
q1.site_id = q2.site_id and
q2.site_id = q3.site_id and

pl1.site_id = q1.site_id and
pl1.post_id_from = q1.id and
pl1.post_id_to = q2.id and

pl2.site_id = q1.site_id and
pl2.post_id_from = q2.id and
```

```
pl2.post_id_to = q3.id and

exists ( select * from comment where comment.site_id = q3.site_id and comment.post_id = q3.id ) and
exists ( select * from comment where comment.site_id = q2.site_id and comment.post_id = q2.id ) and
exists ( select * from comment where comment.site_id = q1.site_id and comment.post_id = q1.id ) and

q1.score > q3.score;
```