XPOSE: Bi-directional Engineering for Hidden Query Extraction

Ahana Pradhan, Jayant Haritsa
Indian Institute of Science
Bangalore, Karnataka, India
{ahanapradhan,haritsa}@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 SPJGAOL 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 TPC-H benchmark.

A two-pronged approach is taken – on the one hand, the existing RE scope is substantially extended to incorporate *union* connectors, *algebraic* filter predicates, and *disjunctions* for both values and predicates. On the flip side, 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, aggregate function *coefficients*, and custom *non-relational* functions. In essence, FE establishes the broad query contours, while RE fleshes out the fine-grained details.

We have evaluated XPOSE on eTPCH, a query suite comprising the complete TPCH benchmark extended with queries featuring unions, diverse join types, and sub-queries. The experimental results demonstrate that its bi-directional engineering approach correctly and efficiently extracts these complex queries, representing a significant step forward in HQE coverage.

Contents

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 [22], Scythe [23], FastQRE [7], REGAL [18, 19], SQUARES [13], PATSQL [17], SICKLE [27], CUBES [2]) have been developed over the past decade.

Hidden Query Extraction

A variant of QRE, called Hidden Query Extraction (HQE), was introduced in Sigmod 2021 [9]. 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 curated series of *input-output examples* produced by this executable.

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 [15, 16] (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 [9], by introducing the **UNMASQUE** tool, which is capable of (non-invasively) extracting flat **SPJGAOL** (Select, Project, Join, GroupBy, Agg, OrderBy, Limit) queries. The extraction operates in the linear pipeline shown in Figure 1. Here, a clause-by-clause extraction, starting with From and ending with Limit, of the hidden query is implemented 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, a minimization of the sample input D_I to a handful of rows is carried out at the beginning of the pipeline.

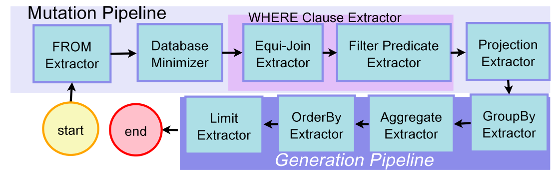


Figure 1: Linear Extraction Pipeline of Unmasque [8, 9]

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 TPC-H benchmark queries, most of which go well beyond flat SPJGAOL? Our analysis showed that the RE-based techniques could be extended to include complex operators, such as Unions and Algebraic Predicates, as described later in this paper. But we also found them to be fundamentally incapable of extracting common query constructs, such as nestings (in From and Where clauses), and custom non-relational functions (e.g. string operators).

The above limitations appear to preclude the extraction of realistic queries. However, it 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]. And what we show in this paper is that a judicious synergy of forward and reverse engineering is capable of precisely extracting the *entire TPC-H benchmark* and more. In a nutshell, FE establishes the broad contours of the query and the non-relational components, 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 $\mathcal{A}$, 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*. This seed query is sent to the **XFE** *(*Xpose*-FE)* module, which contrasts it against TX_Q, and iteratively refines it, using feedback prompting techniques, into a semantically equivalent version of $Q_{\mathcal{H}}$ through query synthesis. The refinement iterates until either of the following occurs: (1) the synthesized query matches $Q_{\mathcal{H}}$ wrt the execution results on D_I, (2) it does not synthesize any new formulation, (3) it exceeds a threshold number of iterations. When the LLM stops due to the latter two cases, we move on to a *combinatorial synthesizer* that searches through all valid combinations of tables within the nesting structure constructed by the LLM. Since the query table cardinality is usually a limited number, the associated computational overheads are expected to be manageable.

Finally, we verify the correctness of the extraction. Similar to Unmasque, we use XData [3] tool to generate small test databases to differentiate $Q_{\mathcal{E}}$ and black-box $Q_{\mathcal{H}}$. Empty output database from XData implies no semantic difference. However, due to the limited scope of this tool, such verification may not be possible for all cases. So, we resort to result equality checking on multiple randomized databases to minimize false positives.

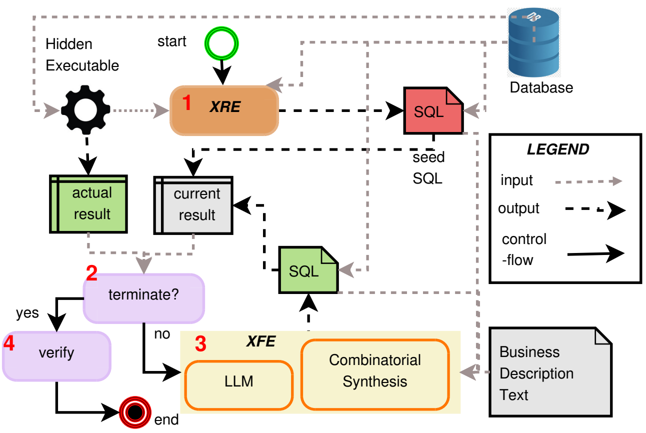


Figure 2: Architecture of Xpose

Illustrative Extraction

To illustrate the above process, consider TPCH **Q20** in Figure 3(a) with encryption. The business description [20] of this so-called "Potential Part Promotion Query", is listed in Figure 3(b).

The XRE output of Xpose on the executable is shown in Figure 3(c). We see here that it correctly extracts the five base tables, the joins between these tables, filter predicates on `l_shipdate`, `p_name` and `n_name`, projection columns, and result ordering. On the downside, however, there are several errors – 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 sub-queries are incorrectly characterized as global joins. Lastly, semi-joins, constructed using subqueries, are extracted as joins.

The XFE component is now prompted with the XRE output (Figure 3(c)) as a seed query $Q_{\mathcal{S}}$, and 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 aligned with $Q_{\mathcal{S}}$ and TX_Q. As a simple instance, the LLM's schematic scope is explicitly restricted to the tables in the seed query 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 listed in Figure 3(e). Finally, as per the verifier module, this query is functionally equivalent to $Q_{\mathcal{H}}$.

