The VLDB Journal (2008) 17:5–37
DOI 10.1007/s00778-007-0073-y

SPECIAL ISSUE PAPER

Modelling retrieval models in a probabilistic relational algebra with a new operator: the relational Bayes

Thomas Roelleke · Hengzhi Wu · Jun Wang · Hany Azzam

Received: 18 September 2006 / Revised: 14 April 2007 / Accepted: 11 June 2007 / Published online: 12 October 2007
© Springer-Verlag 2007

Abstract This paper presents a probabilistic relational modelling (implementation) of the major probabilistic retrieval models. Such a high-level implementation is useful since it supports the ranking of any object, it allows for the reasoning across structured and unstructured data, and it gives the software (knowledge) engineer control over ranking and thus supports customisation. The contributions of this paper include the specification of probabilistic SQL (PSQL) and probabilistic relational algebra (PRA), a new relational operator for probability estimation (the relational Bayes), the probabilistic relational modelling of retrieval models, a comparison of modelling retrieval with traditional SQL versus modelling retrieval with PSQL, and a comparison of the performance of probability estimation with traditional SQL versus PSQL. The main findings are that the PSQL/PRA paradigm allows for the description of advanced retrieval models, is suitable for solving large-scale retrieval tasks, and outperforms traditional SQL in terms of abstraction and performance regarding probability estimation.

Keywords Probabilistic relational modelling · Retrieval models · Probabilistic databases · DB + IR integration

T. Roelleke (✉) · H. Wu · J. Wang · H. Azzam
Queen Mary, University of London, Mile End Road,
London E1 4NS, UK
e-mail: thor@dcs.qmul.ac.uk

H. Wu
e-mail: hzwoo@dcs.qmul.ac.uk

J. Wang
e-mail: wangjun@dcs.qmul.ac.uk

H. Azzam
e-mail: hany@dcs.qmul.ac.uk

1 Introduction

The call for a VLDB special issue on integration of Databases (DB) and Information Retrieval (IR) itself is probably the best evidence of a new era of DB + IR technology. What is triggering DB + IR? The call mentions the usual suspects such as XML, the web, and huge amounts of data. Maybe the integration of structured and unstructured data sources is what drives integrated DB + IR? However, as the call says, early DB + IR attempts date back to early 1970s.

Despite a similar overall aim, namely to process queries and retrieve results, the fields of DB and IR research developed differently and in separate communities. DB focused on expressiveness, structure (data records), and data models, whereas IR focused on free-text query languages, unstructured data, and the inverted list as the ultimate "data model" for large document collections. Whereas in DB, software engineering and productivity issues have always been important, these are of secondary priority in IR research: an improvement in retrieval quality is good, whatever the approach and effort in person-years needed to achieve the improvement. Whereas DB usually targets people who build systems or business applications, and therefore, DB had to provide a useful and, overall, re-usable and generic technology, IR focused mainly on experimental evaluation of retrieval quality and end-user applications.

We could view the DB + IR efforts as a DB-Technology + IR-Service integration. The technology is strong in flexibility, robustness, abstraction, and the service is strong in ranking and presenting retrieved objects, i.e., documents and facts.

To conclude why DB + IR, and why now, we believe that the growing need for customisable ("tunable") search services triggers the demand for DB + IR. For building efficiently effective search systems, IR approaches need to be available in DB technology, and, the other way round, DB

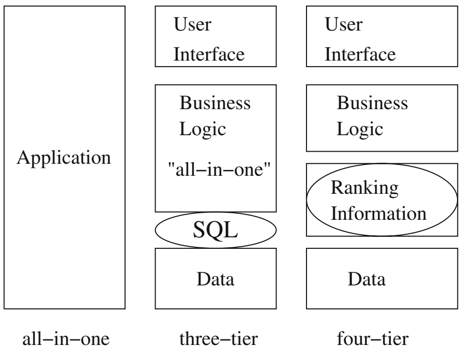

Fig. 1 Data and information independence

External layer	Information sorted by relevance
Logical layer	Probabilistic relations Probability estimation: Bayes[]()
Physical layer	Relational model/algebra; SQL

Fig. 2 External, logical and physical layers

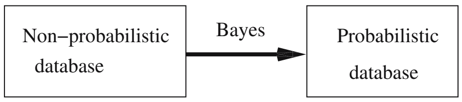

Fig. 3 Creation of probabilistic databases

technology needs to be ready to host IR methods such as relevance-based ranking, result browsing, and vague predicates.

Figure 1 highlights the current trend along a time-line from all-in-one applications in the 1970s/1980s to three-tier architectures becoming the standard in the late 1980s. Certainly, more tiers can be identified in today's IT systems, depending on the emphasis. Certainly, business logics have a complex structure, as the underlying data management systems have.

In the classical three-tier architecture, information management (search logic, relevance-based ranking) is maybe best located at the interface between business logic and DB system, where nowadays SQL has its dominating role. In the four-tier architecture, we leave it intentionally open whether SQL still plays its role as it used to, or whether it plays a new role. It will play a role, but the evaluation of SQL might be richer, more effective, in the sense that only the important tuples are returned to the business logic, the tuples will be sorted by relevance, and for this, the business logic needs to be able to specify what it means by "relevance".

As we will demonstrate in this paper, traditional SQL is already suitable to fulfill such task, like an assembler language is capable of developing an accounting program with a web interface. Though capable, for obvious reasons, the application programmer prefers a higher level and more tailored language than assembler. This highlights the motivation for our research: we are working on data models and SQL variants that are tailored to information management tasks.

To position the probabilistic SQL (PSQL) and the probabilistic relational algebra (PRA) presented in this paper, consider Fig. 2. There is an external layer where information can be accessed, sorted by relevance. Then, there is a logical layer in which the information space and retrieval strategies (ranking functions) are modelled. Finally, there is a physical layer, which is here the relational model/paradigm. Although the traditional relational paradigm has all the expressiveness needed (like an assembler language has all the expressiveness needed), to improve the productivity in developing information management/search applications, we add the logical and the external layers.

This paper deals with the logical layer, and more precisely, the paradigm of probabilistic relations, probability aggregation and estimation, and the probabilistic relational modelling of retrieval models. Probabilistic database models are well established, and we will review prior research in the next section. Our contribution is the formalisation and high-level implementation of retrieval models. This supports the customisation of search strategies. For implementing retrieval models solely by means of a probabilistic relational model, we required and developed a new operator, the relational Bayes. With the Bayes operator, probability estimation is now *within* the probabilistic relational paradigm.

This contribution is highlighted in Fig. 3. The relational Bayes allows to describe the estimation/generation of probabilities from a non-probabilistic database—an important functionality for generating probabilities in a coherent and comprehensive probabilistic relational algebra.

1.1 Outline of this paper

The outline of the paper is as follows:

The sections can be grouped and described as follows:

Part 1: This general part contains background, running example, and requirements.

Part 2: This technical part introduces PSQL (probabilistic SQL) and PRA (probabilistic relational algebra). Section 5 deals with probability *aggregation*; this can be considered as a review of state-of-the-art PRA. Section 6 adds probability *estimation* (the relational Bayes) to the probabilistic algebra.

Part 3: Section 7 shows the modelling of retrieval models in PSQL and PRA.

Part 4: Section 8 evaluates PSQL/PRA. For this, we compare PSQL with traditional SQL, and demonstrate the gain in abstraction, while showing that there are efficiency gains as well.

Part 5: Sections 9 and 10 conclude the paper.

2 Background

2.1 IR with SQL

Over the past two decades attention has grown towards the integration of IR ranking techniques into SQL. In fact, it has been labelled as one of the major challenges facing the community nowadays [1]. While one of the earliest efforts to address this integration date back to [60] and [14], more recent work can be found in [32] where the classical relational model is used to achieve basic integration of structured data and text. Specifically, the Boolean retrieval model is implemented using standard SQL. Other systems that implemented the same model are DBXplorer [2] and DISCOVER [35]. Both retrieval systems are built on two different commercial databases, but rely on a similar architecture to support keyword search.

Unlike [2] and [35], [33] is not just capable of supporting Boolean-AND semantics, but also Boolean-OR semantics. However, more effective retrieval models, like tf-idf (term frequency–inverse document frequency) can be implemented. For example, [28,29] introduce the PowerDB-IR system, which is an IR system built on top of a database cluster. It implements the tf-idf-based model by mapping it to SQL. Furthermore, [3] attempts to apply IR models on database to resolve the “Empty Answer” problem by extending the IR-based tf-idf concepts and developing an idf similarity for database ranking.

Conversely, [11,12] attempt to solve the “Many-Answers” problem by using probabilistic ranking of query results, which is another approach for ranking in databases. This approach will be further discussed in the next subsection. Li et al. [36] introduces RankSQL, which is a RDBMS that fully integrates ranking support as a first-class functionality. A framework is introduced to support efficient evaluations of top-k by extending relational algebra and query optimisation. This approach is different from the typical DB and IR approach because it does not focus on how to rank tuples (apply IR models), but focuses on optimising the returned ranked-list of the results.

A re-innovated look at the integration of structured data and text can be found in [15], which provides a deeper understanding of the requirements and possible system architectures to achieve such an integration. Moreover, the importance of the probabilistic approach for DB + IR integration is emphasised in this work.

Finally, a benchmark for the evaluation of traditional approaches for the integration of IR with SQL has been introduced in [19]. The benchmark, called the TEXTURE Benchmark, introduces queries with relevance ranking, like text-only queries, single-relation mixed queries and multiple-relation mixed queries, rather than those that just compute all answers. Most importantly, the queries are formulated using the “CONTAINS” operator. This operator enables the seamless integration of text and relational processing with top-k ranking. In addition to presenting the benchmark, the performance numbers for three commercial DBMSs and their support for text in current relational database systems are analysed.

2.2 Probabilistic databases

The research efforts towards the integration of DB and IR has led to important findings in the area of probabilistic database technology. It was quickly realised that probabilistic data models are essential for this integration because of their capability of introducing more efficient retrieval models in DB. Consequently, it became possible to measure imprecision in large-scale data, return top-k results subset instead of the whole set of results and make use of early response algorithms possible. Accordingly, probabilistic databases conveyed a message that the overall quality of data has been improved, and the processing time for queries has drastically decreased by returning top-k results subset.

Early work extending relational and object oriented data models using the fuzzy set and the possibility theory was [6]. The notion of quality of databases and its estimation using a probabilistic approach was discussed in [44], where the relational model of data was extended and a quality specification with each relation instance was associated.

Another crucial aspect of probabilistic databases is related to efficient query evaluation. This aspect has been discussed in [16], where a system is discussed that supports arbitrarily complex SQL queries on probabilistic databases and provides an optimisation algorithm that can efficiently compute most queries. In addition, a tutorial, Foundations of Probabilistic Answers to Queries given by Suciu and Dalvi at SIGMOD’05 illustrates a set of probabilistic query answering techniques that underlie several recent database applications [17]. One of these applications is discussed in [4] where using probabilistic databases coupled with relaxed query expressions is suggested as a promising solution for efficient retrieval of large-scale semantic data.

Finally, below is an example adapted from [58], which demonstrates the usage of probabilities in databases. Consider the following table of person data where for the persons Miklau and Bala, we are uncertain about their affiliation and state:

Student			
Name	Affiliation	State	Area
Miklau	UW	WA	Data Security
Miklau	Umass	MA	Data Security
Dalvi	UW	WA	Prob. Data
Bala	UW	WA	Data Streams
Bala	MIT	MA	Data Streams
Bala	Umass	MA	Data Streams

If {Name} is viewed to be the key, then the key condition is violated because of the multiple occurrences (inconsistency) of tuples with the same key.

For the query

```
SELECT * FROM Student WHERE State = 'WA';
```

we could either retrieve the consistent tuples only, or be softer (ready to accept false hits), and retrieve inconsistent tuples as well.

For quantifying the inconsistency in a probabilistic way, an intuitive approach is to assign probabilities based on the number of inconsistent tuples:

Student				
Prob	Name	Affiliation	State	Area
0.5	Miklau	UW	WA	Data Security
0.5	Miklau	Umass	MA	Data Security
1	Dalvi	UW	WA	Prob. Data
0.33	Bala	UW	WA	Data Streams
0.33	Bala	MIT	MA	Data Streams
0.33	Bala	Umass	MA	Data Streams

As we will point out in Sect. 6 on probabilistic SQL, the approach mentioned above is one way of estimating probabilities. We also define in this paper the notion of evidence key: here, {Name} forms the evidence key, and the tuple probabilities are conditional probabilities of the form $P(\tau|\text{Name})$.

We have reviewed a number of approaches dealing with probabilistic relations. One of the contributions of this paper is to define and evaluate a probabilistic SQL technology for large-scale probabilistic databases.

2.3 On probabilistic relational algebra and probability estimation

The relational algebra, the processing basis of SQL, is one of the pillars of database technology. However, from an IR and uncertainty management point of view, the relational algebra lacks relevance-based ranking of retrieved objects. Therefore, many probabilistic extensions for the relational algebra have been defined: see [14] on probabilistic databases, [10,26,44,45] on vague queries (fuzzy predicates), [7] and [8] on probabilistic relational modelling, [37] on probability aggregation, [42] on text retrieval and the relational model, [24] on a PRA for the integration of database and information retrieval, [46] and [23] on NF2 relations, [27] on probabilistic Datalog, [38] on the ProbView system, [30] on text retrieval with SQL, and [54] on probabilistic aggregates.

One may wonder why did so many researchers looked at the problem of adding probabilities to the relational databases?

This is due to the fact that probabilistic relational algebra (PRA) is a powerful candidate for modelling intrinsic uncertainty of knowledge. Eventually, this can be used for modelling an estimate of the relevance of retrieved objects. However, most of the aforementioned models share at least two short-comings. The initial probability estimation is modelled "outside" of the algebra, and, the "how" of the aggregation of uncertainty values is specified. The "outside" nature of the probability estimation was viewed by designers and developers, who used PRA, Datalog and other languages, as a shortcoming. Also, if a model allows to specify the "how" of the aggregation of uncertainty values, then we model on a physical (assembler-like) layer rather than on a logical layer.

Previous works ([7], [24] and [37]) define variants of PRA where the focus is on the definition of the probability *aggregation* for the five basic relational operators (selection, projection, join, union, subtraction); with the Bayes operator, this paper adds probability *estimation*. Probability estimation is with Bayes "inside" PRA. Bayes provides ways to specify the "what" of frequency-based and information-theoretic probability estimation. The "how" is controlled in the physical layer of PRA.

An important aspect of a PRA is highlighted by [54]: attribute value aggregation (sum, average, maximum) is *orthogonal* to probability aggregation! This stresses again that the aggregation of uncertainty values should *not* be implemented in a logical layer of a PRA.

Another aspect of PRA is the discussion of 1NF versus NF2 (non-first-normal-form) nature of probabilistic relations. This discussion is closely related to the discussion whether a probability is assigned to a tuple, or whether a probability is assigned to an attribute value (see [23], [46], and [54]). NF2 relations are expensive in processing and the experience with the 1NF PRA proved that NF2 modelling is not a pre-requisite for effective usage of a PRA model.

2.4 Retrieval models

Retrieval models form a crucial part of information retrieval. We mainly distinguish between two classes: non-probabilistic and probabilistic models. On the non-probabilistic side, tf-idf is the dominant model, and on the probabilistic side, the binary independent retrieval model and language modelling are the main candidates. Probabilistic models come with a theory and some heuristics, whereas non-probabilistic models are mainly based on heuristics.

Probabilistic models date back to [43]. Probabilistic models try to estimate the probability of a document being judged relevant to a particular query. This is denoted as the probability of relevance $P(r|d,q)$. Because there is no direct quantitative method to estimate the relevance probability, there are various methods to estimate the relevance probability. In the late 1970s, [55] established the binary independent retrieval model (BIRM). From the middle to end 1980s, [63] initiated approaches to model IR as the probability $P(d \rightarrow q)$ of a non-classical implication between documents and queries. Early 1990s brought the inference network model [61], middle 1990s contributed the $P(d \rightarrow q)$ framework [64], and late 1990s to early 2000 brought language modelling ([9,41,47,66]) and divergence from randomness ([5]).

In probabilistic information retrieval models, an important aspect is how to estimate the term weight, possibly related to the probability of relevance. Without relevance information, we can estimate the term weight via *idf*. Croft [13], Yu [65] and Robertson [50], etc. have investigated *idf* heuristics against the probabilistic model. More recently, [34], [51], and [56] highlighted relationships between the three main classes of models: tf-idf, BIRM, and LM. In particular, the research on the relationships of models provides input to this paper where we model retrieval models. The work on relationships of models isolates the common components (probability estimations) in models that are the basic ingredients for modelling retrieval models.

3 Running example

This section contains a toy database with two tables, a table named "Person" containing data about persons, and a table named "Coll" representing a document collection. We use these two tables to underline that the SQL-based and PSQL-based implementations investigated in this paper are of generic nature, and are not restricted to document retrieval.

Consider the table "Person" in Fig. 4. For this table, we will show how to describe in PSQL probabilities such as $P(\text{Nationality}|\text{City})$. The contribution of our paper is to add appropriate concepts to SQL, and to prove that the estimations are applicable in large-scale applications with millions of tuples.

To illustrate the application of PSQL to the classical IR task of text retrieval, we use the table/relation "Coll" shown in Fig. 5. Our toy collection has ten tuples, four terms (sailing, boats, east, coast), and five documents (doc1 to doc5). The single horizontal lines we use in the instance (tuple) part of a table are here to help the reader to locate the tuples that belong to one document.

