Ranking-based Processing of SQL Queries

Hany Azzam, Thomas Roelleke, and Sirvan Yahyaei
Queen Mary, University of London, UK. E1 4NS
{hany,thor,sirvan}@eecs.qmul.ac.uk

ABSTRACT

A growing number of applications are built on top of search engines and issue complex structured queries. This paper contributes a customisable ranking-based processing of such queries, specifically SQL. Similar to how term-based statistics are exploited by term-based retrieval models, ranking-aware processing of SQL queries exploits tuple-based statistics that are derived from sources or, more precisely, derived from the relations specified in the SQL query. To implement this ranking-based processing, we leverage PSQL, a probabilistic variant of SQL, to facilitate probability estimation and the generalisation of document retrieval models to be used for tuple retrieval. The result is a general-purpose framework that can interpret any SQL query and then assign a probabilistic retrieval model to rank the results of that query. The evaluation on the IMDB and Monster benchmarks proves that the PSQL-based approach is applicable to (semi-)structured and unstructured data and structured queries.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.2.3 [**Database Management**]: Languages; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—Performance evaluation (efficiency and effectiveness)

General Terms

Algorithms, Languages, Performance

Keywords

PSQL (Probabilistic SQL), retrieval models, integrated DB+IR

1. INTRODUCTION

Applications built on top of search engines such as question answering and data mining are gradually leveraging rich language resources such as microformats (e.g."geo" and "hAtom") and RDF/RDFa markup which can be integrated with content-based resources [16]. Using these resources effectively requires complex structural representations of information needs [4]. Additionally, there may be retrieval environments where users are expert searchers willing to use structured query languages. Librarians, application developers, patent searchers and users of legal information systems often form complex queries.

Both search applications and expert searchers rely on information retrieval (IR) approaches that can support constraint-checking and ranking with respect to document structure and annotations as well as keywords. Such approaches enable direct expression of higher-level constraints on answer structures which can be checked at retrieval time. One approach is to use SQL, which is still the most widely used standard for formulating structured and semantic-expressive queries [23], and extend it with similarity predicates to perform the ranking, e.g. [5]. However, most of these approaches use physical SQL math and aggregation operators (e.g. log, sum) to express the ranking function. In other words, they specify the information need *and* the ranking function in the same query. From an abstraction point of view, this mix is *physical*, since bare SQL is used to implement the ranking. Other approaches, such as [6] implement the ranking function directly on top of a physical document representation. Such engineering approaches can also be problematic because they result in implementations that are difficult to maintain and re-use [11, 8, 12].

A general-purpose framework for ranking structured queries can address these issues by providing transparent and effective ranking. We propose such a framework that interprets *any* SQL query and then assigns a probabilistic retrieval model to rank the results of that query. The interpretation is part of a processing strategy that focuses on satisfying the information need while concurrently adding the model as part of the processing. To implement these models we use Probabilistic SQL (PSQL, [22]), which helps to express them in an abstract and logical way. The abstraction takes control of tuple (the results from the structured query) probability estimation and aggregation, which replaces the physical approaches that use SQL aggregation operators. The resulting implementation is easy to understand, debug and customise.

1.1 Contributions & Outline

To achieve the ranking-based processing of SQL, retrieval models (TF-IDF and language modelling) which are designed primarily for document retrieval are generalised for tuple-based retrieval. The main contribution of this paper is the SQL-to-PSQL algorithms which transform the SQL queries to PSQL programs. These programs deliver an IR-model-based ranking of tuples retrieved for the genuine SQL query and are evaluated using several benchmarks.

The remainder of the paper is organised as follows: Section 2 provides the essential background on probabilistic DB technology, retrieval models and PSQL. Sections 3 and 4 cover the core contributions of this paper: the generalisation from document retrieval to tuple retrieval, and the evaluation. Section 5 summarises the paper and concludes the discussion.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

2. BACKGROUND & PRELIMINARIES

2.1 Probabilistic Databases

The research in this paper draws upon advances in probabilistic data models. Early work extending relational and object-oriented data models using fuzzy sets and probability theory can be found in [3]. [17] discussed the notion of quality and its estimation in databases using a probabilistic approach; the relational model of data was extended and a quality specification was associated with each relation instance. [15] introduced RankSQL, which integrates ranking support into SQL. Another significant approach is the framework for probabilistic ranking of tuples based on a variant of the BIR retrieval model [6]. This framework, which measures the correlation between the specified and unspecified attributes to produce the ranking, differs from the framework proposed here. We support any probabilistic retrieval model and its variations, and the probability estimation need not be modelled outside the probabilistic framework. Efficient query evaluation in probabilistic databases is discussed by [9], which proposes a system that efficiently ranks the top-k answers to an SQL query on a probabilistic database. This system provides an optimisation technique referred to as "safe-plan" that facilitates the efficient computation of probabilistic results for queries. This technique also ensures that the correctness of the probabilistic semantics is maintained. Another approach for improving query processing times, proposed by [19], is based on materialising probabilistic views in probabilistic databases. Other approaches concerned with the efficient ranking of tuples, but geared towards specific types of data such as semi-structured, have been initiated. [24] computes approximate top-k results using probabilistic score estimators. As a result, retrieval efficiency improves without a major loss in effectiveness.