Contributions

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

- Non-invasive RE algorithms to extract unions, algebraic predicates, and disjunctions of predicates.
- FE using LLMs on a seed query augmented with automated corrective feedback prompts. The prompts are designed to correct the errors potentially introduced by RE.
- A novel amalgamation of FE and RE to extract complex SQLs, implemented in Xpose.
- Detailed evaluation of Xpose on E-TPCH, a query suite featuring the complete TPCH benchmark extended with additional queries for capturing diverse join types (many-to-many joins, non-equi-joins), and unions of sub-queries; and STACK database queries featuring high number of joins and disjunction predicates.

Organization

The mutation framework of Unmasque is reviewed in Section 2. In Section 3, we first describe the high-level design choices made for Xpose, followed by the building blocks of XRE and XFE. Detailed XRE extraction strategies for union and algebraic predicates are presented in Sections 4 and 5, respectively. The use of XFE to correct seed query errors is detailed in Section 6. The experimental framework and results are presented in Section 7. Our conclusions and future research avenues are summarized in Section 8.

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

Table 1: Notations

2 Mutation Framework of Unmasque [9]

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.

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.

(a) **Ground Truth 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**
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) **XRE Output of Seed Query**
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 Prompt 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 Extracted Output of XPOSE (XRE + XFE) :
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

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 [9]. Interestingly, it was proved in [9] 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 Satisfying Values

The term Satisfying values or *s-values* was introduced to refer to, for each column, the range of its values that satisfy $Q_{\mathcal{H}}$, and contribute towards producing a populated result. For example, a filter predicate $col_x \leq X$, with interval $[i_{min}, i_{max}]$ as the domain of col_x and a constant X from this domain, restricts the s-value interval of col_x to $[i_{min}, X]$. From Figure 5(a), the extracted satisfying value for *c_acctbal* is bound by the interval $[i_{min}$, 9999.99]. For both *c_custkey* and *o_custkey*, s-value is 23074 due to the join.

2.4 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* $\leq$ 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, 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 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.

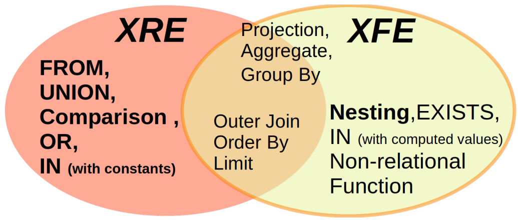


Figure 6: Extraction Scopes of XRE and XFE

For the case shown in Figure 3, if we employ only XFE to carry out the synthesis from scratch, it makes several errors – the spurious introduction of REGION table in the outer query, creation of two instances of LINEITEM and spurious ORDERS table, applying temporal predicate on `o_orderdate` instead of `l_shipdate`, using wrong attribute `l_quantity` in the aggregate comparison, spurious GROUPBY clause etc. On the other hand, given the seed query listed in Figure 3(c), XFE was able to confine its formulation to the set of tables and attributes without redundancy, as in the seed query, 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, existential operators (such as IN operator with query computed values), and non-relational functions can only be handled by XFE. Whereas operators such as PROJECTION, AGG, and GROUPBY 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, ORDERBY and LIMIT clauses, can all be extracted by *either* XRE or XFE. The advantage of XRE is provable correctness whereas that of XFE is performance benefits. In our implementation, we choose to have comparisons handled by XRE, whereas the remainder are delegated to XFE.

3.2 Running Example Query

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

3.3 Mutation Framework in XRE

3.3.1 From Clause Extractor. The EbE (Extraction-by-Error) strategy is sufficient for extracting T_E, the set of tables in $Q_{\mathcal{H}}$. However, for Unions, we need this information at the granularity of *subqueries.* This requires bringing the EbV (Extraction-by-Voiding) strategy also into play. In Section 4, the fine-grained EbV designed for union extraction is elaborated with the running example query.

Business Description: 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 the order price by considering all the incomplete same-day-delivery orders. All the customers having no orders are included in this list irrespective of their balance.

```
Q_H: SELECT *
  FROM (
        ( 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
   q1   )
      UNION ALL
        ( SELECT s_name AS name, s_phone AS phone
              FROM supplier
              WHERE s_suppkey IN (
                  SELECT l_suppkey
                  FROM orders, lineitem
   q2         WHERE l_orderkey = o_orderkey AND s_acctbal <= o_totalprice
              AND l_commitdate = l_receiptdate AND o_orderstatus IN ('P', 'A')
              )
        )
  ) AS people GROUPBY name, phone
```

Figure 7: Hidden Query highlighting new extraction scope

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 at least one pair 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 and the minimizer also ensures retention of this characteristic.

For example, for subquery q_1 listed in Figure 4, unless the notion of FIT-result set is applied, the D^1 shown in Figure 5(a) cannot be obtained. However, when the database minimization process looks for $Q_{\mathcal{H}}$ producing FIT-results after each step of recursive halving, the same D^1 is obtained. The same $\mathcal{R}_{\mathcal{H}}$ shown in Figure 5(b) is the FIT result obtained from the D^1 by the outer-join query q_1.

3.3.3 S-Value Extractor. The binary search-based mutation discussed in Section 2.4 is employed as the s-value extractor in XRE, with the determination of satisfaction wrt FIT-results.

When $Q_{\mathcal{H}}$ is restricted to arithmetic predicates, the s-value bounds correspond to the predicate constants in the query (Section 2), i.e. they are *static*. However, this is no longer the case if $Q_{\mathcal{H}}$ has algebraic predicates such as $col_x < col_y$. Consider $D^1.col_y = Y$, for some $Y \in [i_{min}, i_{max}]$, then col_x has s-value interval $[i_{min}, Y]$, a *floating interval* due to its dependency on Y.

3.3.4 Algebraic Predicate Extractor. The floating dependency is identified later by mutating $D^1.col_y$ with a *pair* of values and checking whether the s-value interval changes across these values. Section 5 details the extraction of $s\_acctbal \leq o\_totalprice$ using the above mechanism.