Person		
Name	City	Nationality
Peter	London	German
Paul	London	Irish
Mary	London	Irish
Thomas	Dortmund	German
Thomas	London	German
Thomas	Hamburg	German
Hany	London	Egyptian
Hany	London	Polish
Jun	London	Chinese
Zhi	London	Chinese

Fig. 4 Relational table for modelling persons

Coll	
Term	DocId
sailing	doc1
boats	doc1
sailing	doc2
sailing	doc2
boats	doc2
sailing	doc3
east	doc3
coast	doc3
sailing	doc4
boats	doc5

Fig. 5 Relational table for document retrieval

The term "sailing" occurs in four documents, "boats" occurs in three documents, and "east" and "coast" occur in one document. The term "sailing" occurs twice in document doc2; otherwise, all term occurrences in documents are single occurrences.

A typical document retrieval task, for example, find all documents about sailing boats, can be easily expressed in SQL. Before expressing document retrieval, we give a simple query on "Person", to illustrate the analogy between traditional data retrieval and document retrieval:

Find all persons of Chinese or Polish nationality.

This query expressed in SQL is as follows:

```
SELECT Name
FROM Person
WHERE Nationality = 'Chinese'
OR Nationality = 'Polish';
```

We could also view the nationalities as query terms. Assume we have a relation "Query(Term,QueryId)". Then:

```
INSERT INTO Query VALUES
('Chinese', 'q1'), ('Polish', 'q1');
```

Next, we join the "Query" table with the "Person" table and retrieve the attribute Person.Name:

```
SELECT Person.Name
FROM Query, Person
WHERE Query.Term = Person.Nationality;
```

Compare the above formulation with the next one showing document retrieval for all documents about sailing boats:

```
INSERT INTO Query VALUES
('sailing', 'q2'), ('boats', 'q2');

SELECT DocId
FROM Query, Coll
WHERE Query.Term = Coll.Term;
```

The structures of those queries are very similar. If we had a standard SQL++ (SQL++ stands here for a SQL with relevance-based ranking) that sorts the retrieved tuples by relevance, then we could easily obtain a ranked retrieval result.

We will extend in Sect. 8 on how tf-idf-based retrieval (tf-idf is probably the most known, easy and effective ranking method) could be implemented in traditional SQL. However, though the traditional SQL is capable of modelling relevance-based ranking, the implementation has what can maybe best described as an "assembler-like" feel, since we describe in SQL the arithmetic to compute the retrieval status values. From an abstraction point of view, and from a probabilistic modelling point of view, this is not satisfactory. We require a more abstract and tailored SQL++, and therefore we introduce and investigate in this paper a probabilistic version of SQL.

An intuitive probabilistic approach would work with probabilistic relations "probQuery", "probPerson", and "probColl". Let these three relations be probabilistic relations in which tuple probabilities somehow (we extend later in the paper in detail how) reflect importance/relevance. For example, consider a possible instantiation of table "probColl" in the following.

probColl		
Prob	Term	DocId
0.5	sailing	doc1
0.5	boats	doc1
0.66	sailing	doc2
0.33	boats	doc2
0.33	sailing	doc3
0.33	east	doc3
0.33	coast	doc3
1.0	sailing	doc4
1.0	boats	doc5

The tuple probabilities here are a result of viewing {Term, DocId} as a *frequency key*, and {DocId} as an—what we call later—*evidence key*. Take document doc2. It has 3 tuples, thus, $1/3 = 0.33$ is the base probability of each doc2 tuple. With the frequency key (Term, DocId), two sailing tuples coincide, and if we add their probabilities, then we obtain the probabilistic tuple 0.66(sailing,doc2) in table "probColl". The double vertical line separates the probabilities from the ordinary attribute values. The double line underlines that probabilities are different from ordinary attribute values: The column *Prob* cannot be referred to in probabilistic SQL.

The probabilities in "probColl" reflect a conditional probability denoted as $P(t|d)$, i.e., the probability that the term t occurs given the document d. This probability is related to what is known in IR as *tf* (within-document term frequency).

Assume for now that there is an operator that produces "probColl" from "Coll" (this is the role of the relational Bayes, Sect. 6). Further, assume that we can assign query term probabilities, where the query term probabilities reflect the inverse document frequency (*idf*) of a term. Let sailing be more frequent than boats, hence we obtain in "probQuery" a lower probability for sailing than for boats. Join "probQuery" and "probColl" in a probabilistic SQL environment, and we have implemented something very close to tf-idf-based retrieval.

```
INSERT INTO probQuery VALUES
0.4 ('sailing', 'q2'), 0.6 ('boats', 'q2');

CREATE VIEW retrieved AS
SELECT DocId
FROM probQuery, probColl
WHERE probQuery.Term = probColl.Term;
```

In the view "retrieved", we obtain:

retrieved	
Prob	DocId
$0.4 \cdot 0.5$	doc1
$0.6 \cdot 0.5$	doc1
$0.4 \cdot 0.66$	doc2
$0.6 \cdot 0.33$	doc2
$0.4 \cdot 0.33$	doc3
$0.4 \cdot 1.0$	doc4
$0.6 \cdot 1.0$	doc5

Remains the question of how to aggregate the non-distinct tuples per document. A probabilistic disjunction seems reasonable. For this we could argue for 'disjoint' (add probabilities), 'independent' (add probabilities and subtract probability of intersections), or 'subsumed' (choose maximal probability). The assumption made for the aggregation will depend on the assumptions made when assigning (generating!) the probabilities in the relations "probColl" and "probQuery".

The aims of this paper are to formalise PSQL, to show the PSQL to PRA translation, and to investigate whether we can model state-of-the-art retrieval models in PSQL/PRA. In addition, the question is whether PSQL/PRA is scalable, and whether we gain an efficiency advantage compared to traditional SQL.

4 Requirements

Classical approaches to probabilistic databases focus on probability *aggregation*. They rely on "some external application" (this is how a peer colleague referred to it) to estimate initial tuple probabilities. Once the initial probabilities are available, it appears straight-forward to define for each algebraic operator reasonable probability aggregation functions.

The main problem with this approach is that external applications estimate probabilities *outside* of the probabilistic relational paradigm. Inside the classical probabilistic relational paradigm, there is no operator, no support, and no control for estimating probabilities from a non-probabilistic or probabilistic relation!

Consider the estimation of probabilities from the non-probabilistic relation "Coll(Term,DocId)" in Fig. 5. The requirement is to assign probabilities to tuples that reflect probabilities such as $P_{Coll}(t|d)$, $P_{Coll}(d|t)$, $P_{Coll}(t)$ and $P_{Coll}(t, d)$, where t is a term, and d is a document. The subscript of the probability function indicates the relation from which the probabilities are generated or estimated.

Such probabilities can be estimated in various ways. One important feature of an estimation is whether the estimation is based on

- the *tuple frequency*, or
- the *value frequency*.

We illustrate the two different frequencies with some examples.

Tuple frequency: We estimate the probability $P_{T,Coll}(t|d)$, where the subscript $T, Coll$ indicates that the tuples of relation "Coll" form the event space. One intuitive choice to estimate such probability is the maximum likelihood estimate, namely the number of d-tuples in which t occurs. For example, we have $P_{T,Coll}(\text{sailing}|\text{doc1}) = 1/2$, and $P_{T,Coll}(\text{sailing}|\text{doc2}) = 2/3$, since doc1 occurs in two tuples of which sailing occurs in one, and doc2 occurs in three tuples of which sailing occurs in two.

Value frequency: We estimate the probability $P_{V,Coll[DocId]}(t)$, i.e., the probability that t occurs in the value-based event space formed by the values in Coll[DocId]. The subscript $V, Coll[DocId]$ indicates the value-based event space for the value key Coll[DocId]. If we base the estimation on the value space that is formed by the distinct documents, then, we have five values, and we obtain, for example, $P_{V,Coll[DocId]}(\text{sailing}) = 4/5$. Note that this value frequency-based probability is different from the tuple frequency-based probability, where we obtain $P_{T,Coll}(\text{sailing}) = 5/10$.

As we will see in Sect. 7, both, tuple and value frequencies are essential for modelling retrieval models. We conclude this section with a formalisation of tuple and value frequency.

Definition 1 (Tuple frequency): Let $n_T(\tau, R)$ denote the tuple frequency, i.e., the number of *tuples* (hence, the T subscript) in relation R, that match the partial tuple τ, where a partial tuple is a tuple with some unspecified attribute values.

For example, for the partial tuple $\tau = (\text{sailing}, \cdot)$, $n_T((\text{sailing}, \cdot), \text{Coll}) = 5$ is the number of tuples in relation "Coll" that match the partial tuple (sailing, ·).

For the unspecified tuple, we obtain the number of tuples in the relation, i.e. $N_T(R) := n_T((\cdot, \cdot, \ldots), R)$ is the total number of tuples in relation R.

Let $h = h_1 \ldots h_n$ and $e = e_1 \ldots e_n$ be lists of attribute values. For example, h = sailing and e = doc1 are lists (lists with just one element) of attribute values. We chose h and e borrowing from the notion of 'hypothesis' and 'evidence' used in the Bayes theorem.

Then, the tuple-based probability $P_{T,R}(h|e)$ estimated based on the tuples in relation R is defined as follows:

Definition 2 (Tuple-based probability):

$$P_{T,R}(h|e) := \frac{n_T((h, e), R)}{n_T((\cdot, e), R)} \tag{1}$$

For example, let t be a value of the key Coll[Term], and let d be a value of the key Coll[DocId]. Then, we obtain the following tuple-based probability:

$$P_{T,Coll}(t|d) = \frac{n_T((t, d), Coll)}{n_T((\cdot, d), Coll)} \tag{2}$$

Here, $\tau = (t, d)$ is a tuple instance, and $\tau = (\cdot, d)$ is a partial instance, where the centered dot means to discard the first attribute value.

For the relation "Coll", we obtain for example: $P_{T,Coll}(\text{sailing}|\text{doc2}) = 2/3$, since $n_T((\text{sailing}, \text{doc2}), \text{Coll}) = 2$, and $n_T((\cdot, \text{doc2}), \text{Coll}) = 3$.

The tuple-frequency-based probability $P(t|d)$ is crucial to IR; it corresponds to the so-called within-document term frequency (tf) of a term. We refer to tuple-frequency-based probabilities for short as tuple-based probabilities.

Another crucial probability in IR is the probability $P(t)$, namely the probability that term t occurs. Here, both tuple and value frequency-based probabilities are common. As we will see in Sect. 7 on modelling retrieval models, the value-based probability $P_{V,Coll[DocId]}(t)$ is fundamental to the binary independent retrieval model, and the tuple-based probability $P_{T,Coll}(t)$ is fundamental to language modelling.

For counting the number of values with which a partial tuple is associated, we need a further notation. We refer with $n_V(h, R[E])$ to the number of E-values (evidence values) with which the hypothesis key h is associated. We define the value frequency formally, and keep the definition analogous to the definition of the tuple frequency (see Definition 1).

Definition 3 (Value frequency): Let $n_V(h, R[E])$ denote the value frequency, i.e., the number of values in key $R[E]$ that are associated with the hypothesis h, where h is a list of attribute values, R is a relation name, and E is a list of attribute names.

For example, $n_V((\text{sailing}), \text{Coll[DocId]}) = 4$ is the number of documents (values of attribute DocId) in which sailing occurs.

The use of upper-case E in the definition of the value frequency, as opposed to the use of lower-case e in the definition of the tuple frequency underlines that E refers to attribute names, whereas e refers to attribute values.

Next, we define analogous to Definition 2 the value-based probability of a hypothesis.

Definition 4 (Value-based probability):

$$P_{V,R[E]}(h) = \frac{n_V(h, R[E])}{N_V(R[E])} \tag{3}$$

For example, let t be a value of Coll[Term]. Then, we obtain

$$P_{V,Coll[DocId]}(t) = \frac{n_V((t), Coll[DocId])}{N_V(Coll[DocId])} \tag{4}$$

Compare the value-based probability above to the tuple-based probability below:

$$P_{T,Coll}(t) = \frac{n_T((t, \cdot), Coll)}{N_T(Coll)} \tag{5}$$

The difference between tuple-based and value-based probabilities is illustrated for the attribute Coll[Term] in the following table:

Term	Probabilities	
	tuple-based	value-based
sailing	5/10	4/5
boats	3/10	3/5
east	1/10	1/5
coast	1/10	1/5

We have discussed the requirements on probability estimation in general, thereby relating the discussion and examples to the probabilities typically required by retrieval models. Before we develop in Sect. 6 the means to describe probability estimation in the probabilistic relational framework, we look in the next section at probability aggregation.

5 PSQL and PRA: probability aggregation: classical operators

In this section, we present PSQL and PRA. We include the basic and composed operators, and show how probability aggregation works.

```
psqlSelect ::= 'SELECT' aggAssumption sqlTargetList
    'FROM' relationList
    'WHERE' sqlCondition
aggAssumption ::= assumption
assumption ::= 'disjoint' | 'independent' | 'subsumed'
sqlTargetList ::= ... as in SQL ...
relationList ::= ... as in SQL ...
sqlCondition ::= ... as in SQL ...
```

Fig. 6 Basic PSQL Syntax

```
prae ::= Selection | Projection |
    Product | Union | Subtraction
Selection ::= 'Select' '[' praCondition ']' '(' prae ')'
Projection ::=
    'Project' assumption '[' praTargetList ']' '(' prae ')'
Product ::= 'Multiply' assumption '(' prae ',' prae ')'
Union ::= 'Unite' assumption '(' prae ',' prae ')'
Subtraction ::= 'Subtract' assumption '(' prae ',' prae ')'
```

Fig. 7 PRA Syntax

5.1 Basic operators

For the basic operators, we present the syntax in Sect. 5.1.1, the translation of PSQL to PRA in Sect. 5.1.2, and the semantics of the basic PRA operators in Sect. 5.1.3.

5.1.1 Syntax of PSQL and basic PRA

This section presents the formal definition of the syntax of basic PSQL and PRA. The syntax of PSQL is very similar to the syntax of classical SQL, apart from few minimal extensions. For example, in the PSQL SELECT statement, one can specify the *aggregation assumption* which is one of 'disjoint', 'independent', or 'subsumed'. Consider the specification of the PSQL syntax in Fig. 6. Terminal symbols are indicated by single quotes. We present the SELECT statement only, which is sufficient for the purpose of this paper.

For processing PSQL, PSQL is translated to PRA, and this translation constitutes the semantics of PSQL. Figure 7 shows the PRA syntax, and the semantics of the PRA expressions (PRAE: probabilistic relational algebra expression) will be defined after the PSQL to PRA translation.

This syntax shows that basic PRA is—apart from the non-terminal "assumption" structured as traditional (non-probabilistic) relational algebra.

5.1.2 Translation of PSQL to basic PRA

Figure 8 illustrates the translation of PSQL to PRA, and Fig. 9 shows an example. The example matches a query index Query(Term,QueryId) against a document index Coll(Term, DocId).

The illustration and the example underline that the translation works very much as usual. The only difference is the probabilistic assumption: the aggregation assumption

```
-- PSQL
SELECT aggAssumption sqlTargetList
FROM ...
WHERE sqlCondition;

# PRA
Project aggAssumption[praTargetList](
  Select[praCondition](
    Multiply(Multiply(...));
# The Multiply(...) expression captures the
# relations in FROM ...
```

Fig. 8 Illustration of the translation of PSQL to PRA

```
-- PSQL example
SELECT DISJOINT
  QueryId, DocId
FROM Query, Coll
WHERE Query.Term = Coll.Term;

# PRA of the PSQL example
Project disjoint[$2,$4](
 Select[$1=$1](
   Multiply(Query, Coll)));
```

Fig. 9 Example of the translation of PSQL to PRA

"aggAssumption" becomes the assumption of the algebraic projection that "selects" the target attributes specified in the SQL SELECT statement. The sqlSelect is translated into an algebraic expression of the form Project[...](Select[...](Multiply(...))), where the algebraic 'Select' captures the sqlCondition, and the algebraic 'Project' "selects" the target attributes. In the paper, upper case "SELECT" indicates a PSQL statement, whereas "Select" indicates the PRA operator.

5.1.3 Semantics of basic PRA operators

A probabilistic relational algebra expression (PRAE) yields a probabilistic relation. A probabilistic relation is a pair (T, P), where T is a set of tuples and P is a probability function $P : T \rightarrow [0; 1]$, i.e., P maps each element (tuple) of T to a value (probability) of the interval [0; 1].

Some may view the specification of P and T redundant, in the sense that we could view T as the set of tuples τ with $P(\tau) > 0$. However, tuples with probability equal to zero are *in* a relation, i.e., $\tau \in T$ holds, and this is different from tuples that are not in a relation. As a first example of the meaning of zero probability tuples, consider a relation that contains terms, and the tuple (term) probability reflects the percentage of documents in which the term occurs. The term space might contain terms that do not occur in any document. By simply discarding zero probability tuples, we would loose information. For example, we will point out for the relational Bayes how to compute a notion of "being informative". A term that occurs in all documents is not informative, i.e., in the occurrence-based term space, such a term has a probability of 1.0 (occurs in all documents), whereas in the informativeness-based term space, such a term has the probability 0.0. Therefore, we distinguish between tuples with probability zero, and tuples that are not part of a relation.

