

# QUIC: Handling Query Imprecision & Data Incompleteness in Autonomous Databases

Subbarao Kambhampati, Garrett Wolf, Yi Chen, Hemal Khatri  
Bhaumik Chokshi, Jianchun Fan, Ullas Nambiar  
Department of Computer Science and Engineering,  
Arizona State University, Tempe, AZ 85287, USA

## ABSTRACT

As more and more information from autonomous databases becomes available to lay users, query processing over these databases must adapt to deal with the imprecise nature of user queries as well as incompleteness in the data due to missing attribute values (aka “null values”). In such scenarios, the query processor begins to acquire the role of a recommender system. Specifically, in addition to presenting answers which satisfy the user’s query, the query processor is expected to provide highly relevant answers even though they do not exactly satisfy the query predicates. This broadened view of query processing poses several technical challenges. We propose a decision theoretic model for ranking answers in the in the order of their expected relevance to the user. This model combines a relevance function that reflects the relevance a user would associate with answer tuples and a density function which reflects the each tuple’s distribution of missing data. Adoption of this model foregrounds three general challenges: (i) how to assess the relevance and density functions automatically (ii) how to support efficient query processing to retrieve relevant tuples and (iii) how to make users trust the recommended answers. We present a general framework for addressing these challenges, describe a preliminary implementation of the QUIC system and discuss the results of our preliminary empirical evaluation.

## 1. INTRODUCTION

The popularity of the World Wide Web has lead to the presence of multiple online databases that are often laxly curated and readily accessible to lay users. An important challenge facing the database community is to develop effective mediators that can handle the data incompleteness (caused by lax curation) and query imprecision (caused by the lay user population). This challenge brings to the realm of structured data many issues that traditionally were only handled in information retrieval. Dealing with this challenge is further complicated by the fact that the databases are autonomous and only support query-based access to the mediator.

**Data Incompleteness:** An increasing number of online databases are being populated with little or no curation. For

example, databases like `autotrader.com` are populated using automated extraction techniques which crawl text classifieds and by car owners entering the data through forms. Similar techniques are also used in integration systems for scientific data such as `CBioC (cbioc.org)`. Unfortunately, these methods are error prone (c.f. [17]) which leads to databases that are *incomplete* in that they contain tuples having many null values. In fact, on a random sample of 25,000 tuples from `autotrader.com`, we found that almost 34% were incomplete!<sup>1</sup>

**Query Imprecision:** More and more lay users, be they causal web users, or domain experts that are nonetheless database novices, are accessing autonomous databases on the web. Often these users do not clearly define what they are looking for and many times the form-based query formulating tools do not support imprecise queries. Imprecision can involve posing queries that are either overly general [6] or overly specific [15] w.r.t. the information that is truly relevant. For example, users end up asking queries such as  $Q : CarDB(Model=Civic)$  when they actually want to ask the query  $Q' : CarDB(Model \approx Civic)$  (where “ $\approx$ ” is to be interpreted as “like”). A car of model Accord that does not exactly satisfy the query can be relevant if the user was looking for a reliable Japanese car but over-specified the query. Further more, the query may be over-general in that among all the answer tuples, the ones with a lower mileage are preferred more by the user. A mediator system should be able to support such imprecision in the user’s queries and return tuples which do not exactly satisfy the query constraints but nevertheless are likely to be relevant to the user.

**Example:** Consider a fragment of online Car database  $DB_f$  as shown in Table 1. Suppose a user poses an imprecise query  $Q' : \sigma_{Model \approx Civic}$  on this table. Clearly tuples  $t_1$  and  $t_6$  are relevant as they are exact answers to the query. In addition, the tuples  $t_4$  and  $t_7$  might also be relevant if there are reasons to believe that the missing value might be a Civic. Finally, tuples  $t_2$  and  $t_8$  might be relevant if Civic is considered similar enough to Accord and/or Prelude.

Ideally, a query processor should be able to present such implicitly relevant results to the user, ranking them in terms of their expected relevance. In this paper, we present a general framework as well as a specific current implementation, called QUIC aimed at solving this problem. We start with the philosophy that query imprecision and data incompleteness are best modeled in terms of relevance and density functions. Informally, the relevance function assesses the rel-

This article is published under a Creative Commons License Agreement (<http://creativecommons.org/licenses/by/2.5/>).