With the above augmented mutation framework, the seed query output by XRE is shown in Figure 8. It captures the union, FROM and the predicates in WHERE clauses correctly.

```
Q_S: ( SELECT c_name AS name, c_phone as phone
     FROM customer, orders WHERE c_custkey = o_custkey
     AND c_acctbal <= 9999.99 OR o_orderkey IS NULL
     GROUPBY 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 o_orderstatus IN ('A', 'P')
     GROUPBY s_name, s_phone )
```

Figure 8: Seed query extracted by XRE for $Q_{\mathcal{H}}$ in Figure 7

3.4 LLM-based Synthesis Framework in XFE

Since XRE's extraction technique is based on D^1, it extracts a minimal conjunctive flat query $Q_{\mathcal{S}}$ for every nested $Q_{\mathcal{H}}$ so that $\mathcal{R}_{\mathcal{S}}$ is equal to $Q_{\mathcal{H}}$ on D^1. We now need to add the nested structure, outer join, and the 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. For instance, "any order" maps to IN, and the nested structure is inferred from it. As elaborated later in Section 6, 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 the seed query $Q_{\mathcal{S}}$ while performing its synthesis. For instance, in the above example, reusing the join, comparison and disjunctive predicates is crucial. Also, specific guidelines instruct the LLM on how to handle result mismatch between $Q_{\mathcal{S}}$ and $Q_{\mathcal{H}}$ on D_I. For example, moving the GROUPBY attributes to the outer query is driven by the guideline.

Finally, the extracted query post-XFE intervention is shown in Figure 9.

```
Q_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.c_custkey
     WHERE (c.c_acctbal <= 9999.99 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 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 o.o_orderstatus IN ('A','P')                )
  )
) as customer_supplier GROUPBY name, phone;
```

Figure 9: Final Synthesized query by XPOSE for $Q_{\mathcal{H}}$ in Figure 7

3.5 Multiple instances of a table

So far in the above content, we do not deviate from one underlying assumption of UNMASQUE, which is of having *no repeated instance of a table* in the FROM clause. However, queries in general have multiple instances of the same table, and hence, we need to incorporate their extraction in XPOSE.

- *Assumption:* When the FROM clause of a subquery has repeated instances of a table, the corresponding WHERE clause is devoid of self-joins.
- *Determination by XRE through Database Minimization:* When $Q_{\mathcal{H}}$ has multiple instances of the same table, a D^1 may not suffice as D_{min}. If the table currently being minimized does not produce FIT-result while having n ($n \geq 1$) rows, but *produced FIT-result while having* $2n$ *rows*, XRE keeps reducing $2n$ rows by removing one row from the table. A D_{min} is reached when it is minimal wrt $Q_{\mathcal{H}}$, i.e. removing any more tuple from the table stops producing FIT-result from $Q_{\mathcal{H}}$. In this case, the minimized table has more than one row. Therefore, XRE produces a $\mathbf{D}^{1+}$, i.e., some tables have more than a single row while others have a single row. The tables with more than rows are the ones with multiple instances in $Q_{\mathcal{H}}$. The tuples in such tables are individually mutated through their positional references,(e.g. `ctid` in PostgreSQL) so that their independent roles in $Q_{\mathcal{H}}$ are extracted.
- *Determination by XFE from the business description:* It is also possible that a single tuple from a table to satisfy both the instances of the table present in the query. In such cases, XRE may not be able to determine multiple occurrences of the table. E.g. TPCH Q21, has 3 instances of LINEITEM, but XRE identifies 2 (because of having a $\mathbf{D}^2$ as the D_{min}). In spite of this, the LLM was able to determine the use of the additional instance from the text description.
- In our experiments, D_{min}s obtained in the above manner were $\mathbf{D}^2$.

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.

- ***Extraction Goals***: 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:

$$\forall i, 0 \leq i \leq n, \texttt{AuxTables}(q_i) \cup \texttt{COMMON}(Q_{\mathcal{H}}) = \texttt{FROM}(q_i).$$

– 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, 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. The auxiliary tables $\texttt{AuxTables}(Q_{\mathcal{H}})$ is given by $T_{\mathcal{H}}$ − $\texttt{COMMON}(Q_{\mathcal{H}})$ by definition. The goal is to make partitions of $\texttt{AuxTables}(Q_{\mathcal{H}})$ so that each partition corresponds to a subquery of $Q_{\mathcal{H}}$, ensuring that every subquery has a matching partition. Therefore, if $\texttt{AuxTables}(Q_{\mathcal{H}})$ is an empty set, $Q_{\mathcal{H}}$ is a flat query, i.e. it does not have a union operator. Otherwise, we enumerate the power set of $\texttt{AuxTables}(Q_{\mathcal{H}})$ and test each member for the possibility to map to a *single subquery*. Note that, only the non-empty pure subsets of $\texttt{AuxTables}(Q_{\mathcal{H}})$ are valid candidates. Algorithm 1 is designed to partition $\texttt{AuxTables}(Q_{\mathcal{H}})$. Figure 10 depicts the overall flow.

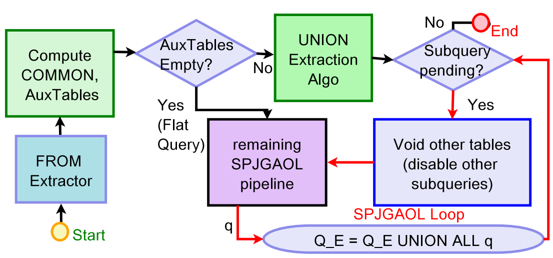


Figure 10: Flow of Union Detection and Extraction

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, $\texttt{AuxTables}(q_i)$ is a set of tables from $\texttt{AuxTables}(Q_{\mathcal{H}})$, any one of which when voided, gets q_i to produce a UNFIT-result. To identify such sets, we consider the power set of $\texttt{AuxTables}(Q_{\mathcal{H}})$, 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*

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

Input : $Q_{\mathcal{H}}$, D_I, $\texttt{AuxTables}(Q_{\mathcal{H}})$, $\texttt{COMMON}(Q_{\mathcal{H}})$