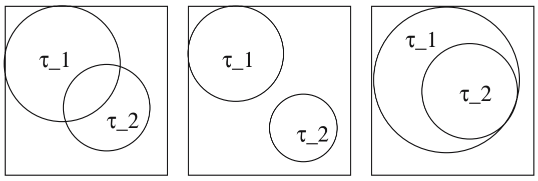

Fig. 10 Assumptions: independent, disjoint and subsumed

Before we define the relational operators, consider Fig. 10 illustrating the set-based meaning of the common probabilistic assumptions "independent", "disjoint" and "subsumed". For these three assumptions, the aggregation of probabilities for the disjunction (union), conjunction (intersection) and negation of events is as follows:

$$P(\tau_i \vee \tau_j) = \begin{cases} P(\tau_i) + P(\tau_j) - P(\tau_i) \cdot P(\tau_j) & \\ & \text{if independent} \\ P(\tau_i) + P(\tau_j) & \text{if disjoint} \\ \max(\{P(\tau_i), P(\tau_j)\}) & \text{if subsumed} \end{cases}$$

$$P(\tau_i \wedge \tau_j) = \begin{cases} P(\tau_i) \cdot P(\tau_j) & \text{if independent} \\ 0 & \text{if disjoint} \\ \min(\{P(\tau_i), P(\tau_j)\}) & \text{if subsumed} \end{cases}$$

$$P(\tau_i \wedge \neg\tau_j) = \begin{cases} P(\tau_i) \cdot (1 - P(\tau_j)) & \text{if independent} \\ P(\tau_i) & \text{if disjoint} \\ P(\tau_i) - P(\tau_j) & \text{if subsumed} \wedge \\ & P(\tau_i) > P(\tau_j) \\ 0 & \text{if subsumed} \wedge \\ & P(\tau_i) \leq P(\tau_j) \end{cases}$$

Next, we formalise the five basic relational operators. The definitions are composed as follows: Each definition starts with an assignment of the form $(T, P) =$ syntactic PRAE, where (T, P) is a probabilistic relation (set of tuples and probability function), and the right side is a syntactic form of the respective relational algebra expression. The definitions for T and P give the semantics of the PRAEs. Relational operators are applied to arguments (probabilistic relations), and we use "a" and "b" to refer to the argument relations.

We start with the definition of the selection.

Definition 5 (Selection):
$(T, P) =$ 'Select'[condition](a)

$$T := \{\tau | \tau \in T_a \wedge \varphi(\tau)\}$$
$$P(\tau) := P_a(\tau)$$

Here, φ represents the semantic truth value function that corresponds to the syntactic "condition" in the selection.

The probabilistic relation (T, P) is the result of the selection, and (T_a, P_a) is the probabilistic relation of the argument relation "a" of the selection.

For example, Select[$1=sailing](Coll) is a selection on the relation "Coll", and the tuples with "sailing" in their first column are selected.

This first definition does not manipulate the probabilities of the selected tuples. However, there are cases where a selection generates probabilities. For example, consider a *vague* selection such as Select[$price IS LOW & $mileage IS LOW](cars). Here, "IS LOW" is a vague predicate. Vague predicates (also referred to as vague selections) generate probabilities. This is the field of Fuzzy SQL. Fuzzy SQL can be expressed in probabilistic relational modelling. The expression Select[$price IS LOW](cars) could be viewed as a composed operation Select[](Join[$price=$price](cars, lowPrice). The empty condition of the selection indicates that this modelling of a vague predicate can be viewed as a complex condition pushing, where the vague predicate leads to a join expression joining cars with a probabilistic relation "lowPrice" to model the probability that a price is low. The modelling of vague predicates is discussed in [24].

Next, we define the projection, an operation that performs probability aggregation. The definition highlights an important difference to the work in [24]. The probabilistic assumption for specifying the probability aggregation is associated with the *algebra operation*, whereas in [24], the probabilistic assumption is associated with each tuple, this being achieved by assigning so-called event expressions to tuples. The event expressions allow for a delayed probability computation, and, overall, they allow for an intensional semantics of the probability computation. These are powerful features. However, the intensional case leads to scalability problems, since complex event expressions have to be transformed into disjunctive normal form. Therefore, in most applications, we apply extensional semantics. For the scope of the algebra variant we discuss here, we work with extensional semantics, i.e., each algebra expression directly aggregates probabilities.

We give next the definition of the probabilistic projection where the assumption is a parameter of the operator.

Definition 6 (Projection):

Let $\tau = \tau'[i_1 \ldots i_n]$ be a tuple composed of the attribute values at columns (positions) $i_1 \ldots i_n$ in tuple τ'.

Let $T_a(i_1 \ldots i_n)$ be the set of tuples of relation "a" that share the same attribute values at columns $i_1 \ldots i_n$.

(T, P) = 'Project' assumption[praTargetList](a)

$$T := \{\tau | \tau = \tau'[i_1 \ldots i_n] \wedge \tau' \in T_a\}$$

$$P(\tau) := \begin{cases} \sum_{\tau \in T_a(i_1 \ldots i_n)} P_a(\tau) \text{ if assumption='disjoint'} \\ 1 - \prod_{\tau \in T_a(i_1 \ldots i_n)} (1 - P_a(\tau)) \\ \quad \text{if assumption='independent'} \\ \max(\{P_a(\tau) | \tau \in T_a(i_1 \ldots i_n)\}) \\ \quad \text{if assumption='subsumed'} \end{cases}$$

If no praTargetList is specified, i.e., Project assumption(a), then this is equivalent to the praTargetList that contains all attributes of the argument relation "a".

For example, assume the following relation to be given:

docFreqSpace		
Prob	Term	DocId
0.2	sailing	doc1
0.2	boats	doc1
0.2	sailing	doc2
0.2	boats	doc2
0.2	sailing	doc3
0.2	east	doc3
0.2	coast	doc3
0.2	sailing	doc4
0.2	boats	doc5

The relation has a constant tuple probability reflecting the probability to draw a document from the document space (the set of five documents). We show the relational description of the probabilistic relation "docFreqSpace" based on the non-probabilistic relation "Coll" in Sect. 6, when we have the relational Bayes defined.

The PRAE "Project disjoint[$1](docFreqSpace)" projects on the first column and forms the sum of non-distinct tuples that coincide in the disjoint projection. We obtain the following probabilistic relation:

dfTermSpace = Project disjoint[$1](docFreqSpace)	
Prob	Term
0.8	sailing
0.6	boats
0.2	east
0.2	coast

In the probabilistic relation "dfTermSpace", the probability of a term can be interpreted as the probability that the term occurs in a document of the collection. This is a value-based probability (see Definition 4), where here value-based corresponds to document-based. This demonstrates that the traditional IR notion of *document frequency (df)* translates to the more general notion of *value frequency (vf)* in the relational framework, where any set of attributes can be defined to form the *value key* on which the value frequency is based. The generalised notion of value-based versus document-based, and the value-based versus tuple-based probabilities play an important role when modelling retrieval models (Sect. 7).

For the binary operators product, union, and subtraction, we only give in the following the definitions.

Definition 7 (Product):

(T, P) = 'Multiply' assumption(a, b)

$$T := \{\tau | \tau_a \in T_a \wedge \tau_b \in T_b \wedge \tau = [\tau_a, \tau_b]\}$$

$$P(\tau) := \begin{cases} 0 \quad \text{if assumption='disjoint'} \\ P_a(\tau_a) \cdot P_b(\tau_b) \\ \quad \text{if assumption='independent'} \\ \min(\{P_a(\tau_a), P_b(\tau_b)\}) \\ \quad \text{if assumption='subsumed'} \end{cases}$$

Definition 8 (Union):
$(T, P) =$ 'Unite' assumption(a,b)

$$T := \{\tau | \tau \in T_a \vee \tau \in T_b\}$$

$$P(\tau) := \begin{cases} P_a(\tau) + P_b(\tau) \quad \text{if assumption='disjoint'} \\ P_a(\tau) + P_b(\tau) - P_a(\tau) \cdot P_b(\tau) \\ \quad \text{if assumption='independent'} \\ \max(\{P_a(\tau), P_b(\tau)\}) \\ \quad \text{if assumption='subsumed'} \end{cases}$$

Definition 9 (Subtraction):
$(T,P) =$ 'Subtract' assumption(a, b)

$$T := \{\tau | \tau \in T_a\}$$

$$P(\tau) := \begin{cases} P_a(\tau) \text{ if assumption='disjoint'} \\ P_a(\tau) \cdot (1 - P_b(\tau)) \\ \quad \text{if assumption='independent'} \\ \max(P_a(\tau) - P_b(\tau), 0) \\ \quad \text{if assumption='subsumed'} \end{cases}$$

The definitions of product and subtraction raise an interesting theoretical (and philosophical) issue since tuples with probability zero or even negative probabilities may be produced.

For example, in a subsumed subtraction, given $P_a(\tau) < P_b(\tau)$, we might obtain $P(\tau) < 0$ in the result if we simply subtract probabilities. However, the illustration of the assumption "subsumed" in Fig. 10 shows that the interpretation of a subtraction as the Boolean combination "AND NOT" leads to $P(\tau) = 0$ if $P_a(\tau) \leq P_b(\tau)$. Thus, we have so far no operation or aggregation that generates negative probabilities.

Regarding zero probabilities, the aggregation is well-defined for all operations. An intuitive optimisation idea is to discard tuples with probability zero. However, from an information point of view, the information that a tuple is in a relation with probability zero is different from the information that a tuple is not in a relation. This will become even more evident when we apply the relational Bayes for generating informativeness-based probabilities, where then a zero probability tuple tells us that an attribute value (for example, "sailing" is an attribute value) is not informative, i.e., that it occurs in all elements of the event space (tuples or values, where, for example, documents are values).

To conclude this section with an example involving several algebra expressions, we return to our running example on document retrieval.

Query	
Term	QueryId
sailing	q1
boats	q1
sailing	q2
boats	q2
east	q2
coast	q2
...	...

Coll	
Term	DocId
sailing	d1
boats	d1
sailing	d2
sailing	d2
boats	d2
...	...

From this, we obtain a probabilistic representation of queries and documents (the next section explains how the probabilities would be generated). For example:

probQuery		
Prob	Term	QueryId
0.5	sailing	q1
0.8	boats	q1
0.5	sailing	q2
0.8	boats	q2
1.0	east	q2
1.0	coast	q2
...	...	...

probColl		
Prob	Term	DocId
0.5	sailing	doc1
0.5	boats	doc1
0.33	sailing	doc2
0.33	sailing	doc2
0.33	boats	doc2
...	...	...

In "probQuery", the probabilities might reflect the discriminativeness of a term: sailing is frequent, boats is less frequent, and east and coast are rare.

Given these probabilistic representations of queries and documents (collection), we can now formulate a retrieval strategy (ranking function) as follows:

```
-- PSQL
CREATE VIEW retrieve AS
SELECT DISJOINT QueryId, DocId
FROM probQuery, probColl
WHERE probQuery.Term = probColl.Term;
```

The translation of the PSQL statement to PRA yields:

```
# PRA
retrieve =
  Project disjoint[$2,$4](
    Select[$1=$1](
      Multiply(probQuery, probColl)))
```

We have defined the five basic operators, including PSQL and PRA examples. In the next section, we define the main composed operators, namely join and division.

5.2 Composed operators

There are a number of composed operators that are equivalent to an expression involving the basic operators. In this paper, we emphasise the following points:

1. The composed operators of classical relational algebra apply in the probabilistic case as well.
2. The division operator does not divide probabilities.

Probably the most common composed operator is the join, basically a product allowing for the specification of a condition:

```
Join assumption[joinCondition] (a,b) :=
    Select[selectCondition](
       Multiply assumption (a,b))
```

The definition shows, that the conditional join is viewed to be equivalent to a selection on the product of two relations. The conditional join is important since it is more elegant to use, and more efficient to compute than its decomposed equivalent. For example, consider the join of “Query” and “Coll” over the attribute “Term”:

```
Join[$1=$1](Query, Coll) :=
  Select[$1=$3](Multiply(Query, Coll))
```

The example illustrates that the column specification in the join condition and select condition are different. In the join condition, the column specification is such that the columns refer to the first and second argument of the join, respectively, whereas in the select condition, the columns refer to the concatenated tuple being the result of Multiply(Query,Coll). When pushing a select condition to a join operation, the column specification is adapted accordingly. We define the PRA here with column numbers, however, we may also use expressions such as Join[$Term=$Term](Query,Coll), given that the schema (the attribute names) of “Query” and “Coll” are defined.

To continue the illustration of the semantics of PRAE, consider the following example in which we work with a simple relation “probQuery(Term)” that contains no QueryId, but just terms. Further, we model a relation “probColl” with a more diverse distribution of probabilities than used in previous examples. In “probColl”, we insert horizontal lines to make it easier to associate tuples that belong to the same document.

probQuery	
Prob	Term
0.5	sailing
0.5	boats

probColl		
Prob	Term	DocId
0.8	sailing	doc1
0.6	boats	doc1
0.6	sailing	doc2
0.4	boats	doc2
1.0	sailing	doc3

Joining “probQuery” with “probColl” yields:

resultBody = Join[$1=$1](probQuery, probColl)			
Prob	probQuery.Term	probColl.Term	DocId
$0.4 = 0.5 \cdot 0.8$	sailing	sailing	doc1
$0.3 = 0.5 \cdot 0.6$	boats	boats	doc1
$0.3 = 0.5 \cdot 0.6$	sailing	sailing	doc2
$0.2 = 0.5 \cdot 0.4$	boats	boats	doc2
$0.5 = 0.5 \cdot 1.0$	sailing	sailing	doc3

A projection on the third column yields a relation with retrieved documents. Applying a disjoint projection, we obtain:

result = Project disjoint[$3](resultBody)	
Prob	DocId
$0.7 = 0.4 + 0.3$	doc1
$0.5 = 0.3 + 0.2$	doc2
$0.5 = 0.5$	doc3

Here, doc1 is estimated to be the most relevant, and doc2 and doc3 are estimated to be equally relevant documents.

Alternatively, using an independent projection, the aggregation of probabilities has the effect that doc2 is viewed less likely to be relevant than doc3, as shown below:

result = Project independent[$3](resultBody)	
Prob	DocId
$0.58 = 0.7 - 0.4 \cdot 0.3$	doc1
$0.44 = 0.5 - 0.3 \cdot 0.2$	doc2
$0.50 = 0.5$	doc3

This example illustrates the effect that probabilistic assumptions may have on the ranking.

Next, consider the division. We follow the definition to be found in [20, p. 224], and [59, p. 102], and other comprehensive database books. Let a and b be PRAEs. Let $X \cup Y$ be the attributes of a and let X be the attributes of b.

```
Divide(a,b) :=
  Subtract(
    Project[Y](a),
      Project[Y](
        Subtract(
          Multiply(b, Project[Y](a)), a)))
```

We show the division here to underline that the division is a composed operator, composed of the basic operators, and the basic operators perform probability aggregation (multiplication, summation, min/max) of probabilities, but do not divide probabilities (or frequencies) as this is required for probability *estimation*. We underline this point to fully clarify why the division is not capable of performing the division of probabilities (frequencies) as it is required for probability estimation.

Next, we step to a new, sixth basic operator of a probabilistic relational algebra, the relational Bayes. As the name indicates, the operator is related to Bayes’ Theorem.

6 PSQL and PRA: probability estimation: the relational Bayes

As highlighted throughout the paper, basic probabilistic relational algebra provides probability *aggregation* but lacks the means to describe probability *estimation*. Therefore, we introduce in this section the relational Bayes, the sixth operator of probabilistic relational algebra.

We present the relational Bayes in three steps, where the three steps are motivated by the type of estimation (probabilistic assumption). Section 6.1 introduces the relational Bayes for the classical assumptions “disjoint”, “independent”, and “subsumed”. Then, Sect. 6.2 adds the logarithmic assumptions, namely “max_log” and “sum_log”. The logarithmic assumptions support the estimation of what we refer to as “informativeness-based” probabilities, as opposed to the “occurrence-based” probabilities obtained via the classical assumptions. Finally, Sect. 6.3 introduces assumptions such as “max_idf” and “sum_idf”. These assumptions correspond to complex relational Bayes expressions and simplify PRA programs.

6.1 Classical assumptions: disjoint, independent and subsumed

6.1.1 Syntax and semantics of the relational Bayes

With respect to our running example, we are looking for a way to describe probabilities such as P_{Coll}(Term|DocId) and P_{Person}(Nationality). How can we describe such probabilities in PRA and PSQL, respectively?

We propose a basic probabilistic relational operator called Bayes. Basically, Bayes divides each tuple probability (the probabilities in a non-probabilistic relation are 1.0) by an aggregated tuple probability, which we refer to as *evidence* probability. For example, the sum of all doc1 tuples can be viewed as an evidence probability. The formal definition of the relational Bayes is:

Definition 10 (Bayes):
(T, P) = 'Bayes' assumption$[i_1 \dots i_n](a)$

$$T := \{\tau | \tau \in T_a\}$$
$$P(\tau) := \frac{P_a(\tau)}{P_b(\tau[i_1 \dots i_n])}$$

The key $i_1 \dots i_n$ is referred to as the *evidence key* since the relational Bayes generates a relation where the tuple probabilities correspond to the conditional probability $P(\tau|\tau[i_1 \dots i_n])$.

The probabilistic relation "b" is the so-called *evidence key* projection:

b = 'Project' assumption$[i_1 \dots i_n](a)$.

If no assumption is specified, i.e., given Bayes[...](...), then the assumption 'disjoint' is the default.

The relational Bayes performs a projection on the evidence key. We refer to the resulting probabilities of the inner projection as *evidence probabilities*. The inner projection is also referred to as evidence projection.

Given the result of the evidence projection, the relational Bayes computes the resulting probability (conditional probability) as the division of the tuple probability and the evidence probability. The division of probabilities reminds of the division operator. However, as the definition of the relational division shows, the relational division is based on the basic operators, and no division of probabilities is performed. Moreover, the relational Bayes is a basic operator, since no other even complex operation supports the division of probabilities.

As an example, consider the generation of a probabilistic relation where the probabilities are of the nature P(Term, DocId|DocId). In a column-based notation, we also write such a probability as $P(\$1, \$2|\$2)$, i.e., the second attribute forms the evidence key.