The probabilistic relational modelling concepts employed herein build upon the aforementioned research and the theoretical foundations by [10], [9] and [19]. The contribution of this paper is to leverage such concepts for the probabilistic ranking of tuples.

2.2 The Core Retrieval Models

This section reviews the core retrieval models (TF-IDF and LM) used in this paper. To define this models, let $N_D(c)$ denote the number of *Documents* in collection "c", and let $n_D(t,c)$ denote the number of *Documents* with term "t" in collection "c", where $\mathrm{df}_t := n_D(t,c)$ is the *document frequency*. Similarly, let $N_L(c)$ denote the number of *Locations* in collection "c", and let $n_L(t,c)$ denote the number of *Locations* with term "t". $N_L(d)$ and $n_L(t,d)$ are the *Location-based* counts for document "d", where $\mathrm{tf}_d := n_L(t,d)$ is the within-document *term-frequency*. The dual notation helps to achieve the unifying definitions of TF-IDF and LM shown next.

DEFINITION 1. *TF-IDF RSV:*

$$\mathrm{TF}(t,d) := \begin{cases} \frac{n_L(t,d)}{n_L(t,d)+K_d} & \textit{BM25-like TF norm.} \\ \frac{n_L(t,d)}{N_L(d)} & \textit{max-likelihood estimate} \end{cases} \quad (1)$$

$$\mathrm{IDF}(t,c) := \begin{cases} -\log_2 \frac{n_D(t,c)}{N_D(c)} \\ -\log_{N_D(c)} \frac{n_D(t,c)}{N_D(c)} \end{cases} \quad (2)$$

The TF-IDF term weight is a combination of TF and IDF values.

$$w_{\textit{TF-IDF}}(t,d,q,c) := \mathrm{TF}(t,d) \cdot \mathrm{TF}(t,q) \cdot \mathrm{IDF}(t,c) \quad (3)$$

The RSV is the sum over the TF-IDF weights.

$$\textit{RSV}_{\textit{TF-IDF}}(d,q,c) := \sum_{t \in d \cap q} w_{\textit{TF-IDF}}(t,d,q,c) \quad (4)$$

The definition captures traditional TF-IDF and some of the probabilistic and information-theoretic interpretations proposed in [7, 20]. $\mathrm{TF}(t,d)$ is the within-document term frequency component. $\mathrm{TF}(t,d)$ can be set to one of the following: the BM25-motivated quantification $\mathrm{tf}_d/(\mathrm{tf}_d + K_d)$, K_d is a normalisation factor reflecting the document length and is usually proportional to the pivoted document length ($\mathrm{pivdl} := \mathrm{dl} \,/\, \mathrm{avgdl}$); and the maximum-likelihood estimate $P_L(t|d) := n_L(t,d)/N_L(d)$, which estimates the within-document term probability ($\mathrm{dl} := N_L(d)$ is the document length, i.e. the number of locations in document d).

$\mathrm{IDF}(t,c)$ is the inverse document frequency component. This can be the negative logarithm of the term probability $P_D(t|c) := n_D(t,c)/N_D(c)$. $\mathrm{IDF}(t,c)$ can also be normalised, for example, $\mathrm{idf}(t,c)/\mathrm{maxidf}$. This corresponds to the logarithm to base $N_D(c)$, since the maximum idf value is $\mathrm{maxidf} := -\log 1/N_D(c)$ [22].

The setting of the $\mathrm{TF}(t,d)$ to the BM25-motivated quantification and the $\mathrm{IDF}(t,c)$ to the normalised (probabilistic) one are the settings used for the experiments in Section 4.

DEFINITION 2. *LM RSV: Language modelling (LM) [18] consists of two term probabilities: the within-document term probability $P(t|d)$ (foreground model) and the collection-wide term probability $P(t|c)$ (background model).*

$$P(t|d) := P_L(t|d) := \frac{n_L(t,d)}{N_L(d)} \left(= \frac{\textit{tf}(t,d)}{\sum_{t'} \textit{tf}(t',d)} \right) \quad (5)$$

$$P(t|c) := P_L(t|c) := \frac{n_L(t,c)}{N_L(c)} \left(= \frac{\textit{tf}(t,c)}{\sum_{t'} \textit{tf}(t',c)} \right) \quad (6)$$

The LM term weight is defined as follows:

$$w_{\textit{LM}}(t,d,c) := \log \left(1 + \frac{\lambda}{1-\lambda} \cdot \frac{P(t|d)}{P(t|c)} \right) \quad (7)$$

$$\textit{RSV}_{\textit{LM}}(d,q,c) := \sum_{t \in d \cap q} \mathrm{TF}(t,q) \cdot w_{\textit{LM}}(t,d,c) \quad (8)$$

The RSV is derived from the conditional probability $P(q|d)$, which is based on $P(q|d)/(P(q) \cdot \prod_{t \in q} (1-\lambda)^{\mathrm{TF}(t,q)})$, i.e. a query-based but document-independent normalisation of the query probability $P(q|d)$.The $0 < \lambda < 1$ is the mixture parameter. The query probabilities are decomposed as follows: $P(q|d,c) = \prod_{t \in q} P(t|d,c)^{\mathrm{TF}(t,q)}$ and $P(q|c) = \prod_{t \in q} P(t|c)^{\mathrm{TF}(t,q)}$, where $P(t|d,c)$ is the mixed term probability.