1 CoreTables ← $\texttt{AuxTables}(Q_{\mathcal{H}})$
2 SideTables, Max-SideTables, Aux, *Froms* ← Empty sets
3 U ← PowerSet($\texttt{AuxTables}(Q_{\mathcal{H}})$) − {∅, $\texttt{AuxTables}(Q_{\mathcal{H}})$}
   `// exclude ⊤ and ⊥ from the lattice`
4 U_{asc} ← Sequence of sets in U in increasing size
5 **for** $u \in U_{asc}$ **do**
6   **if** *a subset of* u *is already in* CoreTables **then**
7   include u in CoreTables
8   **continue** `// skip checking for u`
9   D' ← void all tables in u in D_I
10   **if** $Q_{\mathcal{H}}$ (D') *is UNFIT* **then** include u in CoreTables
11   Revert Mutations done by Line 9
12 SideTables ← U− CoreTables
13 Max-SideTables ← $\{c | c \in$ SideTables, $\forall t \in \texttt{AuxTables}(Q_{\mathcal{H}})$ - $\{c\}$, $c \cup \{t\} \in$ CoreTables}
14 **for** *each* $c \in$ *Max*-SideTables **do**
15   include set $\texttt{AuxTables}(Q_{\mathcal{H}}) - c$ in Aux
16 **for** *each* $p \in$ Aux **do**
17   $from_i$ ← $p \cup \texttt{COMMON}(Q_{\mathcal{H}})$
18   include $from_i$ in *Froms*
19 **return** *Froms*

$Q_{\mathcal{H}}$ = (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)}	
$\texttt{COMMON}(Q_{\mathcal{H}}) = \{o\}$	$\texttt{AuxTables}(Q_{\mathcal{H}}) = \{s, c, l\}$
$U_{desc} = (\{s, c\}, \{s, l\}, \{c, l\}, \{s\}, \{c\}, \{l\})$	
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 7

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 7. For instance, $\{s, c\} \in$ CoreTables because voiding Supplier and Customer together discards the satisfying tuples for both the subqueries in $Q_{\mathcal{H}}$. On the other hand, $\{s, l\} \in$ SideTables because voiding Supplier and Lineitem together discards the satisfying tuples only for the second subquery.

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 4-8 of the algorithm capture it.

Next, we identify the *maximal members* of SideTables. Adding any table from $\texttt{AuxTables}(Q_{\mathcal{H}})$ 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 Iterative Subquery Extraction. The subqueries of $Q_{\mathcal{H}}$ are extracted in a loop after the correct partitioning of $\texttt{AuxTables}(Q_{\mathcal{H}})$. In each iteration i, all the tables other than $\texttt{FROM}(q_i)$ are voided, to ensure that execution of $Q_{\mathcal{H}}$ produces result only from q_i. Then the UNMASQUE pipeline is used to extract the SPJGAOL-subquery.

4.3.5 Mutation Overheads. Executing $Q_{\mathcal{H}}$ for the power set enumeration in Algorithm 1 creates runtime bottleneck. However, in practice, $|\texttt{AuxTables}(Q_{\mathcal{H}})|$ is small (e.g. the number of fact-tables in a schema). Moreover, the execution time of $Q_{\mathcal{H}}$ is also reduced using correlated sampling [26], as described 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 [26]. 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 $\texttt{COMMON}(Q_{\mathcal{H}})$ to each members of set Aux, we prove the following: For any $p \in \texttt{Aux}$, voiding all the table in $\overline{p}$ (i.e. $\overline{p} = \texttt{AuxTables}(Q_{\mathcal{H}}) - p$) 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. $\overline{p} \in \texttt{CoreTables}$. Since $p \in \texttt{Aux}$, $\overline{p} \in$ Max-$\texttt{SideTables}$, 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, $\overline{p} \cup pd_{12} \in \texttt{SideTables}$. But $\overline{p} \in$ Max-$\texttt{SideTables}$ since $p \in \texttt{Aux}$ is given. Therefore, $\overline{p} \cup pd_{12}$ must belong to $\texttt{CoreTables}$, 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. *$\forall i, 1 \leq i \leq n$, $\texttt{FROM}(q_i)$ is a member of set Froms.*

PROOF. Let q_0 be an omitted subquery. So, $\texttt{AuxTables}(q_0)$ is not included in set Aux. Therefore, $\overline{\texttt{AuxTables}(q_0)} \notin$ Max-$\texttt{SideTables}$. So, it is either (1) in $\texttt{CoreTables}$; or (2) in $\texttt{SideTables}$, and some superset of it in Max-$\texttt{SideTables}$. 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 $\overline{\texttt{AuxTables}(q_0)}$ is to include at least one member from $\texttt{AuxTables}(q_0)$. 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, $\texttt{AuxTables}(q_i) \in \texttt{Aux}$, $\forall i, 1 \leq i \leq n$. Therefore, following the construction of the set *Froms* in the algorithm, $\texttt{FROM}(q_i) = \texttt{AuxTables}(q_i) \cup \texttt{COMMON}(Q_{\mathcal{H}})$ is in set *Froms*, $\forall i, 1 \leq i \leq n$.

4.6 Logical Disjunction in Predicates

Now we turn our attention to extract the logical disjunctions present in the WHERE clause of subqueries in $Q_{\mathcal{H}}$, denoted by $\texttt{WHERE}(q_i)$.

4.6.1 Assumptions. We assume $\texttt{WHERE}(q_i)$ is *conjunction of disjunctions of multiple predicates.* Predicate extraction is discussed in Section 5. We keep it abstract now, assuming that it is correct.

4.6.2 Predicate Types. Individual predicates in $\texttt{WHERE}(q_i)$ play a similar role in subquery q_i that is played by the relations in $Q_{\mathcal{H}}$. Consequently, we refer to the predicates using terms similar to those in Section 4.1. The `CommonPredicates` are the predicates in $\texttt{WHERE}(q_i)$ if any of them is not satisfied, q_i produces UNFIT-result. The other predicates are `AuxPredicates`, i.e. when none of them is satisfied, then only q_i produces UNFIT-result. Therefore, the problem is to identify the `CommonPredicates` and `AuxPredicates` in $\texttt{WHERE}(q_i)$.

4.6.3 Extracted Predicates from D^1. Since $Q_{\mathcal{H}}$ produces FIT-result on D^1, `CommonPredicates` remains satisfied. The same may not necessarily hold for the `AuxPredicates`. For instance, in Figure **??**, `o_orderstatus` the value 'A' in D^1, which gives no clue about the other possible values of this attribute. Therefore, the extracted conjunctive predicates from D^1 (by Section 5) contain both `CommonPredicates` and `AuxPredicates`.