We name the relation as "tfCollSpace" (which is short for term frequency collection space). We generate "tfCollSpace" from the non-probabilistic relation "Coll" as follows:

tfCollSpace = Bayes[$2](Coll)

Prob	Term	DocId
0.5	sailing	doc1
0.5	boats	doc1
0.33	sailing	doc2
0.33	sailing	doc2
0.33	boats	doc2
0.33	sailing	doc3
0.33	east	doc3
0.33	coast	doc3
1.0	sailing	doc4
1.0	boats	doc5

For example, consider the computation of "0.33(sailing, doc2)". The probability of this tuple is the result of dividing the probability P_{Coll}((sailing,doc2)) $= 1.0$ by the evidence "probability" for "(doc2)", which is 3.0 since there are three tuples with doc2. The probabilistic semantics is $0.33 = \frac{1/10}{3/10}$, where $1/10$ is the probability that a tuple is drawn from relation "Coll", and $3/10$ is the probability that a doc2 tuple is drawn from relation "Coll".

Note that the relational Bayes operation preserves the non-distinct tuples, e.g. the tuple "0.33(sailing,doc2)" occurs twice in the result of the Bayes operation.

To aggregate the probabilities of non-distinct tuples, we apply a distinct projection, namely a disjoint projection in this case. We obtain:

tf = Project disjoint(Bayes[$2](Coll))

Prob	Term	DocId
0.5	sailing	doc1
0.5	boats	doc1
0.66	sailing	doc2
0.33	boats	doc2
0.33	sailing	doc3
0.33	east	doc3
0.33	coast	doc3
1.0	sailing	doc4
1.0	boats	doc5

The probabilities in the relation "tf" reflect the maximum-likelihood estimate of the form $n(t, d)/N(d)$. This linear estimate is an important estimate with a clear interpretation. To underline its general meaning, consider the estimation of a probabilistic relation with P(Nationality|City) (where we use attribute names in the PRA expressions).

```
# PRA
person_nationality_city =
  Project disjoint(
    Bayes disjoint[$city](
      Project[$nationality,$city](Person)));
```

Despite the fact that the maximum-likelihood is intuitive, it is known for IR that the maximum-likelihood estimate actually proves—for text retrieval—to be inferior to a Poisson-like (fractional) estimate where the probabilities in "tf" are computed via the fractional estimate of the form $n(t,d)/(K+n(t,d))$, where $n(t,d)$ is the number of (t,d)-tuples, and K is a constant to control the rise of the estimate, e.g. $K=1$. For the modelling of such a Poisson-based and—from a text retrieval point of view—effective retrieval function, we either need special Bayes assumptions, or other ways of aggregating probabilities. We will look into this requirement in Sect. 7 (modelling retrieval models).

We have described a *tf*-based probability $P_{T,Coll}(t|d)$, and this probability corresponds to what is in IR known as the *tf*-component in a *tf*-*idf*-like retrieval function.

Next we describe a tuple-based and a value-based term probability $P_{Coll}(t)$, i.e., the probability that a term t occurs. We refer with $P_{T,Coll}(t)$ to the tuple-based probability, and we refer with $P_{V,Coll[DocId]}(t)$ to the value-based probability. The subscript of the value-based probability indicates the attribute that forms the event space.

The tuple-based probability is essential for language modelling. The value-based term probability is the input to *idf*, since the *idf* of a term is defined as $idf(t, Coll) := -\log P_{Coll[DocId]}(t)$.

We name the tuple-frequency-based term space "tfTermSpace", whereas we name the value-frequency-based term space "dfTermSpace".

The tuple-based relation "tfTermSpace" is defined as the disjoint projection on the space of disjoint collection tuples.

```
tfTermSpace =
    Project disjoint[$Term](
        Bayes disjoint[](Coll));
```

In the relation "tfTermSpace", we obtain:

tfTermSpace	
Prob	Term
5/10	sailing
3/10	boats
1/10	east
1/10	coast

The tuple-based probability is fairly straight-forward, when compared to the value-based probability. For the value-based probability, we need to define a value space, and then join the distinct collection with the value space to project on the frequency key, so that the projection aggregates for each frequency key (term) the probabilities of the values (documents) in which the frequency key (term) occurs.

Consider the following PRA program for describing a value-based probability:

```
valueSpace =
    Bayes[](Project distinct[$DocId](Coll));

distinctColl = Project distinct(Coll);

dfTermSpace =
  Project disjoint[$Term](
```

```
psqlSelect ::= 'SELECT' aggAssumption sqlTargetList
    'FROM' relations
    'WHERE' sqlCondition
    [ 'EVIDENCE KEY' '(' sqlEvidenceKey ')' ]
    [ 'ASSUMPTION' estAssumption ]
sqlEvidenceKey ::= targetAttributeNameList
targetAttributeNameList ::= NAME |
    NAME ',' targetAttributeNameList
estAssumption ::= assumption
```

Fig. 11 Extended PSQL Syntax

```
Join[$DocId=$DocId](distinctColl,
                    valueSpace));
```

We obtain:

dfTermSpace	
Prob	Term
4/5	sailing
3/5	boats
1/5	east
1/5	coast

We have introduced the syntax and semantics of the relational Bayes. In addition, we have applied the relational Bayes to create three probabilistic views:

1. View "tf(Term,DocId)" where the tuple probability is a tuple-based probability of the nature $P_{T,Coll}$(Term| DocId). This relational view explains the generation of the previously mentioned relation "probColl". This *tf*-based relation is crucial to the two main retrieval models, namely *tf*-*idf* and language modelling.
2. View "tfTermSpace(Term)", where the tuple probability is a tuple-based probability $P_{T,Coll}(t)$. This relational view is important for language modelling.
3. View "dfTermSpace(Term)", where the tuple probability is a value-based probability $P_{V,Coll[DocId]}(t)$. This relational view corresponds to the relation "dfTermSpace" introduced in Sect. 5.1 when defining the basic operators. The value-based (document-based) term probabilities are important for *idf*.

In the next section, we show the embedding of the relational Bayes into PSQL.

6.1.2 Translation of PSQL to PRA with Bayes

Consider now the definition and translation of PSQL expressions that support probability estimation. In PSQL, we add two new clauses to the SELECT statement: The *evidence key clause*, and the *assumption clause*. Figure 11 shows the syntax of extended PSQL (refer to Fig. 6 for the syntax of basic PSQL). Terminal symbols of the syntax are set in single quotes.

The following PRA expression shows the principle translation of a PSQL SELECT statement to PRA, which includes evidence key or estimation assumption.

```
# PRA
'Bayes' estAssumption [ praEvidenceKey ] (
  'Project' aggAssumption [ praTargetList ] (
    'Select' [ praCondition ] (
      'Multiply' (...) ) ) )
```

The non-terminal symbols of the PSQL statement are translated into the respective PRA expressions. The sqlTargetList, sqlCondition and sqlEvidenceKey contain attribute names, whereas the corresponding PRA symbols may contain attribute names (if the PRA layer knows the schema) or columns.

The PSQL Select statement is translated as usual to a relational projection. If a PSQL statement contains an evidence key clause or an assumption clause, then the relational Bayes is applied to the result of the projection. If only an evidence key is specified, then the assumption "disjoint" is used. If only an estimation assumption is specified, then a Bayes without evidence key is applied ("Bayes(...)"), which means that the evidence key contains all attributes of the relational argument (the projection) of the relational Bayes.

For modelling *tf-idf*, we lack the functionality to estimate the probabilities that are proportional to the *idf* of a term. Therefore, we introduce in the next section two new assumptions, max_log and sum_log, which could be also referred to as logSubsumed (max_log) and logIndependent (sum_log), to highlight their relationship to the classical assumptions.

6.2 Logarithmic assumptions: max_log and sum_log

In IR, a crucial concept is the so-called inverse document frequency $idf(t, c)$ of a term t in a collection c, where *idf* is defined as the logarithm of a frequency-based probability. Let $P(t|c)$ be the probability that the term t occurs in the documents of the collection c. This probability is usually based on the number of documents in which t occurs (denoted as $n_D(t, c)$) and the number of documents in the collection (denoted as $N_D(c)$):

$$\begin{aligned} P(t|c) &:= \frac{n_D(t,c)}{N_D(c)} \\ idf(t,c) &:= -\log P(t|c) \end{aligned} \tag{6}$$

As we saw in Sect. 6.1, $P(t|c)$ can be expressed with the relational Bayes. However, so far, we lack the means to express a probability that is proportional to *idf*.

Therefore, we introduce further assumptions. Since the *idf*, the logarithm, respectively, is related to information theory, we refer to these logarithmic assumptions also as *information-theoretic* assumptions.

We extend in the following the definitions of Projection and Bayes regarding the assumptions max_log and sum_log.

Definition 11 (Logarithmic Projection): This definition extends Definition 6 (Projection). Definition 6 covers only the classical assumptions 'disjoint', 'independent', and 'subsumed'.

$$P(\tau) = \begin{cases} \min(\{P_a(\tau) | \tau \in T(i_1..i_n)\}) \\ \quad \text{if assumption='max\_log'} \\ \prod_{\tau \in T(i_1..i_n)} P_a(\tau) \\ \quad \text{if assumption='sum\_log'} \end{cases}$$

Having defined the semantics of the logarithmic (information-theoretic) assumptions for Project, we can define the logarithmic Bayes, which involves an evidence projection with a logarithmic assumption.

Definition 12 (Logarithmic Bayes): This definition extends Definition 10 (Bayes). Definition 10 covers only the classical assumptions 'disjoint', 'independent', and 'subsumed'.

$$\begin{aligned} (P_b, T_b) &= \text{'Project' assumption}[i_1, \ldots, i_n](a) \\ P(\tau) &= \frac{-\log P_a(\tau)}{-\log P_b(\tau[i_1, \ldots, i_n])} \\ &\quad \text{if assumption} \in \{\text{'max\_log'}, \text{'sum\_log'}\} \end{aligned}$$

For the logarithmic assumptions, the relational Bayes divides the logarithm of the tuple probability by the logarithm of the evidence probability, where the evidence probability is the minimum or the product of the probabilities of the evidence tuples.

For max_log, the maximum of logarithms is equal to the logarithm of the minimum of probabilities:

$$\begin{aligned} &\max(\{-\log P(\tau_1), \ldots, -\log P(\tau_n)\}) \\ &\quad = -\log \min(\{P(\tau_1), \ldots, P(\tau_n)\}) \end{aligned}$$

Hence, the evidence projection Project max_log[]() yields the minimum of the probabilities of the coinciding tuples.

For sum_log, the sum of logarithms is equal to the logarithm of the product of probabilities:

$$\sum_i -\log P(\tau_i) = -\log \prod_i P(\tau_i)$$

Hence, the evidence projection Project sum_log[]() yields the product of the probabilities of the coinciding tuples.

For the Projection, max_log corresponds to a conjunction of subsumed events, and sum_log corresponds to a conjunction of independent events. This is summarised in the following table.

Assumption	Evaluation in projection
max_log	conjunction of subsumed events
sum_log	conjunction of independent events

As an example, consider the computation of an *idf*-based term space:

max_idfTermSpace = Bayes max_log[](dfTermSpace)	
Prob	Term
$\log(0.8)/\log(0.2) \approx 0.1386$	sailing
$\log(0.6)/\log(0.2) \approx 0.3174$	boats
$\log(0.2)/\log(0.2) = 1.0$	east
$\log(0.2)/\log(0.2) = 1.0$	coast

The logarithmic assumptions add an important angle to the relational operators Projection and Bayes, since they allow for the description of so-called informativeness probabilities.

Before we conclude this section, we look at the irregularities of the logarithmic assumptions, namely $\log P(\tau)$ is zero for $P(\tau) = 1$, and it is not defined for $P(\tau) = 0$.

For max_log and sum_log, the evidence probability $P_b(\tau)$ is equal to 1.0 if all probabilities of the evidence tuples are 1.0, i.e., there exists no tuple probability less than 1.0. This means that all evidence is not informative, since only tuples (signals) which occur with a probability of less than 1.0 bear any surprise, and only surprise is considered to be informative.

The evidence probability is zero if there is one zero probability among the coinciding tuples in the evidence key projection. For this case, we can assign zero to the result tuples, since $\lim_{P(\tau)\to 0} - \log P(\tau) = \infty$.

The value-based dfTermSpace (end of Sect. 6.1) is a complex algebra expression. Thus, applying Bayes max_log[](dfTermSpace) is clearly not the easiest expression to evaluate. Therefore, the next section introduces composed Bayes assumptions named max_ivf (also referred to as max_idf) and max_itf (also referred to as max_ilf). These yield two advantages: on one hand, the algebraic expression becomes more compact, which is welcome when modelling in PRA. On the other hand, the composed expressions allow for an index usage that leads to a more efficient processing that the computation of the decomposed expressions.

6.3 Inverse frequency assumptions: max_ivf (max_idf) and max_itf (max_ilf)

Reconsider the computation steps for describing the estimation of an *idf*-based probability, where we work now with general relations (not specific to document retrieval). We generate an inverse-value-based (*ivf*-based) key space. For this, we apply the general notions of value key and frequency key: The value key (for text retrieval, {DocId} is the value key), and the frequency key (for text retrieval, {Term} is the frequency key), are used for defining the value frequency. Let r be a relation. The steps for describing the value frequency are:

1. Generate value space:

```
valueSpace =
  Bayes[](Project distinct[valueKey](r));
```

2. Generate distinct tuple space:

```
tupleSpace = Project distinct(r);
```

3. Generate value frequency:

```
vf =
  Project disjoint[freqKey](
    Join[valueKey=valueKey](
      tupleSpace, valueSpace));
```

4. Generate inverse-value-frequency-based probabilities:

```
max_ivf = Bayes max_log[](vf)
```

To facilitate the specification and evaluation of value-based probabilities, we define now the max_ivf relational Bayes. The max_ivf Bayes requires a projection on the frequency key to be its argument. For example, in "Bayes max_ivf[] (Project[$Term](Coll))", the "Term" attribute of "Coll" is the frequency key.

The distinct collection is joined with the valueSpace to obtain the base for generating the valueFrequencies (the relation "vf" in the derivation above corresponds for the relation "Coll" to the document frequency (df)).

```
Bayes max_ivf[](Project[freqKey](r)) :=
  Bayes max_log [] (
    Project disjoint [freqKey] (
      Join [ valueKey=valueKey ] (
        Project distinct ( r ),
        Bayes [] (
          Project distinct[valueKey] (r)))))
```

Since the general concept of an *ivf*-based probability has its origin in the IR concept of *idf*-based probabilities, we let max_idf be a synonym of max_ivf.

As an example, consider the *ivf* (inverse value (document) frequency) of attribute "Term" in relation "Coll":

```
Bayes max_idf[](Project[$Term](Coll)) :=
  Bayes max_log[](
    Project disjoint[$Term](
      Join[$DocId=$DocId](
        Project distinct(Coll),
        Bayes[](Project distinct[$DocId](Coll))))
```

Here, {Term} is the frequency key, and {DocId} is the value key.

Next, consider the generation of tuple-frequency-based informativeness probabilities via the composed Bayes max_itf operation:

```
Bayes max_itf[](Project[freqKey](r)) :=
  Bayes max_log[](
    Project disjoint[freqKey](Bayes[](r)))
```

The definition is simpler than that of max_ivf, since the tuple space generated by Bayes[](r) is direct input to the disjoint projection over the frequency key.

This section added the relational Bayes to the basic PRA. Now, the PRA components are the five basic operators, the composed operators, the relational Bayes primitives (disjoint, independent, subsumed, max_log, sum_log), and the composed relational Bayes expressions (max_ivf, max_itf). We have now a probabilistic relational paradigm suitable to describe the probability aggregation *and* estimation required for modelling IR models.

7 Probabilistic relational modelling of retrieval models

We start with probabilistic variants of a simple but effective retrieval model, known as tf-idf (Sect. 7.1). Then, we show the modelling of the two major probabilistic retrieval models: binary independent retrieval model (Sect. 7.2) and language modelling (Sect. 7.3). In addition to the modelling of retrieval models, we include the modelling of the most common evaluation measure: precision/recall (Sect. 7.4).

7.1 TF-IDF

The standard definition of the *tf-idf*-based retrieval status value (RSV) is of the form $\mathrm{RSV}(d,q) = \sum_{t \in d \cap q} \mathit{tf}(t,d) \cdot \mathit{idf}(t)$. When investigating the implementation of *tf-idf* in a probabilistic relational framework, we came across different variants we will report in this section. For implementing the standard form, we need to instantiate probabilistic relations to model *tf* and *idf*. Since we move in a probabilistic framework, we need to think about a probabilistic interpretation of *tf-idf*, or, at least, define probabilities that are proportional to *tf* and *idf*, respectively. This is fairly straight-forward for the *tf* component, but for the *idf* component, we need a log-based normalisation and the probabilistic interpretation of the value obtained is not obvious (see [53] for a discussion of the semantics of such a probability).

We illustrate in the following several *tf-idf* implementations.