2.3 Probabilistic SQL (PSQL)

The syntax of PSQL is like SQL apart from a few additional syntactic elements. For example, in the PSQL SELECT statement, one can specify an *aggregation assumption* in front of the target list, an *estimation assumption* per source, and an *estimation assumption* for the SELECT statement. One feature of PSQL is, hence, the estimation assumption, which locates probability estimation within the probabilistic relational paradigm. This differs from probabilistic relational algebra [10] and approaches such as [1] where probability estimation is modelled outside the relational paradigm, a feature perceived as disadvantageous by both developers and designers.

Below we illustrate the most commonly used syntactic elements particular to PSQL and their definitions. Their theoretical underpinnings are described in [22], and to improve readability, a summarised version of the syntax of PSQL is illustrated in Figure 1.

DEFINITION 3. ***Aggregation assumption:*** *DISJOINT, INDEPENDENT and SUBSUMED are the main aggregation assumptions. For DISJOINT, the aggregation corresponds to the sum of probabilities. For INDEPENDENT, the aggregation is based on the usual set overlap: $P(A \vee B) = P(A) + P(B) - P(A) \cdot P(B)$. For SUBSUMED, the aggregation yields the maximal probability.*

```
-- PSQL query syntax (extract)
query ::= select_expr ';'
view ::= 'CREATE' 'VIEW' relName 'AS' select_expr ';'
select_expr ::=
'SELECT' [assumption] targetList
  'FROM' sourceList
  'WHERE' condition
  [ 'EVIDENCE' 'KEY' '(' key ')' ]
  [ 'ASSUMPTION' assumption ]
source ::= tradSrc | condSrc
tradSrc ::= relName | relName varName
condSrc ::= tradSrc '|' [assumption] '('key')'

key ::= | attrName | attrName ',' key
assumption ::= ... % see Definition 5
```

Figure 1: PSQL Syntax. Terminal symbols are indicated by single quotes.

DEFINITION 4. ***Evidence key:*** *An evidence key is a set of attributes. The notion "evidence" is borrowed from the conditional probability $P(h|e)$ where h is the hypothesis and e is the evidence.*

In the relational world, $P(\text{Term}|\text{Doc})$ is a probability that can be defined for a relation with attributes "Term" and "Doc". The set {Doc} is the evidence key. Evidence keys can be specified for the targets (SELECT targets) or for the sources (FROM source1, source2, ...) of an SQL query.

DEFINITION 5. ***Estimation assumption:*** *The estimation assumption is related to the evidence key and describes the way $P(h|e)$ is estimated. The main assumptions are: Disjoint, Independent, Subsumed and the so-called complex assumptions (e.g. VALUE_FREQ (VF), INVERSE_VALUE_FREQ (IVF), MAX_IVF, and similar for TUPLE_FREQ (TF)).*

One important feature of the estimation is whether the assumption is based on *value frequency*, or *tuple frequency*. Value frequency corresponds to the number of distinct relations in which a term occurs. For document-based retrieval (and IR in general), the *value* in a (Key, Value) pair corresponds to a *document*, i.e. (Term, Doc). Value-frequency-based estimation, for instance, is fundamental for estimating the IDF. An example of an assumption that facilitates such an estimation is the MAX_IVF assumption, which computes a probability according to the maximal inverse value frequency. For IR, this is the maximal IDF, i.e. MAX_IDF.

Tuple frequency, on the other hand, corresponds to the number of tuples (locations) in which a term occurs. For example, tuple-based probability corresponds to the so-called within-document term frequency (TF) of a term. This probability can be estimated using the TUPLE_FREQ (TF) assumption.

3. TUPLE RETRIEVAL

In [22] the application of PSQL to document retrieval was demonstrated. This lays the groundwork for this section where the generalisation from document retrieval to tuple retrieval is presented. For the demonstration, assume a term-based representation of queries and documents. The schema consists of two extensional relations, "qIdx(Term, QueryId)" and "docIdx(Term, DocId)". Given such a schema, document retrieval is:

```
-- Document retrieval in SQL
SELECT DISTINCT DocId, QueryId
  FROM qIdx Q, docIdx D
  WHERE Q.Term = D.Term;
```

A typical SQL query, as the one shown above, can be decomposed into two parts: an indexing part and a retrieval part. This decomposition, which facilitates the discussion about the ranking of tuples, is illustrated using the following SQL query on table "properties(Type, Area, Price)": *find the areas with flats or studios in the price range 200-250k.*

```
SELECT Area
  FROM properties
  WHERE (Price BETWEEN 200 AND 250)
  AND (Type='Flat' OR Type='Studio');
```

Step 1: Create a view to index the source "properties".

```
CREATE VIEW areaIndex AS
  SELECT Type, Area
  FROM properties
  WHERE Price BETWEEN 200 AND 250;
```

The view illustrates how an index ("areaIndex(Type, Area)") can be obtained for one attribute (here, the property type) and then used to retrieve areas with flats or studios in the price range 200-250k.