4.6.4 Partitioning of a Table by a Predicate. A predicate partitions a table into two disjoint parts, one that satisfy the predicate and the other that does not. E.g. `o_orderstatus = 'A'` divides table CUSTOMER into two, one with all the tuples having `o_orderstatus = 'A'`, and the other with the tuples having `o_orderstatus != 'A'`. A (purely conjunctive) query with `o_orderstatus = 'A'` predicate produces FIT-result on the former partition, while produces UNFIT-result on the latter.

4.6.5 Extraction of Disjunction through Database Mutation. Therefore, after a predicate is extracted (by Section 5) from D^1, we mutate the database so that the corresponding table holds the partition that does not satisfy the predicate. For our running example in Figure **??**, XRE extracts `o_orderstatus = 'A'` first. Therefore, we mutate CUSTOMER as follows:

```
Alter Table Orders Rename to Orders_dummy;
Create Table Orders as
  Select * From Cust_dummy Where o_orderstatus != 'A';
```

If q_2 produces UNFIT-result on this database, it implies that the predicate is a `CommonPredicate`. On the other hand, if q_i produces FIT-result on this database, it implies that the predicate is an `AuxPredicate`, i.e. it is in disjunction. Query q_1 listed in Figure **??** still produces FIT-result on the above database. The existing mutation pipeline (database minimizer, s-value extractor, and Section 5) is then run to extract an *alternative predicate*, which is `o_orderstatus = 'P'` for us. This process is repeated for every predicate till no more `AuxPredicate` is found. The iteration shown using the "disjunction loop" in Figure 11 depicts it. IN operator extraction is covered by this technique.

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, $1 \leq i \leq n$ 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 $\{=, \leq, \geq\}$

(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 $\{=, \leq, \geq, <, >\}$

5.1.1 Operators. The operator set $\{=, \leq, \geq, <, >, \neq\}$ 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 $col_x \leq 4$, considering col_x as an integer. Consequently, $\leq$ and $\geq$ operators are sufficient to cover $<$ and $>$ respectively. Operator $\neq$ 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.

5.2 Extraction Process Overview

After obtaining D^1 that produces a FIT-result from $Q_{\mathcal{H}}$, s-value extractor $\mathcal{S}i$ is run once to get the s-value intervals of all the attributes, denoted by $\mathcal{S}_E$. In Figure 12(a), D^1 for the q_2 subquery of Figure 7 is listed. $\mathcal{S}_E$ 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 $\mathcal{S}_E$ as inputs, the σ_p-extraction algorithms are executed. Figure 11 depicts the overall flow. We have devised two different algorithms to extract the equality and inequality predicates [14]. This process picks out some members from the input set $\mathcal{S}_E$. The remaining s-value intervals in $\mathcal{S}_E$ are finalized as the arithmetic predicates (col_x, op, v_x). After a set σ_p gets extracted, iterations of the same pipeline is made to identify disjunctive predicates of *each* predicate in σ_p. This process is explained in [14], and skipped in this paper due to space constraint.

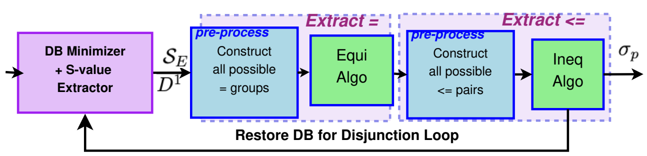


Figure 11: Flow of Algebraic Predicate Extraction

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.*

(a) Minimized Database D^1

Table: D^1.LINEITEM

l_orderkey	*l_commitdate*	*l_receiptdate*	*l_suppkey*
1448519	1992-09-23	1992-09-23	2032

Table: D^1.ORDERS

o_orderkey	*o_totalprice*	*o_orderstatus*
1448519	197740.95	A

Table: D^1.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 12: Showcasing D^1 and $\mathcal{S}_E$ for q_2 in Figure 7

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 2.4), to extract a common s-value interval for the MA-set under consideration. We use notation $\mathcal{S}i_g$ to denote this component.

5.3.3 **Basic Group Mutation.** The group s-value extractor $\mathcal{S}i_g$ 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 $\mathcal{S}i_g$ 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 $\mathcal{S}i_g$ 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 into two and apply $\mathcal{S}i_g$ 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 13 depicts the partitioning for the running example, where n is 4. If $\mathcal{S}i_g$ 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.

Algorithm 2: Equality Predicate Extraction

```
   Input   : Q_H, D^1, Si, Si_g, S_E
1  Pre-processing
2     Part_eq ← Empty Set
3     for each s-value interval [lb, ub] ∈ S_E do
4        Ignore the bounds which are domain boundaries
5     for each s-value equality interval [B, B] ∈ S_E do
6        M_A ← 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     σ_eq, Done ← Empty Sets
10    for each MA-set (B, M_A) ∈ 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| ≤ 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 Si_g 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 ← 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
```

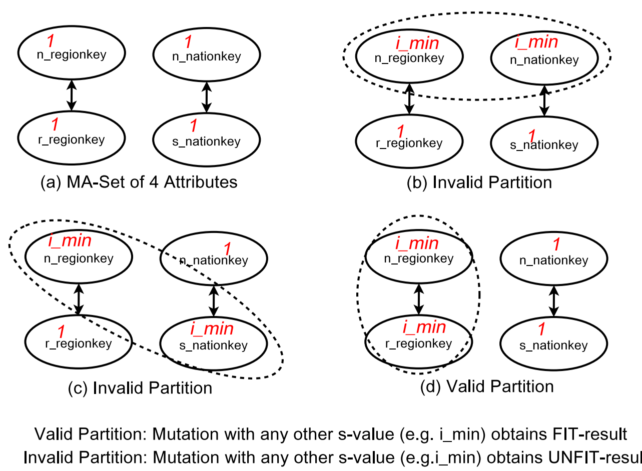


Figure 13: Partition MA-set ($n = 4$) into size i and $n - i$, for $i = 2$.

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 $col_1 \longrightarrow col_2 \longrightarrow \cdots \longrightarrow col_n$. 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.