Consider first the PSQL script for modelling standard *tf-idf*-based retrieval.

```
-- PSQL: standard tf-idf retrieval
-- Extensional relations:
-- Coll(Term, DocId);
-- tf_poissona(term, context);
-- Query(Term, QueryId);

-- within-document term frequency:
CREATE VIEW tfCollSpace AS
    SELECT Term, DocId
    FROM Coll
    ASSUMPTION DISJOINT
    EVIDENCE KEY (DocId);
CREATE VIEW tf AS
    SELECT DISJOINT Term, DocId
    FROM tfCollSpace;

-- Optional: Bind tf to extensional relation.
CREATE VIEW tf AS
    SELECT term AS Term, context AS DocId
    FROM tf_poissona;

-- inverse document frequency:
CREATE VIEW idf AS
    SELECT Term
    FROM Coll
    ASSUMPTION MAX_IDF
    EVIDENCE KEY ();

-- query term weighting and normalisation:
CREATE VIEW wQuery AS
    SELECT Term, QueryId
    FROM Query, idf
    WHERE Query.Term = idf.Term;
CREATE VIEW norm_wQuery AS
    SELECT Term, QueryId
    FROM wQuery
    EVIDENCE KEY (QueryId);

-- retrieve documents:
CREATE VIEW std_tf_idf_retrieve AS
    SELECT DISJOINT DocId, QueryId
    FROM norm_wQuery, tf
    WHERE norm_wQuery.Term = tf.Term;

-- Probabilistic interpretation:
-- For tf_poissona interpreted as P(d|t):
-- P(t|q) P(q) = P(q|t) P(t is informative | c)
-- RSV(d,q) = P(q) sum_t P(d|t) P(t|q)

CREATE VIEW retrieve AS
    SELECT DocId, QueryId
    FROM std_tf_idf_retrieve;
```

The PSQL script contains views for defining the probabilistic relations “tf” and “idf”. For “tf”, the first two views demonstrate how to define a maximum-likelihood estimate, which is of the form $P(t|d) = n(t,d)/N(d)$. This linear estimate is outperformed by a non-linear estimate of the form $n(t,d)/(n(t,d) + K)$, where $n(t,d)$ is the number of times term t occurs in document d, and K is a term-independent value, which might reflect, for example, the document length (BM25, [57]). This non-linear estimate can be viewed as a Poisson approximation, and the term-document pairs with the respective probabilities are stored in relation “tf_poissona”. We report at the end of this section the effect of different “tf” variants.

The query terms are joined with “idf” to generate the relation “wQuery” of weighted query terms. The normalised query terms are required for obtaining a probabilistic interpretation of the sum over the *tf-idf* products. Finally, we define the view “std_tf_idf_retrieve”, which contains the document-query pairs with their probabilistic *tf-idf* retrieval status values.

The translation of the PSQL script yields a PRA program that is equivalent to the PRA program shown next.

```
# PRA: tf-idf retrieval
# Extensional relations:
# Coll(Term, DocId);
# Query(Term, QueryId);

# tfCollSpace(Term, DocId):
tfCollSpace = Bayes[$2](Coll);
# tf(Term, DocId):
tf = Project disjoint[$1,$2](tfCollSpace);

# Optional: Bind tf to extensional relation.
tf = tf_poissona;

# idf(Term):
idf = Bayes max_idf[](Project[$1](Coll));

# wQuery(Term, QueryId):
wQuery =
    Project[$1,$2](Join[$1=$1](Query, idf));

# Normalisation:
norm_wQuery =
    Project[$1,$2](Bayes[$2](wQuery));

# Retrieve documents:
# std_tf_idf_retrieve(DocId, QueryId):
std_tf_idf_retrieve =
    Project disjoint[$4,$2](
        Join[$1=$1](norm_wQuery, tf));

retrieve = std_tf_idf_retrieve;
```

Each PRA equation corresponds to a view in the PSQL script. PSQL views that involve evidence key or assumption lead to PRA expressions in which the relational Bayes performs the required probability estimation. This is the case for the view "tfCollSpace" (see Sect. 6.1 for an example of the relation "tfCollSpace") and for the view "idf" (see Sect. 6.3 for the definition of the assumption max_idf).

We have modelled standard *tf-idf*. The maximum-likelihood estimate is a conceptual part of the minimal probabilistic relational framework we presented so far. It is one of the main contributions of the relational Bayes that such estimations are now part of the probabilistic relational paradigm, and do not need anymore to be computed *outside* of the relational algebra. However, for Poisson-like estimates, we still bind "tf" to the extensional relation "tf_poissona" in which probabilities were generated offline. There are numerous ways in the PSQL/PRA framework to specify Poisson-like probabilities, however, our aim is to integrate probability estimations neatly into the conceptual framework of probabilistic relational modelling, rather than inventing new assumptions and SQL syntax extensions for each way the probabilities can be estimated. The specification and semantics of Poisson-based and other probabilities actually requires to extend the framework we present here in this paper. The extension is based on providing more assumptions for the relational Bayes, and also on providing special assumption for the Join. Since these extensions significantly enhance and enlarge the framework, we focus in this paper on the minimal PRA and its relational Bayes, and we address the extensions in future work.

When implementing *tf-idf*, we encountered less complex PSQL programs that provide a *tf-idf*-like RSV. Consider in the following an alternative and fairly compact PSQL program, where we join *idf*-weighted query terms with the relation "Coll" rather than "tf". In "Coll", we have non-distinct Term-DocId tuples, whereas in "tf", tuples are distinct since the non-distinct Term-DocId tuples have been aggregated into the probabilities of the tuples in "tf".

```
-- PSQL: alternative tf-idf-like retrieval
-- This tf-idf variant does not rely on the
-- generation of an explicit tf relation.

CREATE VIEW alt1_tf_idf_retrieve AS
    SELECT INDEPENDENT DocId, QueryId
    FROM wQuery, Coll
    WHERE wQuery.Term = Coll.Term;
```

The translation to PRA yields:

```
# PRA
alt1_tf_idf_retrieve =
    Project independent[$4,$2](
        Join[$1=$1](wQuery, Coll));
```

The independence assumption leads to an aggregation of the query term probabilities such that we obtain for the probabilities in "alt1_tf_idf_retrieve": $\text{RSV}(d, q) = 1 - \prod_{(t,d)\in Coll} (1 - P(q|t))$. Note that the aggregation of non-distinct (t, d) tuples in the relation "Coll" reflects the within-document term frequency. The light-weight nature of this implementation motivated us to investigate the retrieval quality of this script against *tf-idf*-implementations that contain an explicit relation "tf".

For another candidate with explicit "tf", consider the following script in which we join the non-normalised rather than the normalised query term weights, and view the query terms as independent rather than disjoint.

```
-- PSQL: alternative tf-idf-like retrieval
-- Aggregation of independent, non-normalised
-- query term weights.

CREATE VIEW alt2_tf_idf_retrieve AS
    SELECT INDEPENDENT DocId, QueryId
    FROM wQuery, tf
    WHERE wQuery.Term = tf.Term;
```

Note the difference between "alt2_tf_idf_retrieve" and "std_tf_idf_retrieve": In "alt2_tf_idf_retrieve", we (have to) apply an independence assumption. Thus, a document that contains one very rare term with a high term frequency will be ranked very high, regardless of the other query terms. In "std_tf_idf_retrieve", we (had to) normalise the weighted query terms for the safe application of a disjoint projection.

To investigate the performance of different *tf-idf* notions that emerged when modelling *tf-idf* in PSQL/PRA, we ran

tf-idf	tf	wQuery	avg-prec	prec@10
std1	Poisson *tf*	normalised	0.2713	0.4138
std2	Likelihood *tf*	normalised	0.2077	0.4103
alt1	implicit *tf*	non-normalised	0.2038	0.4091
alt2	Likelihood *tf*	non-normalised	0.1224	0.2586

Fig. 12 Retrieval quality for *tf-idf* alternatives

the *tf-idf* variants on a 500 MB structured collection (INEX collection, http://inex.is.informatik.uni-duisburg.de/) with 12,000 articles and 15 million retrievable contexts (sections, paragraphs, etc). This leads to 32.5 million tuples in a representation similar to the relation "Coll" in our running example.

For the *tf-idf* variants, we obtain the retrieval quality presented in Fig. 12, where the variants are sorted by performance.

The experiment confirms *tf-idf* with Poisson-like *tf* to perform best. The standard variants (std1 and std2) work with normalised *idf*-based probabilities for query term weighting, whereas the alternative variants (alt1 and alt2) work with non-normalised query term weights. The variant with implicit *tf*, where the join of query terms with the relation "Coll" followed by an independent projection implicitly captures the *tf* part, performs quite well, taking into account that this implementation actually frees the system from providing a view "tf" or even a materialised relation.

Actually, it is in this paper not our aim to discuss retrieval quality. We know that depending on the application and data, we need to adjust retrieval strategies. What we do not know yet, but what we can investigate now given the expressiveness of PSQL, is for example which retrieval function is best to retrieve the 'Chinese or English people that we should recruit to open a business branch in China'. The point of PSQL is that we can define and refine ranking for any query, in particular for queries that involve complex relational schemas, and not just a relation of terms and document ids, as it is mostly the case in document retrieval applications.

What the *tf-idf*-variation demonstrates is that PSQL is flexible regarding probability estimation and aggregation, is applicable to large-scale data, and allows to formulate and investigate retrieval models in an abstract, relatively compact but still efficient representation.

7.2 Binary independent retrieval model (BIRM)

The binary independent retrieval model (BIRM, [55]) is a theoretical pillar of probabilistic retrieval. We investigate in this section the probabilistic relational modelling of the BIRM.

The BIRM defines the RSVas follows:

$$\mathrm{RSV}_{\mathrm{BIRM}}(d,q) = \sum_{t \in d \cap q} \left[\log \frac{P_D(t|q,r)}{P_D(\bar{t}|q,r)} - \log \frac{P_D(t|q,\bar{r})}{P_D(\bar{t}|q,\bar{r})} \right]$$

Here, the variations of the $P_D(t|q,r)$ probabilities are the document-frequency-based probabilities that term t occurs in the respective set of relevant and non-relevant documents. Assuming the collection to approximate the set of non-relevant (i.e., $c = \bar{r}$), and applying the *idf*-definition, the BIRM can be rewritten as a linear combination of *idf*-values ([18]):

$$\mathrm{RSV}_{\mathrm{BIRM}}(d,q) = \\ = \sum_{t \in d \cap q} \left[idf(t,c) - idf(t,r) + idf(\bar{t},r) - idf(\bar{t},c) \right]$$

The probabilistic relational implementation is based on the linear combination of *idf*-values. Based on whether or not the negative term events are taken into account, and based on the choice of the set of non-relevant documents, there are four variants of the BIRM. We present the implementation of the variant $idf(t,c) - idf(t,r)$ where we combine the positive term events in the collection and the set of relevant documents, and we disregard the negative term events.

In the PSQL implementation, we define accordingly the views "idf_c" and "idf_r". The PSQL script is as follows:

```
-- PSQL: birm retrieval
-- Extensional relations:
-- Coll(Term, DocId);
-- Query(Term, QueryId);
-- relevant(query, context);

-- collection of relevant documents:
CREATE VIEW relColl AS
    SELECT Coll.Term, DocId
    FROM Query, relevant, Coll
    WHERE Query.QueryId = relevant.query
    AND relevant.context = Coll.DocId;

-- idf in collection:
CREATE VIEW idf_c AS
    SELECT Term
    FROM Coll
    ASSUMPTION MAX_IDF
    EVIDENCE KEY ();

-- idf in relevant:
CREATE VIEW idf_r AS
    SELECT Term
    FROM relColl
    ASSUMPTION MAX_IDF
    EVIDENCE KEY ();

-- query term weighting:
CREATE VIEW wQuery_c AS
    SELECT Term, QueryId
    FROM Query, idf_c
    WHERE Query.Term = idf_c.Term;
CREATE VIEW norm_wQuery_c AS
    SELECT Term, QueryId
    FROM wQuery_c
    ASSUMPTION DISJOINT
    EVIDENCE KEY (QueryId);
CREATE VIEW wQuery_r AS
    SELECT Term, QueryId
    FROM Query, idf_r
    WHERE Query.Term = idf_r.Term;
CREATE VIEW norm_wQuery_r AS
    SELECT Term, QueryId
    FROM wQuery_r
    ASSUMPTION DISJOINT
    EVIDENCE KEY (QueryId);

-- combination of normalised weights:
CREATE VIEW wQuery AS
    norm_wQuery_c MINUS SUBSUMED norm_wQuery_r;
CREATE VIEW norm_wQuery AS
    SELECT Term, QueryId
    FROM wQuery
    ASSUMPTION DISJOINT
    EVIDENCE KEY (QueryId);

CREATE VIEW distinctColl AS
    SELECT DISTINCT Term, DocId
    FROM Coll;

-- retrieve documents:
CREATE VIEW birm_retrieve AS
    SELECT DISJOINT DocId, QueryId
    FROM norm_wQuery, distinctColl
    WHERE norm_wQuery.Term = distinctColl.Term;

CREATE VIEW retrieve AS
    SELECT DocId, QueryId
    FROM birm_retrieve;
```

The view “relColl” contains the Term-DocId tuples of the relevant documents. Then, the views “idf_c” and “idf_r” are defined over “Coll” and “relColl”, respectively. This is followed by query term weighting and the subsumed subtraction of query term weights. Finally, the join of query term weights and the distinct collection representation yields the retrieval result. Note that we join with a distinct view of the collection to reflect the nature of the BIRM.

The translation to PRA yields a PRA program equivalent to the PRA program shown next:

```
# PRA: birm retrieval
# Extensional relations:
# Coll(Term, DocId);
# Query(Term, QueryId);
# relevant(query, context);

# Collection of relevant documents:
# relColl(Term, DocId):
relColl=
    Project[$5,$6](
      Join[$4=$2](
        Join[$2=$1](Query, relevant), Coll));

# idf in collection:
idf_c = Bayes max_idf[](Project[$1](Coll));

# idf in relevant documents:
idf_r = Bayes max_idf[](Project[$1](relColl));

# Query term weighting:
wQuery_c =
    Project[$1,$2](Join[$1=$1](Query, idf_c));
wQuery_r =
    Project[$1,$2](Join[$1=$1](Query, idf_r));

# Normalisation:
norm_wQuery_c = Bayes[](wQuery_c);
norm_wQuery_r = Bayes[](wQuery_r);

# Combination of query term weights:
wQuery =
    Subtract subsumed(norm_wQuery_c,
                      norm_wQuery_r);
norm_wQuery = Bayes[](wQuery);

distinctColl = Project distinct[$1,$2](Coll);

# Retrieve documents:
birm_retrieve =
    Project disjoint[$4,$2](
       Join[$1=$1](norm_wQuery, distinctColl));

retrieve = birm_retrieve;
```

There are two equations for the *idf*-based probabilities of terms: “idf_c” for the collection, and “idf_r” for the set of relevant documents. The subsumed subtraction performs the linear combination $idf(t, c) - idf(t, r)$ for the respective query terms. The disjoint projection sums per document-query pair over the query term probabilities.

There are a number of issues regarding the implementation of BIRM. One issue is that the implementation shows the parallel between *tf*-*idf* and BIRM. The *tf*-*idf* script contains only the view “idf”, whereas the BIRM script contains the views “idf_c” and “idf_r”, and this clearly shows how the BIRM proposes to consider relevance information for query term weighting. Another issue is the semantics of the implementation. If a term is frequent in the collection, then it has a small probability in idf_c. If the same term is rare in the relevant documents, then it has a relatively large probability in idf_r. This is certainly a poor term for selecting relevant documents. According to the discussion for a subtraction over subsumed events in Sect. 5.1, such a term will have a probability of zero, and thus it will not affect the ranking. In the genuine formulation of the BIRM, poor terms have a negative impact on the RSV. To achieve a “correct” implementation of the BIRM, we would need negative probabilities, which we have excluded for now. Also, the max_idf-based normalisations consider the cardinality of the collection and the set of relevant documents, whereas the genuine formulation does not consider the cardinality. The relationship of the genuine BIRM and its probabilistic relational implementation is a topic of future research.

We have achieved a PSQL/PRA implementation of the BIRM, and we continue in the next section with the other main probabilistic approach to IR, namely language modelling.

7.3 Language modelling (LM)

Language modelling linearly combines the probability that a term occurs in the collection $P_T(t|c)$ and the probability that a term occurs in a document $P_T(t|d)$. These probabilities are estimated in the tuple space, which is indicated by the T subscript. The RSV is defined as follows:

$$\mathrm{RSV}_{\mathrm{LM}}(d, q) = \sum_{t \in q} \log\left(\lambda \cdot P_T(t|d) + (1-\lambda) \cdot P_T(t|c)\right)$$

The mixture parameter λ is to be set: It can be term-dependent, query-dependent, or background-dependent.

The following PSQL script is an implementation of LM:

```
-- PSQL: lm retrieval
-- Extensional relations:
-- Coll(Term, DocId);
-- Query(Term, QueryId);
-- tf_sum(term, context);
-- mixture(name);

-- mixture:
DELETE FROM mixture;
INSERT INTO mixture VALUES
0.8 ('p_t_d'), 0.2 ('p_t_c');

CREATE VIEW lambda1 AS
    SELECT FROM mixture
    WHERE mixture.name = 'p_t_d';
CREATE VIEW lambda2 AS
    SELECT FROM mixture
    WHERE mixture.name = 'p_t_c';

-- P(t|d):
-- Principle description via views:
CREATE VIEW tfCollSpace AS
    SELECT Term, DocId
    FROM Coll
    EVIDENCE KEY (DocId);
CREATE VIEW p_t_d AS
    SELECT DISJOINT Term, DocId
    FROM tfCollSpace;

-- For efficiency,
-- bind p_t_d to extensional instance.
CREATE VIEW p_t_d AS
    SELECT term AS Term, context AS DocId
    FROM tf_sum;

-- P(t|c):
CREATE VIEW p_t_c_evidence AS
    SELECT Term
    FROM Coll
    EVIDENCE KEY ();
CREATE VIEW p_t_c AS
    SELECT DISJOINT Term
    FROM p_t_c_evidence;

-- retrieved(DocId, QueryId):
-- Needed for generating schema-compatible
-- views docModel and collModel.
CREATE VIEW retrieved AS
    SELECT DISTINCT DocId, QueryId
    FROM Query, Coll
    WHERE Query.Term = Coll.Term;

CREATE VIEW docModel AS
    SELECT Term, DocId
    FROM lambda1, p_t_d;

CREATE VIEW collModel AS
    SELECT Term, DocId
    FROM lambda2, p_t_c, retrieved;

-- combine document and collection models
CREATE VIEW lm1_p_t__c_d AS
    docModel UNION DISJOINT collModel;

-- retrieve documents
CREATE VIEW lm1_retrieve AS
    SELECT SUM_LOG DocId, QueryId
    FROM Query, lm1_p_t__c_d
    WHERE Query.Term = lm1_p_t__c_d.Term;

-- Probabilistic interpretation:
-- P(t|c,d) = lambda1 P(t|d) + lambda2 P(t|c)
-- RSV(d,q) = prod_t P(q|t) P(t|c,d)

CREATE VIEW retrieve AS
    SELECT DocId, QueryId
    FROM lm1_retrieve;
```

The PSQL script shows the probabilistic views “p_t_d” and “p_d_d”, where the probabilities correspond to $P_T(t|d)$ and $P_T(t|c)$, respectively. Similar to the *tf-idf* script, we show the principle generation of $P_T(t|d)$, which we then overwrite by a view that takes advantage of a materialised relation “tf_sum” that contains the pre-computed probabilities. This is purely for reasons of efficiency, since the view “tf” requires an aggregation of probabilities, and this aggregation can be pre-computed in a materialised relation. Then, the join with “tf” is more efficient.