Step 2: Retrieve in document-retrieval fashion.

```
SELECT DISTINCT Area
  FROM areaIndex D
  WHERE D.Type = 'Flat' OR D.Type = 'Studio';
```

Figure 2 illustrates that this general decomposition helps to transform any SQL query into a form aligned with that of document retrieval. The significance of this alignment is that the ranking functions for document retrieval, such as the ones illustrated in the previous section, become applicable to tuple retrieval.

```
SELECT DISTINCT Doc
  FROM docIdx D
  WHERE D.Term = 'sailing' OR D.Term = 'east';

SELECT DISTINCT Area
  FROM areaIndex D
  WHERE D.Type = 'Flat' OR D.Type = 'Studio';
```

Figure 2: Alignment of Document and Tuple Retrieval

Having attained this aligned, or in other words, generalised, SQL structure, a translation process takes place through which a well defined ranking is attached to this SQL query. This translation process is the transformation of the generalised SQL structure into PSQL programs that implement ranking functions, such as TF-IDF or LM. Note that the original information need ("Find from table properties the areas ...") is intact; yet, instead of a set-based processing of the SQL query, a rank-based processing takes place.

In the following sections we will illustrate the SQL-to-PSQL (SQL2PSQL) translation process. To define the algorithms that automate the SQL2PSQL translation, we use the following notation. Each SQL query has a set of *conditioned attributes* (referred to as X), a set of *target attributes* (referred to as Y) and a *source relation(s)* (referred to as Z). For example, in document retrieval $X = \{\text{Term}\}$, $Y = \{\text{DocId}\}$ and $Z = \text{docIdx}$. Figure 3 presents the generalised structure of an SQL query.

```
SELECT DISTINCT Y                                  -- Y: {target attributes }
  FROM Z                                           -- Z: {sources}
  WHERE Z.X1='value1' OR Z.X1='value2' ...; -- X: {conditioned attr.}
```

Figure 3: Generalised Structure of an SQL Query

We now formalise the SQL2PSQL translation. The first algorithm decomposes the SQL expression and generates probabilistic

views (spaces) that are later combined according to the ranking rationales of retrieval models, such as TF-IDF and LM. With respect to the generalisation of document retrieval to tuple retrieval, the location frequency corresponds to the tuple frequency, and the document frequency corresponds to the value frequency.

ALGORITHM 1. *SQL2PSQL: Basic Views*

For each conditioned attribute in X in SQL query "SELECT DISTINCT Y FROM Z WHERE X" create the following views:

1. *Tuple-based (Location-based) probabilities: p_Z_X. Probability $P_Z(X)$ is the tuple-based probability of X. View naming example: p_docIdx_Term.*

```
-- Tuple retrieval                  -- Document retrieval
CREATE VIEW p_Z_X AS                -- p_docIdx_Term
 SELECT SUM X                       -- X=Term
 FROM Z | ();                       -- Z=docIdx
-- Prob semantics: P_Z(X)           -- P_docIdx(Term)
```

2. *Value-based (Document-based) probabilities: p_Y_Z_X. Probability $P_{Z[Y]}(X)$ is the value-based probability of X. View naming example: p_DocId_docIdx_Term.*

```
-- Tuple retrieval                  -- Document retrieval
CREATE VIEW p_Y_Z_X AS              -- p_DocId_docIdx_Term
 SELECT X                           -- X={Term}
 FROM Z                             -- Z=docIdx
 ASSUMPTION VALUE_FREQ              -- DocId_Freq
 EVIDENCE KEY ();
 -- Prob semantics: P_Z[Y](X)       -- P_docIdx[DocId](Term)
```

3. *Conditional probabilities in sources: p_Z_X_Y. Probability $P_Z(X|Y)$ is the conditional probability of X. View naming example: p_docIdx_Term_DocId.*

```
-- Tuple retrieval                  -- Document retrieval
CREATE VIEW p_Z_X_Y AS              -- p_docIdx_Term_DocId
 SELECT SUM X, Y                    -- X={Term} Y={DocId}
 FROM Z | (Y);                      -- Z=docIdx
 -- Prob semantics: P_Z(X|Y)        -- P_docIdx_Term_DocId
```

4. *Information-based probabilities: p_inv_Y_Z_X. Probability $P_Z(X \text{ informs})$ is the probability of X is informative. View naming example: p_inv_DocId_docIdx_Term.*

```
-- Tuple retrieval                  -- Document retrieval
CREATE VIEW p_inv_Y_Z_X AS          -- p_inv_DocId_docIdx_Term
 SELECT X FROM Z                    -- X={Term} Y={DocId} Z=docIdx
 ASSUMPTION MAX_IDF                 -- idf(t)/max_idf
 EVIDENCE KEY ();
```

The SQL2PSQL algorithm generates probabilistic views independent of a particular retrieval model. The algorithms illustrated in the following sections show how and which of these views are used to construct retrieval models such as TF-IDF, LM and BM25.

3.1 TF-IDF-based Processing of SQL Queries

Algorithm 2 shows how a TF-IDF PSQL program is automatically generated for the genuine SQL query in Figure 3.