You may copy, distribute, display, and perform the work, make derivative works and make commercial use of the work, but you must attribute the work to the author and CIDR 2007.

<sup>1</sup>This type of incompleteness is expected to increase even more with services such as GoogleBase and Craigslist.com which provide users significant freedom in deciding which attributes to define and/or list.

Id	Make	Model	Year	Color	Body Style
1	Honda	Civic	2000	red	coupe
2	Honda	Accord	2004	blue	coupe
3	Toyota	Camry	2001	silver	sedan
4	Honda	null	2004	black	coupe
5	BMW	3-series	2001	blue	convt
6	Honda	Civic	2004	green	sedan
7	Honda	null	2000	white	sedan
8	Honda	Prelude	1999	blue	coupe

Table 1: Fragment of a Car Database

evance the user places on a tuple  $t$  with respect to a query  $Q$ . The density function attempts to capture the probability distribution that an incomplete tuple is in reality representing a complete tuple  $t$  in that all their non-null attribute values are the same.

**Challenges:** Given information about relevance and density functions, it is possible, in theory, to rank a set of possibly incomplete tuples in terms of their expected relevance given a specific query. Simple as it sounds, realizing such a query processing architecture presents several technical challenges, brought about mostly due to the autonomous nature of the databases, and the impatience of the user population:

**Ranking Challenges:** (i) We need a way to combine the density and relevance functions to provide a ranking of the tuples. (ii) Since the users are often impatient, we need methods for automatically and non-intrusively assessing the appropriate relevance function. (iii) Since the databases are autonomous, we need methods for automatically assessing the density function.

**Optimization Challenges:** Since often the query processor has only a form-based interface to access the databases, we need to be able to rewrite the queries appropriately in order to retrieve tuples that are likely to be relevant to the user. This necessitates novel query rewriting techniques.

**Explaining/Justifying Answers:** Since the query processor will be presenting tuples that do not correspond to exact answers to the user’s query, it needs to provide explanations and justifications to gain the user’s trust.

We have been addressing these challenges in the context of a preliminary prototype called QUIC. Our aim in the rest of this paper is to (i) describe the broad issues that arise in tackling these challenges and (ii) describe the approaches that have been taken in the current implementation of QUIC. The architecture of QUIC is shown in Figure 1. QUIC acts as a mediator between the user and autonomous web databases. Because of the autonomous nature of these databases, QUIC has virtually no control or prior knowledge over them. QUIC computes relevance and density functions from a probed sample of the autonomous databases. These functions are in turn used to reformulate the user query in order to retrieve tuples that are relevant even though they may not be exact answers. The retrieved tuples are ranked in terms of a ranking function that we call *expected relevance ranking*. The ranked results are returned with an explanation of the ranking scheme. The current implementation of QUIC uses approximate functional dependencies (AFDs, c.f. [10]) to mine and represent attribute dependencies. As we shall see, AFDs wind up being useful in multiple ways in QUIC.

**Contributions over Prior Work:** The problem we address brings together several core database challenges — including uncertainty, incompleteness and query imprecision, to the context of web with its autonomous databases and lay user population. The problems of data incompleteness and

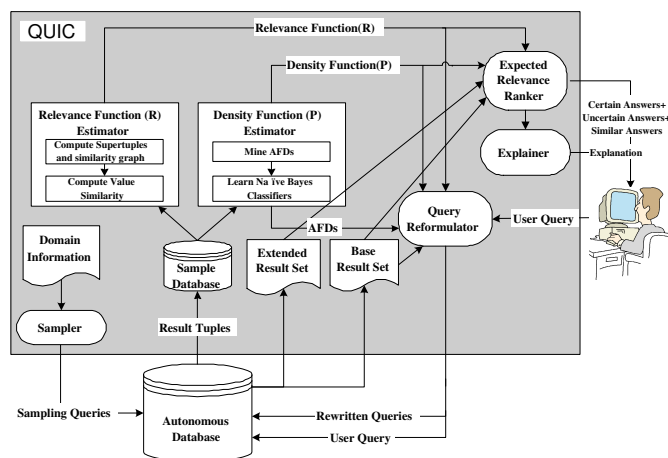


Figure 1: QUIC System Architecture.

query imprecision have, in the past, been tackled in isolation in traditional databases. However, the methods tended to involve direct modification of the databases, to rewrite null values (e.g. [11]) or to insert similarity measures between attribute values (e.g. [13]). The autonomous nature of the web sources, as well as the impatience of the lay users precludes straightforward adaptation of these methods. The complications include the need to automatically, indirectly and non-intrusively assess relevance and density functions, as well as the need to use these assessed functions in reformulating the user query so as to retrieve tuples that are expected to be of relevance to the user.