Consider next a PRA program equivalent to the outcome of the PSQL to PRA translation:

```
# PRA: lm retrieval
# Extensional relations:
# Coll(Term, DocId);
# Query(Term, QueryId);
# tf_sum(term, context);
# mixture(name);

# Mixture:
_delete(mixture);
0.8 mixture(p_t_d);
0.2 mixture(p_t_c);
lambda1 = Project[](Select[$1=p_t_d](mixture));
lambda2 = Project[](Select[$1=p_t_c](mixture));

# P(t|d): p_t_d(Term, DocId):
tfCollSpace = Bayes[$2](Coll);
p_t_d = Project disjoint[$1,$2](tfCollSpace);

# Optional usage of pre-computed tf:
p_t_d = tf_sum;
```

```
# P(t|c): p_t_c(Term):
collSpace = Bayes[](Project[$1](Coll));
p_t_c = Project disjoint[$1](collSpace);

# Retrieved documents for the generation of
# the collection model that can be united with
# the document model.
# retrieved(DocId):
retrieved =
    Project distinct[$4](
        Join[$1=$1](Query, Coll));

# Document model:
# docModel(Term, DocId):
docModel = Join[](lambda1, p_t_d);

# Collection model:
# collModel(Term, DocId):
collModel =
    Join[](lambda2, Join[](p_t_c, retrieved));

# Combination of docModel and collModel:
lm_term_weight =
    Unite disjoint(docModel, collModel);

# Retrieve documents:
lm_retrieve =
    Project sum_log[$4,$2](
        Join[$1=$1](Query, lm_term_weight));

retrieve = lm_retrieve;
```

The PSQL views correspond to their respective PRA equations. The view “collModel” involves an expensive join of query term weights based on $P(t|c)$ with the retrieved documents. This join is required since the relational union requires schema-compatible relations “docModel” and “collModel”.

The implementation shown above is semantically correct but because of the required schema compatibility not efficient. We have started to look into alternative PRA formulation, and we have defined an extended PRA with special mixture joins that support a correct and efficient implementation of LM. This is related to the description of Poisson-like estimates mentioned in the Sect. 7.1 on *tf-idf*. We will report on the PRA extensions regarding Poisson and probability mixtures in future work.

We have presented the PSQL/PRA implementation of language modelling. With this, we have completed the implementation of three main models, namely *tf-idf*, BIRM, and LM. For *tf-idf* and LM, we showed semantically correct implementations, whereas the BIRM implementation does not implement the genuine BIRM formulation. Proving the correctness of PSQL/PRA implementations is an important task; for the implementations shown here, the correctness has been investigated but the formal proofs have been excluded from this paper. Also, the PSQL/PRA scripts for *tf-idf*, BIRM and LM have been verified in a prototypical implementation. In the next section, we add the probabilistic relational modelling of precision/recall.

7.4 Precision/recall

Precision and recall are frequently used measures to compare retrieval quality. Precision and recall can be interpreted as the conditional probabilities $P(\text{relevant}|\text{retrieved})$ and $P(\text{retrieved}|\text{relevant})$, respectively. This interpretation implies that we can model precision and recall in a probabilistic relational framework that supports the description of conditional probabilities. This has two benefits: Firstly, the measures become part of a conceptual framework in which we model IR. Secondly, by replacing black-box tools that produce precision/recall values, we enable the application-specific modification of measures.

For an illustration, consider the following data in relations “Retrieved” and “Relevant”:

Retrieved	
QueryId	DocId
q1	doc2
q1	doc4
q1	doc6
q1	doc8
q1	doc1
q1	doc3
q1	doc5
q1	doc7
q1	doc9
q2	doc5
q2	doc4

Relevant	
QueryId	DocId
q1	doc1
q1	doc4
q1	doc9
q1	doc11
q1	doc14
q1	doc19
q2	doc4

Based on these extensional relations, we define three views that are later used for defining precision and recall.

```
-- PSQL
-- Extensional relations:
-- Retrieved(QueryId, DocId);
-- Relevant(QueryId, DocId);

CREATE VIEW retrievedSpace AS
    SELECT QueryId, DocId
    FROM Retrieved
    ASSUMPTION DISJOINT
    EVIDENCE KEY (QueryId);

CREATE VIEW relevantSpace AS
    SELECT QueryId, DocId
    FROM Relevant
    ASSUMPTION DISJOINT
    EVIDENCE KEY (QueryId);

CREATE VIEW retrieved_and_relevant AS
    SELECT QueryId, DocId
    FROM Relevant, Retrieved
    WHERE Relevant.QueryId = Retrieved.QueryId
    AND Relevant.DocId = Retrieved.DocId;
```

The view “retrievedSpace” contains for each query the probabilistic tuples that reflect the probability that a document is among the retrieved documents of the query. The view “relevantSpace” has the analogous role for the relevant documents. Given these spaces and the view "retrieved_and_relevant", we describe precision and recall:

```
-- PSQL: precision and recall

CREATE VIEW precision AS
  SELECT DISJOINT query
  FROM retrieved_and_relevant, retrievedSpace
  WHERE retrieved_and_relevant.QueryId =
       retrievedSpace.QueryId
  AND retrieved_and_relevant.DocId =
       retrievedSpace.DocId;

CREATE VIEW recall AS
  SELECT DISJOINT query
  FROM retrieved_and_relevant, relevantSpace
  WHERE retrieved_and_relevant.QueryId =
       relevantSpace.QueryId
  AND retrieved_and_relevant.DocId =
       relevantSpace.DocId;
```

The translation of the first PSQL script with views "retrievedSpace" and "relevantSpace" yields the following PRA program:

```
# PRA
retrievedSpace = Bayes[$1](Retrieved);
relevantSpace = Bayes[$1](Relevant);

retrieved_and_relevant =
  Project[$1,$2](
   Join[$1=$1,$2=$2](
    Relevant, Retrieved));
```

The first two equations yield the two spaces "retrievedSpace" and "relevantSpace", where in each space a document occurs with the probability $P_{\text{space}}(d|q) = 1/N(q)$, where $N(q)$ is the number of documents for query q.

The third equation yields the relation of retrieved and relevant documents.

We obtain the following probabilistic relations:

retrievedSpace		
Prob	QueryId	DocId
1/9	q1	doc2
1/9	q1	doc4
1/9	q1	doc6
1/9	q1	doc8
1/9	q1	doc1
1/9	q1	doc3
1/9	q1	doc5
1/9	q1	doc7
1/9	q1	doc9
1/2	q2	doc5
1/2	q2	doc4

relevantSpace		
Prob	QueryId	DocId
1/6	q1	doc1
1/6	q1	doc4
1/6	q1	doc9
1/6	q1	doc11
1/6	q1	doc14
1/6	q1	doc19
1	q2	doc4

retrieved_and_relevant		
Prob	QueryId	DocId
1	q1	doc1
1	q1	doc4
1	q1	doc9
1	q2	doc4

Next, consider the PRA equations for precision and recall:

```
# PRA: precision and recall

precision =
  Project disjoint[$1](
   Join[$1=$1,$2=$2](
     retrieved_and_relevant, retrievedSpace));

recall =
  Project disjoint[$1](
   Join[$1=$1,$2=$2](
     retrieved_and_relevant, relevantSpace));
```

The joins of "retrieved_and_relevant" with the respective spaces, followed by disjoint projections, yield the precision and recall values:

precision	
Prob	QueryId
3/9	q1
1/2	q2

recall	
Prob	QueryId
3/6	q1
1	q2

We have demonstrated how PSQL/PRA enables to express precision and recall. This result embeds both, retrieval models and quality measures, into the conceptual framework of probabilistic relational modelling. The expressiveness of relational modelling allows to customise the measures. For example, to capture the dependency of tuples (documents) in relation "Retrieved", we would join "Retrieved" with a relation "Dependency(DocId1, DocId2)" to perform a post-processing of the retrieval result, and to base a measure on the obtained alternative of retrieved documents.

The modelling of precision and recall completes the probabilistic relational modelling of main IR concepts. In the next section, we evaluate PSQL/PRA against the modelling of IR models and probability estimation in standard SQL.

8 Evaluation

In this section we compare the following:

- The modelling of *tf-idf* retrieval using traditional SQL versus PSQL. The comparison highlights the abstraction and expressiveness of each approach.
- The efficiency and scalability of modelling *tf-idf* retrieval using traditional SQL versus PSQL. The comparison focuses on investigating the performance of each approach for handling large-scale data.
- The scalability of estimating probabilities using SQL versus PSQL. The investigation focuses on the performance of generating probabilities in large-scale databases.

We first investigate in Sect. 8.1 the implementation of *tf-idf* in both traditional SQL and PSQL. Then, we discuss probability estimation in Sect. 8.2.

Coll		
Attribute	*Type*	*Index*
Term	varchar	non-clustered
DocId	varchar	none

CollStats		
Attribute	*Type*	*Index*
NumOfDocs	int	none

DocSpace (DocumentSpace)		
Attribute	*Type*	*Index*
DocId	varchar	clustered
Length	int	none

TermFreq		
Attribute	*Type*	*Index*
Term	varchar	non-clustered
DocId	varchar	none
P_t_d	float	none

TermSpace		
Attribute	*Type*	*Index*
Term	varchar	clustered
DocFreq (DF)	int	none

TermSpaceDF		
Attribute	*Type*	*Index*
Term	varchar	non-clustered
P_t_c	float	

QTerms		
Attribute	*Type*	*Index*
Term	varchar	non-clustered

Fig. 13 SQL database schema for SQL-based modelling of text retrieval

According to their different natures, we refer to the implementation of modelling IR by traditional SQL as "IR on DB", and we refer to the PSQL approach as "DB + IR". We demonstrate our implementations and discuss our analysis in the following sections.

8.1 TF-IDF-based retrieval: SQL versus PSQL

As we presented in Sect. 4, *tf-idf*-based retrieval can be denoted using the probability $P(t|d)$ that term t occurs in document d, and the probability $P(t|c)$ that term t occurs in a document of collection c.

$$\mathrm{RSV}(d,q) = \sum_t P(t|d) \cdot -\log P(t|c)$$

We take the "Coll" relation from the running example in Sect. 3 to demonstrate our implementations.

8.1.1 TF-IDF using traditional SQL

Figure 13 shows the database schema applied for the tf-idf implementation with SQL.

We start with the SQL view named "CollStats" containing collection-wide statistics, for example, the number of documents.

```
CREATE VIEW CollStats AS
SELECT count(DISTINCT DocId) AS NumOfDocs
FROM Coll;
```

We obtain:

CollStats
NumOfDocs
5

We define "CollStats", and also the relations to follow, as views, since the idea is that all these relations are based on the persistent relation "Coll(Term, DocId)", in which we model the representation of the collection. In an ideal scenario, updates on the possibly huge relation "Coll" update automatically the views in which the statistics are maintained (Fig. 13).

Next, we create a table of documents (relation "DocSpace") where we store for each document the document length.

```
CREATE VIEW DocSpace AS
  SELECT DocId, count(Term) AS Length
  FROM Coll
    GROUP BY DocId;
```

We obtain:

DocSpace	
DocId	Length
doc1	2
doc2	3
doc3	3
doc4	1
doc5	1

Now, we are ready to compute the probabilities $P(t|d)$ and $P(t|c)$. However, we treat their modelling differently. In the case of $P(t|d)$ we compute the final probability because we assume that there will be no partial update of the document. On the other hand, we delay the computation of $P(t|c)$ in order to be prepared for updating the collection incrementally.

Consider next the creation of view "TermFreq"(Term, Document, P_t_d):

```
CREATE VIEW TermFreq AS
  SELECT Term, Coll.DocId,
    count(Term)/DocSpace.Length AS P_t_d
  FROM DocSpace, Coll
  WHERE Coll.DocId = DocSpace.DocId
    GROUP BY Coll.DocId, Term;
```

We obtain:

TermFreq		
Term	DocId	P_t_d
sailing	doc1	1/2
boats	doc1	1/2
sailing	doc2	2/3
boats	doc2	1/3
sailing	doc3	1/3
east	doc3	1/3
coast	doc3	1/3
sailing	doc4	1.0
boats	doc5	1.0

Coll	
Attribute	*Type*
Term	varchar
DocId	varchar

TermFreq	
Attribute	*Type*
Term	varchar
DocId	varchar

TermSpaceDF	
Attribute	*Type*
Term	varchar

TermSpaceIDF	
Attribute	*Type*
Term	varchar

Fig. 14 PSQL database schema for modelling text retrieval

We have modelled the so-called normalised within-document term frequency. Next, we create the table "TermSpace" where we maintain for each term the number of documents (the so-called document frequency) in which the term occurs.

```
CREATE VIEW TermSpace AS
  SELECT Term, count(DISTINCT DocId) AS DF
  FROM Coll
  GROUP BY Term;
```

We obtain:

TermSpace	
Term	DF (DocFreq)
sailing	4
boats	3
east	1
coast	1

The explicit "TermSpace" is not a common practice. In [31], a static model is proposed where idf is directly computed. This is shown in the following SQL statement:

```
CREATE VIEW idf AS
  SELECT Term,
    -log(count(DISTINCT DocId)/
    CollStats.NumOfDocs)
  FROM CollStats, Coll
    GROUP BY Term;
```

The above view "idf" is problematic for update operations. If we add a document to the collection, then the tuples in the view need to be updated because CollStats.NumOfDocs has changed. Therefore, we model in "TermSpace" the "DocFreq" as the total count and apply the logarithm and perform normalisation (with respect to the total number of documents) at retrieval time. Consequently, this incremental nature is different from the static approach described in [31].

Now, we are ready to describe *tf-idf*-based retrieval. The probabilities based on document frequency can be computed from "TermSpace" as follows:

```
CREATE VIEW TermSpaceDF AS
  SELECT Term,
    TermSpace.DF/CollStats.NumOfDocs AS P_t_c
  FROM TermSpace, CollStats;
```

We obtain:

TermSpaceDF	
Term	P_t_c
sailing	4/5
boats	3/5
east	1/5
coast	1/5

The delayed application of the logarithm keeps our model tidy, since now we have in "TermFreq" and in "TermSpaceDF" probabilistic weights with a clear semantics, namely $P(t|d)$ and $P(t|c)$.

Finally, we describe *tf-idf*-based retrieval as the aggregations of query terms with document frequencies and within-document term frequencies.

```
SELECT sum(P_t_d * -log(P_t_c)), DocId
  FROM QTerms, TermSpaceDF, TermFreq
  WHERE QTerms.Term = TermSpaceDF.Term
    AND TermSpaceDF.Term = TermFreq.Term
    GROUP BY DocId;
```

8.1.2 TF-IDF using PSQL

Figure 14 shows the probabilistic database schema we use for modelling document retrieval. First, we describe the probability $P(t|d)$, i.e., the probability that term t occurs in document d.

```
CREATE VIEW TermFreq AS
  SELECT DISJOINT Term, DocId
  FROM Coll
  EVIDENCE KEY (DocId);
```

We obtain:

TermFreq		
$P(\tau)$	Term	DocId
1/2	sailing	doc1
1/2	boats	doc1
2/3	sailing	doc2
1/3	boats	doc2
1/3	sailing	doc3
1/3	east	doc3
1/3	coast	doc3
1.0	sailing	doc4
1.0	boats	doc5

Next, consider the creation of a term space in which the term probabilities reflect $P(t|c)$, i.e., the probability that t occurs in (a document of) c.

```
CREATE VIEW distinctTerms AS
  SELECT DISTINCT Term, DocId
  FROM Coll;

CREATE VIEW TermSpaceDF AS
  SELECT Term
  FROM distinctTerms
  EVIDENCE KEY ();
```

We obtain:

TermSpaceDF	
$P(\tau)$	Term
4/5	sailing
3/5	boats
1/5	east
1/5	coast

Next, we apply an advanced feature of PSQL, namely so-called informativeness-based probability estimations. Through this operation, we obtain a term space in which the probabilities reflect the informativeness of terms.

```
CREATE VIEW TermSpaceIDF AS
  SELECT Term
  FROM TermSpaceDF
  ASSUMPTION MAX_LOG
  EVIDENCE KEY ();
```

The assumption max_log inverts the occurrence probabilities in "TermSpaceDF", assigning high probabilities to rare terms, and low probabilities to frequent terms.