ALGORITHM 2. *SQL2PSQL(TF-IDF)*

Step 1: Create for each conditioned attribute X in SQL query "SELECT DISTINCT Y FROM Z WHERE cond(X)" one view to reflect tf and another one to reflect idf.

```
-- PSQL views generated for TF-IDF-based Retrieval

-- Tuple retrieval                  -- Document retrieval
CREATE VIEW Z_X_Y_tf                -- docIdx_Term_DocId_tf
 SELECT X, Y FROM p_Z_X_Y;          -- p_docIdx_Term_DocId

CREATE VIEW pidf_Z_X                -- pidf_docIdx_Term
 SELECT X FROM p_inv_Y_Z_X;         -- p_inv_DocId_docIdx_Term
```

Step 2: In the genuine query, replace each source (Z) by the tf (variable D) and idf (variable Q) relations of the specified attributes. Replace each ordinary *condition Z.X='value' by Q.X='value' AND Q.X=D.X. Replace each* join *condition Z1.X=Z2.X by D1.X=D2.X AND Q1.X=D1.X AND Q2.X=D2.X.*

For a genuine query SELECT Y FROM Z WHERE Z.X='value1' OR Z.X='value2', the output is:

```
SELECT SUM D.Y                      -- Y={DocId}
 FROM pidf_Z_X Q, Z_X_Y_tf D        -- Z=docIdx, X={Term}
 WHERE Q.X='value1' AND Q.X=D.X
 OR    Q.X='value2' AND Q.X=D.X;
```

To improve readability, we omit the output for join conditions. The algorithm generates a PSQL program for TF-IDF-based retrieval.

THEOREM 1. *Algorithm SQL2PSQL(TF-IDF) is correct.*

PROOF. The view Z_X_Y_tf (for document retrieval, p_docIdx_Term_DocId, i.e. the "TF" component) has the probabilistic semantics $P_Z(X|Y)$ ($P_{\text{docIdx}}(\text{Term}|\text{DocId})$). The view pidf_Z_X (for document retrieval, pidf_docIdx_Term) has the semantics $P(\text{X is informative}) = \text{idf}(X)/\text{maxidf}(c)$. The probabilities in the SQL query "SELECT SUM D.Y" correspond to $\sum_{t \in d \cap q} \text{TF}(t,d) \cdot \text{IDF}(t,c)/\text{maxidf}(c)$. □

3.2 LM-based Processing of SQL Queries

Algorithm 3 shows how an LM PSQL program is automatically generated for the genuine SQL query in Figure 3.

ALGORITHM 3. *SQL2PSQL(LM)*

Step 1: Create for each conditioned attribute X in SQL query "SELECT DISTINCT Y FROM Z WHERE cond(X)" one view for the foreground *and one view for the* background *model. These correspond to the document model and the collection model, respectively. The foreground and background models are weighted with the respective mixture parameters. Next, the foreground is united with the background model. This yields the LM mixture in view "Z_X_mix".*

```
-- PSQL views generated for LM-based Retrieval

-- Tuple retrieval                  -- Document retrieval
-- foreground (fg) model            -- document model: P(t|d)
CREATE VIEW Z_X_Y_fg AS             -- docIdx_Term_DocId_fg
 SELECT X, Y                        -- X={Term}, Y={DocId}
 FROM p_Z_X_Y;                      -- p_docIdx_Term_DocId

-- background (bg) model            -- collection model: P(t|c)
CREATE VIEW Z_X_bg AS               -- docIdx_Term_bg
 SELECT X                           -- X={Term}
 FROM p_Z_X;                        -- p_docIdx_Term

 -- To conserve space, the automatic generation of weighted
 -- foreground model and background model is omitted

 -- lambda_Z_X_fg
 -- lambda_Z_X_bg

CREATE VIEW Z_X_mix AS              -- docIdx_Term_mix
 lambda_Z_X_fg UNION DISJOINT       -- lambda_docIdx_Term_fg
 lambda_Z_X_bg;                     -- lambda_docIdx_Term_bg
```

Step 2: In the genuine SQL query, replace each source (Z) by Z_X_mix. For a genuine query SELECT Y FROM Z WHERE Z.X='value1' OR Z.X='value2', the output is:

```
SELECT PROD D.Y
 FROM Z_X_mix D
 WHERE D.X = 'value1' OR D.X = 'value2' ...
```

THEOREM 2. *Algorithm SQL2PSQL(LM) is correct.*

PROOF. The probabilistic views Z_X_Y_fg ($P(t|d)$) and Z_X_bg ($P(t|c)$) correspond to the docModel and collModel views in the reference implementation. The view Z_X_mix has the semantics $P(t|d,c) = \lambda \cdot P(t|d) + (1-\lambda) \cdot P(t|c)$. The probabilities in the SQL query "SELECT PROD D.Y" correspond to $P(q|d,c) = \prod_{t \in q} P(t|d,c)$. □

This concludes the ranking-based processing of SQL queries. The SQL2PSQL translation has been given for a simple SQL query (one source relation, one specified attribute, one target attribute). The translation, however, applies to multiple sources, ordinary and joined conditions and several attributes in the target list, all of which have been demonstrated in the algorithms and proofs.

4. EVALUATION

The purpose of the evaluation is to demonstrate the feasibility of the proposed technology, namely the PSQL framework and the SQL2PSQL functionality.