Algorithm 3: Inequality Predicate Extraction

Input : $\mathcal{S}i$, $Q_{\mathcal{H}}$, D^1, C_E and $\mathcal{S}_E$ produced by $\mathcal{S}i$ on D^1

$E \leftarrow$ `Build_Edge_Set_E(`C_E, $\mathcal{S}_E$`)` (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, $\mathcal{S}i$, col_{src}, col_{snk}`)` (Section 5.4.2)
    $E \leftarrow$ `Extract_Static_LB(`D^1, col_{src}, $\mathcal{S}i$, E`)` (Section 5.4.3)
  $E \leftarrow$ `Extract_Static_UB(`D^1, seq, $\mathcal{S}i$, E`)` (Section 5.4.3)
return E

5.4.1 Enumerate Inequality Candidates (Edge-Set E).

With inputs D^1 and $\mathcal{S}_E$, we construct edge set E capturing all possible $\leq$ relationships among pairs of attributes. E is initiated from the static s-value bounds $\mathcal{S}_E$, as identified by $\mathcal{S}i$ 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 $D^1.col_x \leq \mathcal{S}_E.col_y$.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 4: Build Edge-Set E

Function `Build_Edge_Set_E(`C_E, $\mathcal{S}_E$`)`
  $E \leftarrow$ Empty set
  for *each s-value interval* $[lb, ub]$ *of* $col_x \in \mathcal{S}_E$ **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 $D^1.col_x \leq \mathcal{S}_E.col_y$.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 $col_x \longrightarrow col_y$ 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 $\mathcal{S}i$ is used on col_y to extract its bounds. If $\mathcal{S}_E.col_y$.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 14(a). In Figure 14(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, $s\_acctbal \leq o\_totalprice$.

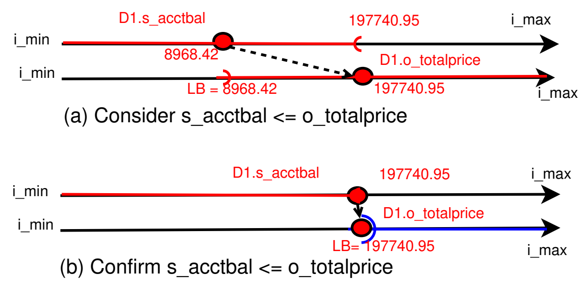


Figure 14: Identifying dependency between two attributes

Algorithm 5: Confirm LEQ

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

Algorithm 6: Extract Static Lower Bounds

Function `Extract_Static_LB(`D^1, col_{src}, $\mathcal{S}i$, E`)`
  $mut\_lb_{src} \leftarrow$ LB of col_{src} extracted by $\mathcal{S}i$ // Actual LB of col_{src}
  $min\_val_{src} \leftarrow val$ **if** $(val, col_{src}) \in E$ **else** i_{min} // Ideal LB of col_{src} as per E
  if $mut\_lb_{src} \neq min\_val_{src}$ **then**
    // There is a cut at the actual LB
    include $(mut\_lb_{src}, col_{src})$ in E
    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
  return E

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 $[i_{min}, i_{max}]$ 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

Algorithm 7: Extract Static Upper Bounds

Function Extract_Static_UB(D^1, seq, $\mathcal{S}i$, E)
  for i *from* $|seq|$ *down to* 1 **do**
    $col_c \leftarrow seq[i]$
    $mut\_ub_c \leftarrow$ UB of col_c extracted by $\mathcal{S}i$ `// Actual UB of colc`
    $max\_val_c \leftarrow val$ **if** $(col_c, val) \in E$ **else** i_{max} `// Ideal UB of colc as per E`
    if $mut\_ub_c \neq max\_val_c$ **then**
      `// There is a cut at the actual UB`
      include (col_c, mut_ub_c) in E
    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`
  return E

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 $col_1 \longrightarrow col_2 \longrightarrow \cdots \longrightarrow col_n$, 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 7 is thus extracted, as shown in Figure 8.

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 8 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 $l \longrightarrow col_1 \longrightarrow col_2 \longrightarrow \cdots \longrightarrow col_n \longrightarrow r$, where $i_{min} \leq l \leq r \leq i_{max}$, and $\longrightarrow$ 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 $col_i \longrightarrow col_{i+1}$, the only predicates in $Q_{\mathcal{H}}$ involving col_i is $l + \delta_2 * \Delta \longrightarrow col_i$, and involving col_{i+1} is $col_{i+1} \longrightarrow r - \delta_3 * \Delta$, 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, $\mathcal{S}_E.col_i.LB = l + \delta_2 * \Delta$, $\mathcal{S}_E.col_{i+1}.UB = r - \delta_3 * \Delta$. Assume the contradiction, $col_i \longrightarrow col_{i+1} \notin E$ 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}, $\mathcal{S}_E.col_i.UB = v_{i+1}$, $\mathcal{S}_E.col_{i+1}.LB = v_i$, which ensures that Algorithm 3(pre-processing) includes $col_i \longrightarrow col_{i+1}$ 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 $l \longrightarrow col_1 \longrightarrow col_2 \longrightarrow \cdots \longrightarrow col_n \longrightarrow r$, where $i_{min} \leq l \leq r \leq i_{max}$, and $\longrightarrow$ is either $\leq$ or $<$, and for any $col_i \longrightarrow col_{i+1}$ pair in the predicate chain, at least one of the static bounds $LB_{i+1} \longrightarrow col_{i+1}$ and $col_i \longrightarrow UB_i$ exists, Algorithm 3 extracts the predicate chain correctly.*

Proof. $UB_i < LB_{i+1}$ implies $col_i \longrightarrow col_{i+1}$ is a redundant predicate. Therefore, we prove that Algorithm 3 extracts $col_i \longrightarrow col_{i+1}$ when $LB_{i+1} \longrightarrow UB_i$. Now, it is given that $\mathcal{S}_E.col_{i+1}.LB = LB_{i+1}$, and $\mathcal{S}_E.col_i.UB = UB_i$. 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 $v_i \longrightarrow v_{i+1}$ due to σ_{chain}. The given static LB of col_{i+1} can only happen in the presence of σ_{chain} if $v_i \longrightarrow LB_{i+1}$. Due to our initial assumption, $v_i = LB_{i+1}$. Therefore, when $D^1.col_i$ is mutated with a higher

value UB_i, $\mathcal{S}_E.col_{i+1}.LB$ 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, $\mathcal{S}_E.col_{i+1}.LB$ 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 $\mathcal{S}_E.col_i.LB = lb_i$ 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 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.