Finally, retrieval is described as a join of weighted query terms and the relation "TermFreq".

```
SELECT DISTINCT DocId
  FROM QTerms, TermSpaceIDF, TermFreq
  WHERE QTerms.Term = TermSpaceIDF.Term
    AND QTerms.Term = TermFreq.Term;
```

We have modelled *tf-idf*-based retrieval in PSQL. While in traditional SQL aggregation operators were necessary, in PSQL we worked on a conceptual probabilistic layer and defined evidence keys and probabilistic assumptions. Thus, it becomes feasible to apply IR concepts to any relational database.

8.1.3 Comparison of efficiency and scalability

To evaluate efficiency and scalability, we used two systems since there no existing system can process both SQL and PSQL.

The first system, referred to anonymously as A, is a well-known open source multi-thread database and the second one, referred to as B, is our generic DB + IR prototype HySpirit ([49]). On both systems, we implemented a *tf-idf* text retrieval application. In other words, using candidate A we implemented the ranking function by mapping the IR models onto standard SQL (IR-on-DB), while using candidate B we implemented the ranking model in PSQL (DB + IR). The implementation details of both systems were described in Sects. 8.1.1 and 8.1.2, respectively.

It is important to emphasise that the aim of the comparison is to show the flexibility and scalability of the two different approaches (IR-on-DB and DB + IR). We are not comparing the actual systems. Although the experimental environments are not entirely the same, for the purpose of demonstrating the flexibility and scalability this setup is sufficient.

The experiments were run on a Linux server (Fedora core 2) that is equipped with one Intel Pentium4 2.60 GHz CPU and 2 GB memory. The testing data was produced from the enterprise track of TREC 2005 data [http://trec.nist.gov/], and the original text size is 1.9 GB. After transformation there are about 40 million tuples in the table "Coll", which in system A uses 1.3 GB with a 416 MB B-tree index. In system B, it uses 1.0 GB for the table and 1.3 GB for a Hash-based index. Thus, the initial data size of system A is 1.7 GB, and system B is 2.3 GB. After pre-processing, intermediate tables (which can be considered as materialised views) and corresponding indexes were built. In total, the database size of system A is 3.0 GB, and system B is 4.4 GB. The database sizes are different because they use different storage data structures and different indexing mechanisms.

For evaluating the pre-processing cost, we measured the processing time for 10, 20, 30, and 40 (scaling points $1\times, 2\times, 3\times$ and $4\times$) million tuples. For evaluating the retrieval time, we measured the performance of various queries, where the queries vary with respect to the number of tuples they retrieve (so-called selectivity of the query). A query with higher selectivity returns less tuples. We measured processing time for $10, 25, \ldots,$ and 100 (scaling points $1\times, 2.5\times, \ldots, 10\times$) thousand tuples returned.

In Sect. 8.1.1, we discussed using views for the intermediate relations. In real-world IR applications, the update operation is not as frequent as in transaction-oriented databases. Therefore, we can store the intermediate outputs to extensional tables, while an alternative is to use materialised views providing that the database back-end supports them. As a result, the pre-processing time includes the construction time of the intermediate relations (materialised view) and corresponding indexes.

Figure 15 shows the performance of pre-processing. The indexing processes were executed in batch mode, and 10–40 million tuples were loaded.

Figure 15a shows the database construction time, where the processing steps follow the sequence described in Sect. 8.1.1. First, the source data was loaded to the "Coll" table, and the B-tree index was built. When the data was loaded, the loading and indexing time for 10 million tuples is about 4,600 s, and for 40 million tuples is about 6,800 s. As soon as the data was loaded, we computed the statistic of "Coll" and stored it in "CollStats". Because the index was available, the statistic was calculated instantly. In the third step, we generated the "DocSpace" table, which contains the document Ids against their document lengths. The curve shows that the time increases in proportion with the number of tuples. Forth, we generated the term frequency in "TermFreq". The curve of processing time seems proportional while less than 3 million tuples were loaded, but the

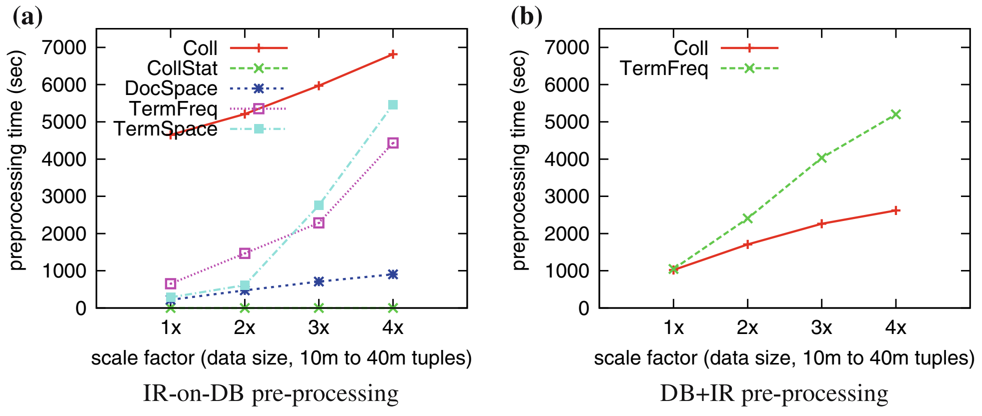

Fig. 15 Analysis of pre-processing costs: IR-on-DB and DB+IR

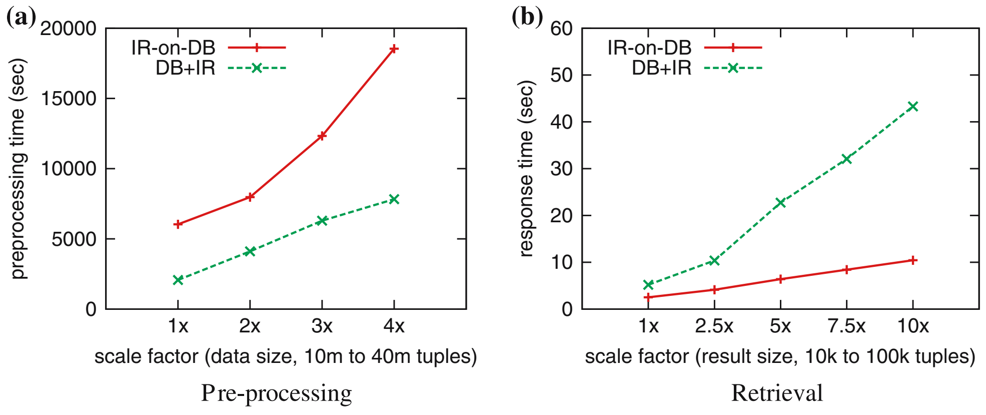

Fig. 16 Comparison of IR-on-DB and DB+IR: pre-processing time and retrieval time

time dramatically increased when we loaded 4 million tuples. The last step computed the document frequency that was in "TermSpace". The processing time increased sharply after 2 million tuples, this is because to aggregate *df* needs to perform a distinct projection over the entire "Coll" table, and the aggregation time increased polynomially.

Figure 15b shows the DB+IR pre-processing time, where we needed to load and index the data and compute the term frequency. The other intermediate relations were generated on-the-fly. The curve shows the loading time plus indexing time, where it is about proportional to the data size. Based on the relation "Coll", term frequency were calculated and stored in "TermFreq", the step includes both computation and indexing. Because the document frequency can be obtained from the probabilistic index (i.e., index on "Coll(Term)"), its computation is saved, and no other pre-processing is needed.

Figure 16 gives the overall performance. To compare the total pre-processing time of the two approaches, we sum up the times of the pre-processing steps, and the result is shown in Fig. 16a. We find that the DB+IR needs less pre-processing time than IR-on-DB, and DB+IR pre-processing is linear to the data size, whereas IR-on-DB is of polynomial complexity. Figure 16b shows the retrieval time of both systems. The database system outperforms HySpirit, but we compare here a mature database product with established algebra optimisation and cache usage against our prototypical implementation of a probabilistic database.

To conclude, we emphasise that the IR-on-DB approach needs a long preparation phase to become ready to perform large-scale retrieval. For the DB + IR approach, the preparation cost are less than those of IR-on-DB. Therefore, the DB+IR approach is more scalable than the IR-on-DB approach regarding the estimation of probabilities over millions of tuples.

8.2 Probability estimation: SQL versus PSQL

In this section, we investigate the performance of probability estimations using SQL versus PSQL. The assumption is that with tailored indexes for probability estimation, we can be faster than traditional SQL in which we use aggregation functions to implement probability estimation.

We load millions of tuples to both systems, and then we retrieve tuple-based and value-based probabilities. For this

Person		
Attribute	*Type*	*Index*
Name	varchar	none
Nationality	varchar	non-cluster
City	varchar	none
Prob	float	none

Fig. 17 SQL database schema of "Person" relation

investigation, we use for SQL the traditional table "Person". The schema is shown in Fig. 17. The schema of the probabilistic table is similar, but without the explicit attribute "Prob".

8.2.1 Probability estimation based on tuple frequency

For the attribute "Nationality", there is an index. We want to estimate the tuple-based probability of "Nationality" given "City", i.e., we want to estimate the probability $P_{T,Person}(n|c)$, where n is a value of "Nationality", and c is a value of "City".

In SQL, we create a view called "nationalitySpace" for counting the total number of values in "Nationality" grouped by the values in "City". Then, we obtain the tuple-based probability by dividing the count of "Nationality" per "City" by the number of nationalities ("NumOfNa") per "City".

```
-- SQL
CREATE VIEW nationalitySpace AS
  SELECT City, count(Nationality) AS NumOfNa
  FROM Person
    GROUP BY City;

SELECT Nationality, Person.City,
  count(Nationality)/nationalitySpace.NumOfNa
  AS P_n_c
FROM nationalitySpace, Person
WHERE Person.City = nationalitySpace.City
  GROUP BY Person.City, Nationality;
```

In PSQL, we specify the probability aggregation and estimation instead of aggregation and mathematical functions (log). We create the view "nationalitySpace" by specifying the evidence key "City". Then, we aggregate the probabilities in a disjoint selection.

```
-- PSQL
CREATE VIEW nationalitySpace AS
  SELECT Nationality, City
  FROM Person
  EVIDENCE KEY (City);

SELECT DISJOINT Nationality, City
FROM nationalitySpace;
```

We have described the SQL-based and the PSQL-based implementation of a tuple-based probability. Next, we describe the implementation of probability estimations based on value frequencies.

8.2.2 Probability estimation based on value frequency

In this section, we present the implementations of value-based probability estimations. We use a similar configuration as for tuple-based probabilities. We calculate the probability of "Nationality" based on the number of values in "City" a nationality is associated with, i.e., we compute the probability $P_{V,Person[City]}(n)$, where n is a value from "Nationality".

First, we create a view "personStats" to compute the total number of distinct values in "City". Then, we group by "Nationality", count for each nationality the number of distinct values in "City" the nationality is associated with, and divide the count by the number of cities. The negative logarithm yields an *idf*-value for the values in "Nationality". The corresponding SQL statements are as follows:

```
-- SQL
CREATE VIEW personStats AS
  SELECT count(DISTINCT City) AS NumOfCity
  FROM Person;

SELECT Nationality,
  -log(count(DISTINCT City)/personStats.NumOfCity)
FROM Person, personStats
  GROUP BY Nationality;
```

In PSQL, the expression is more compact. We define an *idf*-based space over "Nationality" by specifying the assumption max_idf.

```
-- PSQL
SELECT Nationality
FROM Person
  ASSUMPTION MAX_IDF
  EVIDENCE KEY ();
```

Next, we report the performance results for tuple-based and value-based probability estimations.

8.2.3 Performance of probability estimation

For measuring the performance, we generate a relation "Person". We insert 3 million tuples (628 MB) to conduct the experiment. Both HySpirit and the other database system built indexes. The index size in HySpirit is 105 MB, and the index size in the database system is 37 MB. We use the same system configuration reported for the previous experiment in Sect. 8.1.3.

Both experiments were also performed in batch mode. We estimated the probabilities based on the whole data set, and we recorded the processing elapse time on the measurement points.

Figures 18 and 19 show the performance for tuple and value frequencies, respectively. Both experiments show that the PSQL processing outperforms the SQL processing. In particular, when estimating the tuple frequency, the pre-processing of the database system did not finish even

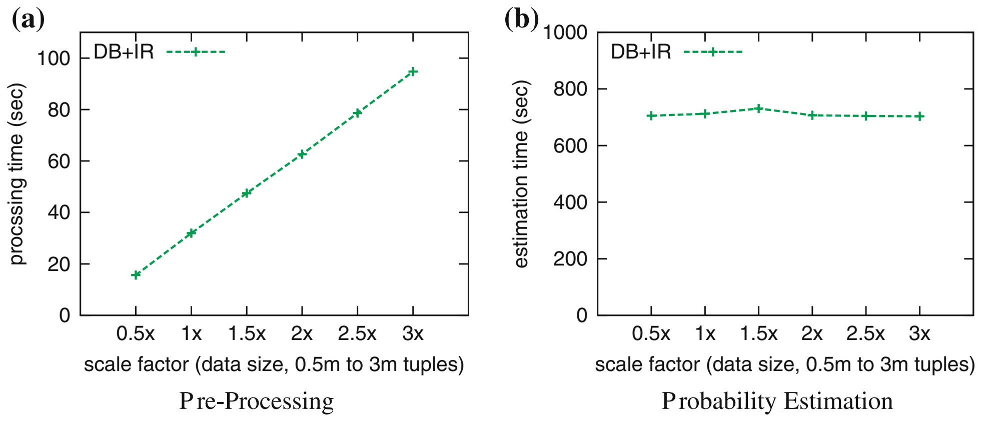

Fig. 18 Probability estimation based on tuple frequency: pre-processing time and estimation run time (IR-on-DB not shown due to scalability problem)

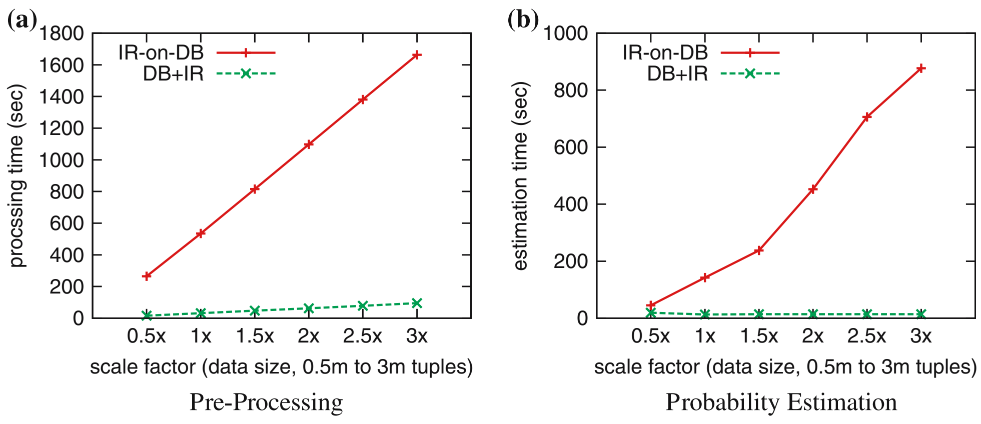

Fig. 19 Probability estimation based on value frequency: pre-processing time and estimation run time

after 14 h, even though proper indexes had been used. When tracing the reason for this, we found that the selectivity of "Nationality" is very high, i.e., the number of distinct values in "Nationality" is relatively low. In contrast, the selectivity of the attribute "City" is very low, i.e., the number of values in "City" is higher than in "Nationality". Therefore, the traditional approach seemed to struggle grouping by "City", and then counting the number of values in "Nationality" per "City".

Because in HySpirit, frequencies are maintained in indexes, the probabilities can be derived directly. The standard indexing mechanism in the database system does not provide a similar functionality, and the probability estimation involves expensive aggregations. Therefore, the probability estimations were included in the pre-processing stage in the experiment discussed in Sect. 8.1.

9 Discussion and outlook

In this section, we present discussion points and outlook in the form of frequently asked questions, which occurred in various contexts like talks, reviews, discussions, and when applying the technology.

You view the aggregation of probabilities in traditional SQL as physical. What does this mean? If we model probabilistic tuples in traditional SQL, then probabilities are treated equal to normal attribute values such as Name, Price, Nationality, etc. The SQL programmer implements the ranking strategy using SQL aggregation functions such as sum, max, and arithmetic functions such as log. In PSQL, however, we view probabilities and attribute values to be *orthogonal*, i.e., the PSQL programmer has no direct access to probabilities. Instead, the algebra operation defines the aggregation. Therefore, PSQL is a logical layer, whereas traditional SQL is physical in this sense, since the probability arithmetics are described in SQL.

What does "probabilities and attribute values are orthogonal" mean? This aspect is excellently covered in [54]. When aggregating attribute values in a probabilistic relational framework, the aggregation is related to computing the expectation value. For example, consider a probability distribution over prices. When we ask for the expected price, then the expected price is defined as $E[\text{price}] = \sum P(\text{price}) \cdot \text{price}$. This illustrates that there is an implicit usage of probabilities in a PRA, whereas the "normal" attribute values are explicitly mentioned in the PSQL query.

Tuple weights greater than one or less than zero might occur when using a disjointness assumption. Is such a relational model probabilistic? The pure algebra has no safety net for expressions where assumptions are—from a probabilistic semantics point of view—wrongly specified (see [17] for the notion of safe expressions, and [24] for intensional semantics). One topic of future research is to map a probabilistic model automatically to PSQL/PRA, and, vice versa, how to re-engineer the probabilistic model from a PSQL/PRA script.

Why is the division not used for modelling probability estimation? The division is equivalent to an algebra expression composed of projection, join, and subtraction (see Sect. 5.2).