Evaluating such a framework that primarily argues for abstraction and openness, in our point of view, requires benchmarks that measure the capability of a technology to satisfy both the needs of structured search and the broader needs that arise because of task complexity, customisation, and change management. Indeed, the ability to calculate the *costs* for improving the retrieval quality would be ideal for judging the benefits of the proposed technology. Such productivity-oriented (change management) benchmarks and evaluations are part of future research. For the scope of this paper, we instead present a retrieval quality vs. retrieval time evaluation.

4.1 Set-up

Table 1 outlines the statistics of the two benchmarks which the evaluation employs.

	Size MB	docs $N_D(c)$	topics $N_Q(c)$
IMDB	1,800	437,281	40
Monster	2,005	1,040,000	60

Table 1: Benchmarks Statistics

The experiments were run on a single CPU using a DB+IR prototype [22] that supports the retrieval of text, structured and semi-structured data. The prototype provides an open-box framework with high-level and customisable concepts to represent/model data and PSQL to implement ranking models.

To demonstrate how the modelling framework performs, we evaluate the generated retrieval models, TF-IDF and LM (settings in Section 2.2), with respect to a restricted evidence size. In standard IR this corresponds to processing the most selective terms first and choosing the top-k document Ids from the posting list. Similar processing techniques were used herein.

We present how the retrieval quality and time increase proportionally with an increase in evidence size. The aim is to investigate the implementation of the retrieval models by examining how much quality could be achieved and at what cost. This cost-based evaluation is with respect to the retrieval model itself and how favourably or poorly it utilises the underlying evidence.

One achievement of this work is to support nested expressions whereby the required efficiency can be achieved via top-k and evidence-based processing. Nowadays, relational approaches to IR often rely on materialisation and are based on stepwise processing pipelines. The views used to implement the models, however, have *not* been materialised, and, hence, the cost includes the time to generate and process them.

We report the retrieval quality of the generated programs and the average retrieval time (total retrieval time divided by the total number of queries) of the generated programs against the number of evidence tuples. The number of considered evidence tuples ranges from 10,000 to 200,000. We use Mean Average Precision (MAP) and Reciprocal Rank as the evaluation metrics.

4.1.1 Baselines

The baselines are based on the document-oriented TF-IDF and LM retrieval models with no top-k processing. In other words, the structure of the data is not taken into consideration by the retrieval model and a bag-of-words representation is utilised (this is similar to the experimental methodology reported in [2]). In the case of tuple retrieval, however, the document structure (see Section 4.2 for the list of element types) is considered when constructing the SQL queries. Note that the purpose of the evaluation is primarily to present the *feasibility* of the modelling framework and translation algorithms. Other retrieval models, such as [14], can be generated by the SQL2PSQL algorithms, specifically Algorithms 1 and 3.

4.1.2 Query Formulation

For both benchmarks, the keyword queries and the *ideal* mapped attribute for each query term are used. This ideal attribute is extracted from the set of relevant documents for each query.

The intuition behind the mapping is that if a term occurs frequently within a certain element type then the term is more likely to be "characterised" by that particular type [14]. This mapping method offers a convenient way to automatically transfer a keyword-based query to a structured one. The correctness of the extracted attribute is verified manually. After the SQL query is formulated it is automatically augmented with the desired ranking strategy which results in a relevance-based processing of that query. Below is an example of an SQL-based formulation of IMDB query number 28 "*gladiator action maximus scott*".

```
# gladiator action maximus scott
SELECT Movie
 FROM movies WHERE Title = 'gladiator' AND Genre = 'action'
 AND Plot = 'maximus' AND Director = 'scott';
```

As noted above, when creating the SQL queries for the IMDB and Monster benchmarks, only the top-mapped attribute for each query term is considered. For example, the top-mapped attribute for query term "gladiator" is "Title". This is particularly important when evaluating a larger and noisier dataset such as Monster because the significant term overlap among the different elements makes estimating the correct attributes a challenge. In practice, the entire dataset, and not just the relevant documents, is used to deduce the mappings. Additionally, experiments [14] have shown that improving the accuracy of the mappings and considering more mappings per query term can improve retrieval performance. However, the focus of our experimental study is not on the mapping process, and the SQL2PSQL facility is flexible enough to process queries with multiple attributes in the target list and any other customisations that adapt a retrieval model to a particular application.

4.2 IMDB

The evaluation results in the previous section indicate a good retrieval quality below the one second ceiling on an IR-specific text dataset. We now extend the evaluation of the translation methods

and ranking functions to more semantic/structured benchmark. The IMDB dataset is formatted in XML and was constructed from text data (http://www.imdb.com/interfaces#plain). Each document corresponds to a movie. The element types are "title", "year", "releasedata", "language", "genre", "country", "location", "colorinfo", "cast", "team" and "plot".