Assessment of relevance and density functions are special cases respectively of the general problems of “utility/preference elicitation” [4, 5] and “density estimation” [9, 16, 17] that have been studied extensively in the AI, statistics and pattern recognition communities. The challenge we face is adapting, from this welter of literature, the right methods that are sensitive to the autonomous nature of the databases as well as the possibly non-cooperative user population. Given that our treatment of incomplete tuples involves converting them into uncertain tuples, work on probabilistic databases [3, 18, 2] shares some of our goals. It however focuses on single centralized databases, assuming that the uncertainty in the base data is already quantified and can be recorded and handled within the databases themselves. In contrast, QUIC focuses on autonomous databases, and thus must assess the uncertainty in the base data automatically and handle the uncertainty by the mediator without modifying the underlying databases.

Work on QUIC is built on our previous work on the AIMQ system [15] that focuses exclusively on query imprecision, and our more recent work on QPIAD [12] which focuses exclusively on data incompleteness. QUIC extends these prior efforts both in terms of addressing query imprecision and data incompleteness together, as well as investigating a larger spectrum of methods for assessing relevance and density functions, and using them for query rewriting.

**Organization:** In the next section, we discuss how QUIC handles each of the challenges discussed above. In each sub-section we first discuss the broader challenges, and then present the current approaches taken in QUIC. This is followed in Section 3 by an empirical evaluation of the current implementation of QUIC. We conclude with a discussion of the directions we are currently pursuing.

## 2. CHALLENGES & CHOICES MADE

In this section, we define *Expected Relevance* as the basis for ranking results in the presence of query imprecision and data incompleteness. We use  $\hat{t}$  to denote a (possibly incomplete) tuple of a database  $D$  and  $t$  to denote a complete tuple. A complete tuple has a non-null value for each of the attributes, while an incomplete tuple has null values for some of its attributes. A tuple  $t$  is considered to belong to the set of completions of  $\hat{t}$  (denoted  $C(\hat{t})$ ), if  $t$  and  $\hat{t}$  agree on all the non-null attribute values.

### 2.1 Expected Relevance Ranking Model

The expected relevance ranking of a (possibly incomplete) tuple  $\hat{t}$  given a query  $Q$  by a user  $U$  over a database  $D$ ,  $\mathcal{ER}(\hat{t}|Q, U, D)$  is specified in terms of two independent functions, the relevance function  $\mathcal{R}$  and the density function  $\mathcal{P}$  as defined below.

$\sigma_{Model \approx Civic}$	<i>Civic</i>	<i>Accord</i>	<i>Prelude</i>	<i>Corolla</i>
Relevance	1.0	0.78	0.59	0.48
Density	0.62	0.21	0.17	0.0

**Table 2: Relevance for the query  $Q'$ :  $\sigma_{Model \approx Civic}$  and Density for tuple  $\hat{t}_4$ .**

**Relevance Function  $\mathcal{R}$ :** The relevance function,  $\mathcal{R}(t|Q, U)$ , is the relevance that user  $U$  associates with a complete tuple  $t$  as an answer to the query  $Q$ . Table 2 shows a possible relevance function for the query  $Q$ :  $Model \approx Civic$ . Here  $\mathcal{R}(t|Q, U)=1$  for tuples having  $Model=Civic$ , 0.78 for tuples with  $Model=Accord$ , 0.59 for tuples with  $Model=Prelude$ , and so on. Moreover, it need not be the case that all tuples having  $Model=Civic$  are of the same relevance to the user. Specifically, by considering the user's query to be under specified, we can give varying relevance scores to tuples with the same values for the constrained attributes (c.f [6]). For example, given two tuples each having  $Model=Civic$ , it could be the case that the car with lower *Mileage* is more relevant to the user even though they may not have made it explicit.