Is the CONTAINS predicate in SQL not sufficient for IR tasks? Yes and no. Yes, if we are happy with modelling documents as atomic attribute values, which means that there are no or only very restricted means of looking inside the document. No, if we want to apply the expressive power of the relational model for representing the knowledge contained in a document. We propose to model the content of documents in a relational schema. The simplest schema is "Coll(Term,DocId)", but it is the particular strength of the relational model to represent any information, e.g. links, types of links, structure of documents, objects that occur in document, and the relationships between objects.

You load large relations to database systems, and then you complain that there are problems with scalability. You should use the inverted list! We load document representations to database systems, so that we can reason/search across the structured and unstructured data. By representing document content in the structured data model world, we gain a high level of integration. For example, join "Person.Nationality" with a document representation and thus retrieve documents or document parts that mention the nationality of a particular group of persons. The opposite direction, namely to export structured data into the unstructured world for doing retrieval is a principle alternative. However, we favour to preserve the semantics and structure of data.

What are the next challenges? Our work programme includes: (1) relevance-based processing of traditional SQL, (2) design and correctness of probabilistic logical programs, (3) expressiveness, (4) scalability and optimisation, (5) special predicates, and (6) interfaces and languages.

1. *Relevance-based processing of SQL*: The idea is to convert traditional SQL statements automatically into PSQL statements in which a ranking strategy is reflected. Then, all existing (traditional) SQL queries may yield a relevance-sorted result. Relevance-based SQL could be viewed as the external layer in Fig. 2.
2. *Design and correctness of PSQL/PRA programs*: For a PSQL/PRA program, we encountered in many contexts the need to derive the probabilistic semantics, so that the knowledge engineer (the person who works in PSQL/PRA) can verify his/her scripting. For this, we have developed a proof methodology which has been part of an earlier version of this paper, but will be reported separately.
3. *Expressiveness*: The expressiveness of PSQL/PRA allows for the modelling of not only retrieval models, but also evaluation measures such as precision/recall. Next steps include to incorporate average precision, precision@10, reciprocal rank, etc. In general, this is the field of increasing the expressiveness where we take, like for the relational Bayes, a careful and conceptual approach, trying to improve expressiveness but not overloading an otherwise tidy paradigm with special operators and functions.
4. *Scalability and optimisation*: Stream-based processing (see [48] and top-k processing ([22,62]) are key to scalable and efficient retrieval in large-scale applications. We covered the DB approaches (known as top-k or RankSQL) in the background, but we have said little about how this applies to PSQL. We are working on soft-sorting algebra operators, that would guarantee real-time response times while risking that the ranking is sub-optimal.
5. *Special predicates*: The previous aspect is not to be confused with stream-based predicates. Stream-based predicates allow to compare tuple values of subsequent tuples. For example, in the stream of term-document pairs, we would like to be able to find the documents where the terms sailing and boats appear near to each other. Another family of special predicates are the relevance-based predicates. We denote a new relevance-based implication predicate as "$\rightarrow$", borrowing the notation from [63] where the concept of relevance-based implication was proposed. We generalised the document-implies-query approach, and the new relevance-based predicate can be applied to any two attributes in a relational condition.
6. *Interfaces and languages*: PSQL/PRA might appeal to some, but others will prefer interfaces they feel comfortable with. Whether it is Datalog, description logic dialects, XML-based languages, or SQL dialects for assisting RDF retrieval, the likes are many. Our approach is basically to investigate the evaluation of such languages by translating them to PSQL/PRA. In a recent study, we mapped SPARQL queries ([4]), in the past we have mapped POOL (probabilistic object-oriented logic, [52,40,39]), where POOL triggered recently POLAR (probabilistic object-oriented logic for annotation-based retrieval, [21]), and POLIS (probabilistic object-oriented logic for information summarisation, [25]), which are highly abstract and tailored languages to assist the comfortable modelling of specific retrieval tasks.

10 Summary and Conclusions

This paper presented a probabilistic variant of SQL in which we describe probability *aggregation* (Sect. 5) and *estimation* (Sect. 6). It is one of the main contributions to describe both, aggregation *and* estimation, within the coherent framework of a probabilistic relational algebra. Since neither the standard five basic operators, nor division, nor attribute value aggregation are suitable for probability estimation, we required and developed a new probabilistic operator: The relational Bayes.

The other main contribution of this paper is the probabilistic relational modelling, i.e., a relatively abstract modelling, of retrieval models (Sect. 7). We have demonstrated the modelling of different *tf-idf* variants, and the modelling of the two main probabilistic retrieval models, binary independent retrieval model and language modelling. Also, we modelled precision/recall. This allows for describing task-specific measures, as required for example, for structured document retrieval.

The modelling of retrieval models in a probabilistic relational framework is desirable and useful, since it supports the development of ranking strategies beyond classical document retrieval. Also, since we represent classical document retrieval in a relational model, we gain the expressive power of the relational model to reason across structured and unstructured data. For example, we can join attributes such as "Person (Nationality)" with a text representation, to retrieve documents that are related to selected nationalities. And so forth. For all queries, we can define probabilistic interpretations of relations that meet the requirements of customised ranking strategies.

We presented in this paper PRA and PSQL as syntactical layers; we have developed further interfaces such as probabilistic Datalog variants and terminological logic variants, not reported in this paper. An early version of this paper included a theoretical evaluation (correctness proofs) of the implementations of retrieval models. These proofs will be reported in a separate publication.

The conceptual and experimental evaluation of PSQL/PRA is threefold. First, we demonstrated how we can express different retrieval models and their variants. Second, we compared the modelling of tf-idf in traditional SQL versus PSQL, on the one hand with respect to abstraction, and on the other hand with respect to performance (scalability/efficiency). Third, we investigated the performance and suitability of SQL versus PSQL for estimating probabilities. The main finding of the evaluation is that PSQL scales better than SQL for probability estimation.

With this paper, we contribute a coherent probabilistic logical layer to DB technology. The technology has been applied in domains such as financial news mining and expert finding, and is planned to be applied for crime prevention and detection. The probabilistic relational layer is capable of modelling advanced retrieval strategies, and, is in general suitable for the management of uncertainty and the uncertain reasoning in large-scale applications.

Acknowledgements We would like to acknowledge the excellent and deep reviews that helped improving the earlier version of this paper. We thank Mounia Lalmas for her thorough editorial pass, and Ingo Frommholz for his technical improvements and tests. The relational Bayes has been promoted and funded through the contributions of CLCUC (Combined London College University Challenge seed fund) and the technology transfer department of Queen Mary University of London.

References

1. Abiteboul, S., Agrawal, R., Bernstein, P., Carey, M., Ceri, S., Croft, B., DeWitt, D., Franklin, M., Garcia-Molina, H., Gawlick, D., Gray, J., Haas, L., Halevy, A., Hellerstein, J., Ioannidis, Y., Kersten, M., Pazzani, M., Lesk, M., Maier, D., Naughton, J., Schek, H., Sellis, T., Silberschatz, A., Stonebraker, M., Snodgrass, R., Ullman, J., Weikum, G., Widom, J., Zdonik, S.: The lowell database research self assessment (2003)
2. Agrawal, S., Chaudhuri, S., Das, G.: Dbxplorer: a system for keyword-based search over relational databases. In: ICDE, pp. 5–16 (2002)
3. Agrawal, S., Chaudhuri, S., Das, G., Gionis, A.: Automated ranking of database query results. In: CIDR (2003)
4. Azzam, H., Roelleke, T.: Efficient processing of ontological queries. In: 2nd VLDB Workshop on Ontologies-based Techniques for Databases and Information Systems, Seoul (2006)
5. Amati, G., van Rijsbergen, C.J.: Probabilistic models of information retrieval based on measuring the divergence from randomness. ACM Trans. Inf. Syst. **20**(4), 357–389 (2002)
6. Bosc, P., Galibourg, M., Hamon, G.: Fuzzy querying with sql: extensions and implementation aspects. Fuzzy Sets Syst. **28**(3), 333–349 (1988)
7. Barbara, D., Garcia-Molina, H., Porter, D. : A probabilistic relational data model. In: Bancilhon, F., Thanos, C., Tsichrizis, D. (eds.) Advances in Database Technology – EDBT '90., pp. 60–74. Springer, Berlin (1990)
8. Barbara, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Trans. Knowl. Data Eng. **4**(5), 487–502 (1992)
9. Berger, A., Lafferty, J.: Information retrieval as statistical translation. In: SIGIR (ed.) SIGIR '99 Proceedings of the 22nd International Conference on Research and Development in Information Retrieval, pp. 222–229, ACM, New York (1999)
10. Bosc, P., Pivert, O.: Fuzzy queries and relational databases. In: Proceedings of the 1994 ACM Symposium on Applied Computing, pp. 170–174 ACM Press, New York (1994)
11. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic ranking of database query results. In: VLDB pp. 888–899 (2004)
12. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst. **31**(3), 1134–1168 (2006)
13. Croft, W.B., Harper, D.J.: Using probabilistic models of document retrieval without relevance information. J. Doc. **35**, 285–295 (1979)
14. Cavallo, R., Pittarelli, M.: The theory of probabilistic databases. In: Proceedings of the 13th International Conference on Very Large Databases, pp. 71–81 Morgan Kaufman, Los Altos (1987)

15. Chaudhuri, S., Ramakrishnan, R., Weikum, G.: Integrating db and ir technologies: What is the sound of one hand clapping? In: CIDR, pp. 1–12 (2005)
16. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB, pp. 864–875 (2004)
17. Dalvi, N.N., Suciu, D.: Answering queries from statistics and probabilistic views. In: VLDB, pp. 805–816 (2005)
18. de Vries, A., Roelleke, T.: Relevance information: a loss of entropy but a gain for idf? In: ACM SIGIR, Salvador, Brazil (2005)
19. Ercegovac, V., DeWitt, D.J., Ramakrishnan, R.: The texture benchmark: Measuring performance of text queries on a relational dbms. In: VLDB, pp. 313–324 (2005)
20. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison-Wesley, Reading (2000)
21. Frommholz, I., Fuhr, N.: Probabilistic, object-oriented logics for annotation-based retrieval in digital libraries. In: Marchionini, G., Nelson, M.L., Marshall, C.C. (eds.) JCDL., pp. 55–64. ACM, New York (2006)
22. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
23. Fuhr, N., Roelleke, T.: A probabilistic NF2 relational algebra for integrated information retrieval and database systems. In: Tanik, M.M., Bastani, F.B., Gibson, D., Fielding, P.J. (eds.) Proceedings of the 2nd World Conference on Integrated Design and Process Technology (IDPT), Society for Design and Process Science (SDPS), Austin, pp. 17–30 (1996)
24. Fuhr, N., Rölleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Trans. Info. Syst. **14**(1), 32–66 (1997)
25. Forst, J.F., Tombros, A., Roelleke, T.: solving the enterprise trec task with probabilistic data models. In: Proceedings of TREC 2006, pp. xx–yy (2006)
26. Fuhr, N.: A probabilistic framework for vague queries and imprecise information in databases. In: McLeod, D., Sacks-Davis, R., Schek, H. (eds.) Proceedings of the 16th International Conference on Very Large Databases., pp. 696–707. Morgan Kaufman, Los Altos (1990)
27. Fuhr, N.: Probabilistic datalog—a logic for powerful retrieval methods. In: Fox, E.A., Ingwersen, P., Fidel, R. (eds.) Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval., pp. 282–290. ACM, New York (1995)
28. Grabs, T., Böhm, K., Schek, H.-J.: Powerdb-ir - information retrieval on top of a database cluster. In: CIKM, pp. 411–418 (2001)
29. Grabs, T., Böhm, K., Schek, H.-J.: Powerdb-ir - scalable information retrieval and storage with a cluster of databases. Knowl. Inf. Syst. **6**(4), 465–505 (2004)
30. Grossman, D.A., Frieder, O.: Information Retrieval: Algorithms and Heuristics. Kluwer, Massachusetts (1998)
31. Grossman, D.A., Frieder, O.: Information Retrieval. Algorithms and Heuristics, 2nd edn. vol. 15 of The Information Retrieval Series. Springer, Berlin Heidelberg (2004)
32. Grossman, D.A., Frieder, O., Holmes, D.O., Roberts, D.C.: Integrating structured data and text: a relational approach. JASIS **48**(2), 122–132 (1997)
33. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: VLDB, pp. 850–861 (2003)
34. Hiemstra, D.: A probabilistic justification for using tf.idf term weighting in information retrieval. Int. J. Digit. Libr. **3**(2), 131–139 (2000)
35. Hristidis, V., Papakonstantinou, Y.: Discover: keyword search in relational databases. In: VLDB, pp. 670–681 (2002)
36. Li, C., Chang, K.C.-C., Ilyas, I.F., Song, S.: Ranksql: Query algebra and optimization for relational top-k queries. In: SIGMOD Conference, pp. 131–142 (2005)
37. Lee, S.K.: An extended relational database model for uncertain and imprecise information. In: Proceedings of the 18th VLDB Conference, pp. 211–220 Morgan Kaufman, Los Altos (1992)
38. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: Probview: a flexible probabilistic database system. ACM Trans. Database Syst. **22**(3), 419–469 (1997)
39. Lalmas, M., Roelleke, T.: Modelling vague content and structure querying in XML retrieval with a probabilistic object-relational framework. In: Proceedings of the 6th International Conference on Flexible Query Answering Systems (FQAS), LNCS, Lyon, Springer, Berlin (2004)
40. Lalmas, M., Roelleke, T., Fuhr, N.: Intelligent hypermedia retrieval. In: Szczepaniak, P.S., Segovia, F., Zadeh, L.A. (eds.) Intelligent Exploration of the Web, Springer, Heidelberg (2002)
41. Lafferty, J., Zhai, C.: Probabilistic Relevance Models Based on Document and Query Generation, chap. 1. Kluwer (2002)
42. Macleod, I.A.: Text retrieval and the relational model. J. Am. Soc. Inf. Sci. **42**(3), 155–165 (1991)
43. Maron, M.E., Kuhns, J.L.: On relevance, probabilistic indexing, and information retrieval. J. ACM **7**, 216–244 (1960)
44. Motro, A.: Vague: a user interface to relational databases that permits vague queries. ACM Trans. Off. Inf. Syst. **6**(3), 187–214 (1988)
45. Motro, A.: Accommodating imprecision in database systems: Issues and solutions. Sigmod Rec. **19**(4), 69 (1990)
46. Niemi, T., Järvelin, K.: A straightforward NF2 relational interface with applications in information retrieval. Inf. Process. Manage. **31**(2), 215–231 (1995)
47. Ponte, J.M., Croft, W.B.: A language modeling approach to information retrieval. In: Bruce Croft, W., Moffat, A., van Rijsbergen, C.J., Wilkinson, R., Zobel, J. (eds.) Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 275–281. ACM, New York (1998)
48. Pfeifer, U., Fuhr, N.: Efficient processing of vague queries using a data stream approach. In: Proceedings of the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 189–198, New York (1995)
49. Roelleke, T., Lübeck, R., Kazai, G.: The HySpirit retrieval platform, demonstration. In: Croft, B., Harper, D.J., Kraft, D.H., Zobel, J. (eds.) Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New Orleans (2001)
50. Robertson, S.E.: Term frequency and term value. In: SIGIR, pp. 22–29 (1981)
51. Robertson, S.E.: Understanding inverse document frequency: On theoretical arguments for idf. J. Doc. **60**, 503–520 (2004)
52. Roelleke, T.: POOL: Probabilistic Object-Oriented Logical Representation and Retrieval of Complex Objects. Shaker Verlag, Aachen Dissertation (1999)
53. Roelleke, T.: A frequency-based and a Poisson-based probability of being informative. In: Callan, J., Cormarck, G., Clarke, C., Hawking, D., Smeaton, A. (eds.) Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Toronto, Canada, pp. 227–234 (2003)
54. Ross, J.G.R., Subrahmanian, V.S.: Probabilistic aggregates. In: 13th International Symposium on Methodologies for Intelligent Systems (ISMIS), Lyon, Foundations of Intelligent Systems. Springer, Heidelberg (2002)
55. Robertson, S.E., Sparck Jones, K.: Relevance weighting of search terms. J. Am. Soc. Inf. Sci. **27**, 129–146 (1976)
56. Roelleke, T., Wang, J.: A parallel derivation of probabilistic information retrieval models. In: ACM SIGIR (2006)
57. Robertson, S.E., Walker, S., Hancock-Beaulieu, M.M.: Large test collection experiments on an operational interactive system: Okapi at TREC. Inf. Process. Manage. **31**, 345–360 (1995)

58. Suciu, D., Dalvi, N.N.: Foundations of probabilistic answers to queries. In: SIGMOD Conference, p. 963 (2005)
59. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database Systems Concepts, 4th Edn. McGraw-Hill Higher Education (2002)
60. Schek, H.-J., Pistor, P.: Data structures for an integrated database management and information retrieval system. In: Proceedings of the 8th International Conference on Very Large Data Bases, pp. 197–207, Morgan Kaufman, Los Altos (1982)
61. Turtle, H., Croft, W.B.: Inference networks for document retrieval. In: Vidick, J.-L. (ed.) Proceedings of the 13th International Conference on Research and Development in Information Retrieval., pp. 1–24. ACM, New York (1990)
62. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation withprobabilistic guarantees. In: VLDB, pp. 648–659 (2004)
63. van Rijsbergen, C.J.: A non-classical logic for information retrieval. Comput. J. **29**(6), 481–485 (1986)
64. Wong, S.K.M., Yao, Y.Y.: On modeling information retrieval with probabilistic inference. ACM Trans. Inf. Sys. **13**(1), 38–68 (1995)
65. Yu, C.T., Lam, K., Salton, G.: Term weighting in information retrieval using the term precision model. J. ACM **29**(1), 152–170 (1982)
66. Zhai, C.X., Lafferty, J.D.: Two-stage language models for information retrieval. In: SIGIR, pp. 49–56 (2002)