	MAP	RecipRank		Time (s)
SQL2PSQL(TF-IDF:10k)	44.01	45.42		00.24
SQL2PSQL(TF-IDF:20k)	46.09	47.42		00.59
SQL2PSQL(TF-IDF:50k)	48.80	48.58		00.70
SQL2PSQL(TF-IDF:100k)	49.96	49.27		00.87
SQL2PSQL(TF-IDF:200k)	**54.37**†	(+13.06)	**55.64**† (+13.92)	01.28
Baseline: TF-IDF:All	48.09	48.84		
SQL2PSQL(LM:10k)	27.75	29.19		00.19
SQL2PSQL(LM:20k)	31.30	32.19		00.50
SQL2PSQL(LM:50k)	42.70	43.89		00.75
SQL2PSQL(LM:100k)	*43.90*†	(+09.37)	*44.43*† (+08.63)	01.05
SQL2PSQL(LM:200k)	41.85	42.47		01.27
Baseline: LM:All	40.13	40.90		

Table 2: Quality w.r.t number of evidence tuples considered; the best overall result is in bold, and the best result out of the baseline and its tuple-based variant is italicised; results that are statistically significant ($p < 0.05$) above the baseline are denoted by †, as determined by a signed t-test.

There are several processing steps that present the data in the form that is best situated for achieving effective retrieval. For example, we have opted to propagate the keywords occurring within elements such as "actor" and "director" upwards to their corresponding part. Having a coarser schema helps to improve the accuracy of the derived attributes for each query term. Another example is that the terms that occur in a specific context have also been propagated upwards to the root element. This propagation helps to model document-based retrieval as opposed to element-based one.

[14] provided the test-bed which included 40 queries. These queries were created assuming a situation in which a user wants to find a movie using partial information spanning over many elements. Relevant documents were found manually.

As discussed in Section 3, the SQL queries are decomposed into two parts: an indexing part and a retrieval part. The indexing part is done using the representation of the IMDB benchmark. In the retrieval part, for each query the SQL2PSQL algorithm (Algorithm 1) generates a set of probabilistic views independently of a particular retrieval model. These views are then used to construct the desired retrieval model. For the evaluation, the generated relevance-based ranking of the SQL queries is based on SQL2PSQL(TF-IDF) and SQL2PSQL(LM) algorithms (Algorithm 2 and Algorithm 3).

In Table 2 the values in parenthesis represent the relative (percentage) difference in performance from the baseline results. If evidence beyond the 200,000 tuples for the generated TF-IDF program is considered, retrieval quality continues to monotonically increase. However, considering further evidence tuples for the generated LM program (optimal performance $\lambda \approx 0.7$) does not significantly increase retrieval quality. In fact, after a certain amount of evidence is considered, the quality starts to decrease. As previously discussed, the degradation in performance is due to the top-k operators, which are optimised for TF-IDF-based models.

4.3 Monster

The benchmark contains longer documents with more complex structure and full text content than the IMDB. Moreover, the queries, which are requests for jobs created by real users of the Monster service, are longer and more complex. The data and queries were licensed from "http://www.monster.com". An example of a keyword-based query is "*aircraft airplane mechanic technician tech fort riley kansas*". This query seeks an Aircraft Structural Mechanic to determine the methods to repair malfunctioning or damaged components of aircrafts in Fort Riley, Kansas.

	MAP	RecipRank		Time (s)
SQL2PSQL(TF-IDF:10k)	22.00	39.17		00.41
SQL2PSQL(TF-IDF:20k)	23.89	42.51		00.98
SQL2PSQL(TF-IDF:50k)	24.79	46.94		01.13
SQL2PSQL(TF-IDF:100k)	27.04	51.84		01.32
SQL2PSQL(TF-IDF:200k)	**29.22**†	(+11.34)	**54.45**† (+13.39)	01.85
Baseline: TF-IDF:All	26.23	48.02		
SQL2PSQL(LM:10k)	17.10	22.25		00.42
SQL2PSQL(LM:20k)	18.04	26.51		00.78
SQL2PSQL(LM:50k)	20.91	30.94		00.91
SQL2PSQL(LM:100k)	*22.84*†	(+08.09)	*32.84*† (+08.24)	01.31
SQL2PSQL(LM:200k)	21.02	31.45		01.78
Baseline: LM:All	21.13	30.34		

Table 3: Quality w.r.t number of evidence tuples considered; the best overall result is in bold, and the best result out of the baseline and its tuple-based variant is italicised.

Similar to when representing the IMDB benchmark we have adopted a coarse schema which helps to improve the accuracy of the derived SQL queries. The selected elements in the schema include "resumetitle", "summary", "desiredjobtitle" and "schoolrecord". The terms occurring in a specific context have been propagated upwards to the root element. The SQL queries and the baseline were constructed similarly to those in the IMDB benchmark.

Table 3 illustrates the quality results of the TF-IDF and LM generated programs. As observed for the IMDB, considering evidence tuples beyond 200,000 does not significantly increase retrieval quality. The average retrieval times demonstrate the processing cost for the ranking-based processing of the SQL queries. The retrieval quality is reasonable at the one second ceiling, but further precision requires longer retrieval times. Although the retrieval times appear to be scalable, the processing costs can present a further challenge if lager text-based or semi-structured datasets are used. One solution is proposed by [13], which demonstrates how the probabilistic relational/descriptive approach employed herein can be parallelised to achieve scalable processing.

5. SUMMARY & CONCLUSION

[22] demonstrated that PSQL provides sufficient expressiveness for modelling document retrieval models. This paper advances the feasibility of using IR models for processing SQL queries and develops the application of PSQL for tuple retrieval. The SQL2PSQL translation algorithms were shown to generate PSQL programs that assign an IR-model-based, probabilistic ranking to the tuples retrieved for an SQL query.