6.1 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.

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 15 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 GROUPBY .
G15. If $\mathcal{R}_{\mathcal{S}}$ has fewer rows as compared to the actual output, consider either adding more GROUPBY attributes or having more GROUPBY clauses through nestings.

Figure 15: Guideline Instructions for Query Synthesis by XFE

For the example $Q_{\mathcal{S}}$ in Figure 8, 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 GROUPBY operator to the outer query is triggered by G14.. In this manner, the prompts direct the LLM towards an accurate extraction.

(a) Initial Prompt [IP]
You are an expert in formulating SQL queries from high-level business descriptions.
Formulate SQL to answer the following:<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 $Q_{\mathcal{H}}$)
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.
(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

6.2 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.

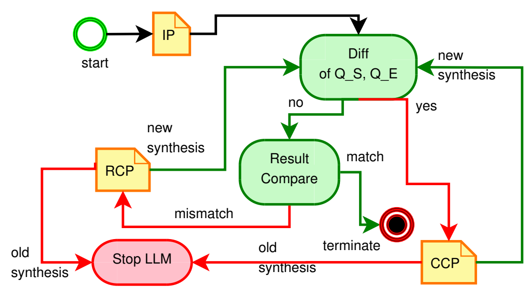


Figure 16: Prompting Flow in XFE

The overall prompting flow in XFE is shown in Figure 16. 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 7), we carry out an enumerative combinatorial synthesis, described next.

6.3 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 redistributions 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 GroupBy attributes from the *base tables*. Further, that two layers of GroupBy 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 GroupBy , Select , and OrderBy clauses. These redistribution strategies are detailed, with examples, in [14].

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.

6.4 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 [24], SQLSolver [6]) 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 [3], 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.

7 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 [12] LLM, configured with 0 temperature to minimize hallucinations, is the synthesis agent in the XFE module.

We present results for the following three complex query suites:

1. TPCH: The standard decision support benchmark [20], which models a data warehousing environment and features 22 analytic queries labeled Q1 through Q22. The business descriptions are taken from the official documentation [20].

2. E-TPCH: Only key-based equi-joins are modeled in TPCH, and there are no Unions of sub-queries. However, as highlighted in [5, 10], 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 [5] – the full details are in [14]. This extended benchmark is referred to as E-TPCH, and its textual inputs were created by mildly augmenting the corresponding TPCH descriptions.