The evaluation showed that the PSQL framework and the SQL2PSQL facility deliver good retrieval performance. This outcome is a direct consequence of the IR models. The main contributions of SQL2PSQL are customisable rankings and the expressiveness to model tuple retrieval while incorporating keyword-based document retrieval models. A quality-oriented evaluation is required. Ideally, however, this evaluation should measure how DB+IR technology improves *productivity*, and specifically, how it adapts to customer-tailored and changing relevance criteria. The SQL2PSQL architecture delivers this flexibility, and developing such an evaluation measure is part of future research.

The ranking-aware processing of SQL queries is particularly relevant at a time when SQL queries, and more generally, structured queries, are increasingly being generated by applications built on top of search engines. The SQL2PSQL facility is also applicable to

other structured query languages such as SPARQL. For example, an SPARQL2SQL translation, which acts as a query processing front-end, or a more direct one, such as SPARQL2PSQL, can be implemented. This re-use of the proposed SQL2PSQL technology can lead to general-purpose DB+IR solutions.

The automatic generation of PSQL from SQL has a twofold impact. The generation of "seamless" DB+IR technology becomes more accessible since applications/users are not required to learn PSQL to achieve customisable and re-usable retrieval strategies. Moreover, well-defined rankings are available as modules ready to be tailored to specific search tasks and to other structured query languages. In summary, the presented DB+IR approach supports the use of SQL and extends the well-defined IR models for the effective ranking of documents *and* tuples.

6. ACKNOWLEDGMENTS

We are grateful for the support of Yahoo! Labs Barcelona. We would also like to thank Prof. Bruce Croft for providing us with the two benchmarks, and the reviewers for their excellent suggestions.

7. REFERENCES

[1] D. Barbara, H. Garcia-Molina, and D. Porter. A probabilistic relational data model. In *Proc. of EDBT*, 1990.

[2] M. Bilotti, P. Ogilvie, J. Callan, and E. Nyberg. Structured retrieval for question answering. In *Proc. of SIGIR*, 2007.

[3] P. Bosc, M. Galibourg, and G. Hamon. Fuzzy querying with sql: extensions and implementation aspects. *Fuzzy Sets Syst.*, 28(3):333–349, 1988.

[4] J. Callan. Search engine support for software applications. In *Proc. of CIKM*, 2010.

[5] A. Chandel, O. Hassanzadeh, N. Koudas, M. Sadoghi, and D. Srivastava. Benchmarking declarative approximate selection predicates. In *Proc. of SIGMOD*, 2007.

[6] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic IR approach for ranking of DBs query results. *ACM TODS*, 31(3):1134–1168, 2006.

[7] K. Church and W. Gale. Inverse document frequency (idf): A measure of deviation from Poisson. In *Proc. of the Workshop on Very Large Corpora*, 1995.

[8] R. Cornacchia and A. de Vries. A Parameterised Search System. In *Proc. of ECIR*, 2007.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic DBs. In *Proc. of VLDB*, 2004.

[10] N. Fuhr and T. Roelleke. A probabilistic relational algebra for the integration of IR and DB systems. *ACM TOIS*, 14(1):32–66, 1997.

[11] D. Hawking. Challenges in enterprise search. In *Proc. of ADC*, 2004.

[12] D. Hiemstra and V. Mihajlovic. A DB approach to IR: The remarkable relationship between language models and region models. Technical Report arXiv:1005.4752, 2010.

[13] I. Klampanos, H. Azzam, T. Roelleke: A case for probabilistic logic for scalable patent retrieval. In *Proc. of the workshop on Patent information retrieval*, 2009.

[14] J. Kim, X. Xue, and W. Croft. A probabilistic retrieval model for semistructured data. In *Proc. of ECIR*, 2009.

[15] C. Li, K. Chen-Chuan Chang, I. Ilyas, and S. Song. RankSQL: Query algebra and optimization for relational Top-k queries. In *Proc. SIGMOD*, 2005.

[16] P. Mika, E. Meij, and H. Zaragoza. Investigating the semantic gap through query log analysis. In *Proc. of ISWC*, 2009.

[17] A. Motro. Vague: A user interface to relational DBs that permits vague queries. *ACM Trans. on Office Information Systems*, 6(3):187–214, 1988.

[18] J. Ponte and W. Croft. A language modeling approach to IR. In *Proc. of SIGIR*, 1998.

[19] C. Ré and D. Suciu. Materialized views in probabilistic DBs: for information exchange and query optimization. In *Proc. of VLDB*, 2007.

[20] S. Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:503–520, 2004.

[21] S. Robertson, S. Walker, and M. Hancock-Beaulieu. Large test collection experiments on an operational interactive system: Okapi at TREC. *IP&M*, 31:345–360, 1995.

[22] T. Roelleke, H. Wu, J. Wang, and H. Azzam. Modelling retrieval models in a probabilistic relational algebra with a new operator: The relational Bayes. *VLDB Journal*, 17(1):5–37, January 2008.

[23] S. Sakr and G. Al-Naymat. Relational processing of rdf queries: a survey. *SIGMOD Rec.*, 38(4):23–28, 2009.

[24] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In *Proc. of VLDB*, 2005.