3. STACK: A real-world benchmark from StackExchange [11] 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.

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 GROUPBY supp_nation, cust_nation, l_year
ORDERBY supp_nation, cust_nation, l_year;
```

Figure 17: E-TPCH Q7 Business description and SQL

Due to space limitations, 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, **24** for E-TPCH, and **4** for STACK.

7.1 TPCH Extraction

We begin by executing XPOSE on encrypted 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 6.4, and these results are shown in Table 4. We observe that 4 queries – **Q2, Q13, Q16, Q21** – could be successfully verified by XData [3], our best choice from a deployment perspective. For queries that were outside its scope, the probabilistic LLM-based and 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.

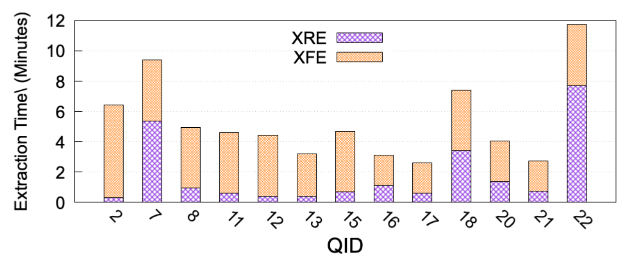
Equivalence Checker	*TPCH Query ID*
XDATA [3]	2, 13, 16, 21
QED [24]	18, 21
SQLSolver [6]	2, 21, 22

Table 4: Checkers for Extraction Correctness (TPC-H)

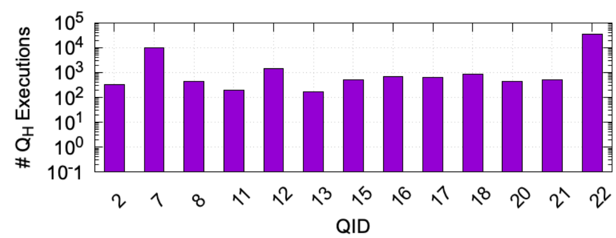
7.1.1 Extraction Overheads. We now turn our attention to the time overheads incurred by the extractions. These results are shown in Figure 18(a) and we find that the extractions are typically completed in less than ten minutes – the sole exception is Q22, which has a large number of disjunctions (7 each in two tables) and exclusion predicates, and even this "hard-nut" query is drawn out in 12 minutes. These overheads appear reasonable given that HQE is expected to be 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 with many string literals, such as in Q22. Extracting each literal requires one round of database minimization, shooting up the 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.



(a) Extraction Times



(b) Number of times $Q_{\mathcal{H}}$ is called by XRE

Figure 18: XPOSE Overheads for TPC-H Queries

7.1.2 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 18(b). We observe here that most of the queries take up to a few thousand invocations, but a couple (Q7 and Q22) go well beyond even this mark.

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

7.1.3 Synonymized Databases. A pertinent question that could be asked here is whether the good extraction performance is an *artifact* of GPT-4o being previously trained on TPC-H, which is publicly available. To assess this concern, we created a *synonymized* version of the benchmark (we choose synonymization rather than anonymization to ensure that the business descriptions continue to retain meaning). Specifically, the names of the tables and attributes were renamed using English synonyms, as well as synonyms from other languages. Correspondingly, the texts were also edited with the synonyms (the whole sentence remains in English).

The good news is that, despite these material changes, all the TPCH queries continued to be extracted correctly. However, they required a couple more iterations of the clause-correction prompts to reach extraction closure.

7.1.4 Choice of LLM. A second relevant question is whether GPT-4o could have been substituted with a smaller model. We tried out a variety of LLMs, including Llama 3.2 [21], Qwen 2.5 (both decoder-only and coder versions) [25], and DeepSeek-r1 [4] – but none of them were able to extract *any* of the queries. These results again suggest that industrial-strength HQE is a complex learning task requiring heavy-duty training and reasoning power.

7.2 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. We found equivalence in 6 extraction cases (the same queries reported in Table 4) by XDATA and the formal checkers. The remaining 17 "bi-directional" queries are outside the scope of the formal checkers – therefore, apart from our manual verification, we used the LLM-based and Result-equivalence-based probabilistic tests to confirm extraction correctness.

7.2.1 Extraction Overheads. The extraction times for the E-TPCH queries are shown in Figure 19. 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 due to having several string disjunctions and multiple instances of tables, respectively. We also observe that the time-split ratio between XRE and XFE is query-specific and covers a large range of values.

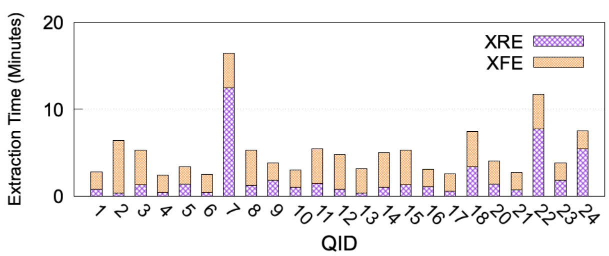


Figure 19: Extraction Overheads (E-TPCH)

7.3 Drill-down Analysis (E-TPCH)

The E-TPCH scenario foregrounds the need for bi-directional engineering, as shown in Table 5, 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.

Clause/Operator Type	*XRE extracted*	*XFE extracted*
T_E (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 GROUPBY (15)	3	12

Table 5: Extraction Distribution

7.3.1 XFE Prompts. Drilling down into XFE, the prompt sequences leading to successful extraction is shown in Table 6 on a per-query basis, along with the associated overall token counts. We find that all of the 23 queries, corresponding to rows 1 and 2 in the table, are completed within 4 prompts. In the remaining 3 queries, Q20 required additional clause-corrections, whereas Q2 and Q13 proved to be "prompt-resistant" after synthesizing nesting structures for the first time, requiring invocation of the computationally heavy Combinatorial Synthesis step as a last resort for extraction closure – Q2 required redistribution of tables, whereas Q13 required redistribution of GROUPBY attributes. All the queries were refined with less than 4000 tokens. Given the current pricing of \$2.5/*million* input tokens for GPT-4o [12], even these "hard-nut" cases cost less than a cent apiece.

Prompt Sequence	*QID*	*Total Token Count*
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 6: Prompts Sequence for Query Synthesis

7.3.2 Effectiveness of Guidelines. We observed that the LLM often deviates from the synthesis guidelines, especially with regard to confirming with the seed query Q_S on its provable extractions. Hence, in the feedback prompts, *query-specific instances* of the guidelines were required in several cases, in addition to the result mismatch reports. As a case in point, the Q_S for Q17 has tables FROM 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_S`: PART table 1 time, WEB_LINEITEM table 2 times`) to prevent it from continuing to do so. The opposite phenomenon was seen for Q24, where multi-instance tables in Q_S appeared only once in the subsequently synthesized FROM clause. However, in spite of such issues, repeated feedback prompting was sufficient to eventually resolve all 23 queries.

7.4 STACK Extraction

Our final experiment is on the 4 bi-directional STACK queries. All the 4 extractions could be verified using XDATA.

7.4.1 Overheads. The overheads incurred in the extractions are shown in Figure 20. In this limited set, XRE takes much more than XFE in the time-split ratio. The genesis of the XRE preponderance is due to the large number of joins (e.g. 19 joins in Q8), that feature commonly in STACK making the execution time of $Q_{\mathcal{H}}$ high on the 100 GB database. Self-joins and cyclic joins are the major complexities handled by XRE, whereas the primary contribution of XFE was to determine nested structure connected by existential operators.

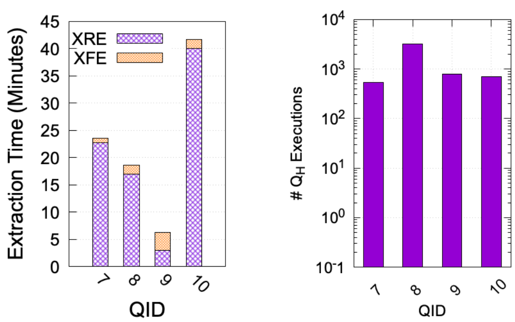


Figure 20: Extraction Overheads (STACK)

JH: Overhead figure needs to be corrected to reflect 100GB Stack

AP: Q7, Q9, Q10 data is complete. Q8 is on sampled data. Full DB execution times: Q7 -> 2mins 48sec, Q8 -> x, Q9 -> 1min 41sec, Q10 -> 6mins. To-Do: Q8

8 Conclusions

Reverse engineering SQL queries requires significant prior experience. For a human user, it corresponds to domain knowledge, understanding of the data, familiarity with the operational behaviors of the application, and SQL expertise. In our attempt for this purpose, we developed XRE, the reverse engineering module, that extracts the core operators of the opaque application deterministically to generate a sufficient seed query. Subsequently, XFE, the forward engineering module, uses one of the leading LLM as well as combinatorial query synthesis to refine the seed to match with the ground truth. Thus our bidirectional engineering of SQL formulation could bring the theoretical problem of HQE into the practical world, by extracting industry standard TPC-H analytical benchmark queries. Extracting such complexities in SQLs has never been attempted in the literature before, thereby points out the fundamental contribution of XPOSE.

A Appendix

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);
```

References

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

[2] 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.

[3] 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.

[4] DeepSeek-AI et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. Accessed: 2025-02-13.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] 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.

[14] A. Pradhan and J. Haritsa. XPOSE technical report, 2024.

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

[16] 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.

[17] 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.

[18] 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.

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

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

[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

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

[23] 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.

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

[25] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024.

[26] 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.

[27] 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.