**Density Function  $\mathcal{P}$ :** The density function,  $\mathcal{P}(t|\hat{t}, D)$ , is the probability that a (possibly incomplete) tuple  $\hat{t}$  from database  $D$  corresponds to a complete tuple  $t \in C(\hat{t})$  (where  $C(\hat{t})$  is the set of completions of  $\hat{t}$ ). Table 2 shows a possible density function for the incomplete tuple  $\hat{t}_4$ , which estimates the probabilities of the possible values of the null over the domain of the *Model* attribute (for example, based on the rest of the attribute values and correlations between attributes). In this case, the distribution of possible values is: 0.62 probability for *Civic*, 0.21 for *Accord*, 0.17 for *Prelude*, etc.

**Expected Relevance  $\mathcal{ER}$ :** Our ranking function,  $\mathcal{ER}(\hat{t}|Q, U, D)$  can now be defined as:

$$\mathcal{ER}(\hat{t}|Q, U, D) = \sum_{t \in C(\hat{t})} \mathcal{R}(t|Q, U) \mathcal{P}(t|\hat{t}, D)$$

It is easy to see that the *ERR* model is general enough to handle traditional database query processing where the relevance and density function are dirac delta functions ( $\mathcal{R}(t|Q, U) = 1$  if  $t.A = v$  and 0 otherwise and  $\mathcal{P}(t|\hat{t}, D) = 1$  if  $\hat{t} = t$  and 0 otherwise) as well as query processing under data incompleteness and/or query imprecision.

Processing the query  $Q$ :  $Model \approx Civic$  on the car database, we retrieve not only the tuples that exactly satisfy

$Q$ , namely  $t_1$  and  $t_6$ , but also highly relevant tuples, including incomplete tuples  $t_4$  and  $t_7$ , and similar tuples  $t_2$  and  $t_8$ . These tuples are then ranked and presented in the decreasing order of their expected relevance based on the *ERR* model, where both the relevance and density functions are considered.

### 2.2 Estimating Relevance Function

**Broad Challenges:** Assessing the relevance function in general involves preference elicitation. For example, preferences could be used to determine relative importance of attributes as well as similarity between two attribute values. In the web scenarios, preferences could be obtained in several ways: (i) Preferences can be directly expressed by users as part of their queries. Several IR systems allow user specification of imprecise queries. The problem is that shifting the burden of preference expression to the users does not normally work well (ii) Preferences can be learned by monitoring user's clickstream or stare patterns. However, individually monitoring every user's preferences might be expensive. (iii) Preferences can be learned indirectly by analyzing users' past query patterns. Understanding the trade-offs offered by these approaches is an open problem.

**Current Implementation in QUIC:** QUIC currently focuses on imprecision due to overly-specific queries. Rather than assess user-specific relevance functions, QUIC attempts to assess relevance functions for the whole user population. Further more, QUIC assumes that relevance function is based on attribute value similarity and attribute importance. Currently, three alternative approaches are supported for computing the value similarities: content based filtering, collaborative filtering which uses the click patterns of the user population (where available), and co-occurrence statistics.

**Content Based:** In QUIC's content based relevance measurement, we learn a relevance function between two attribute values without any user interaction. QUIC issues a query binding each attribute value pair (AV) to a sample database, where the query result forms a supertuple for that AV pair (given a set  $S$  of tuples, a supertuple  $t_S$  is constructed as a tuple of bags, where each bag contains the possible values a particular attribute took in the tuples in  $S$  [15]). QUIC computes the similarity between supertuples (as measured by Jaccard coefficient with bag semantics) to estimate the similarity between AV pairs:  $Sim_J(t_S^1.A_i, t_S^2.A_i) = \frac{|t_S^1.A_i \cap t_S^2.A_i|}{|t_S^1.A_i \cup t_S^2.A_i|}$ , where  $t_S^1, t_S^2$  are the supertuples, and  $t_S^1.A_i$  ( $t_S^2.A_i$ ) is the bag of values of attribute  $A_i$  in  $t_S^1$  ( $t_S^2$ ).

**Co-click Based:** In contrast to the content based relevance function assessment which does not involve actual user transactions, we also tested collaborative filtering based on co-click patterns. In this implementation, we obtained user clickstreams from the Yahoo! Autos ([autos.yahoo.com](http://autos.yahoo.com)) web pages by scraping the collaborative recommendations provided by the site. Specifically, each car's web page contained 1 to 3 recommendations for other cars which were viewed by people who also viewed the car on the given page. Using this information, we create an undirected weighted similarity graph for attributes like *Make* and *Model* (where multiple links between two nodes are reduced to a single link having a weight equal to the total number of links between the nodes).

Let  $CSim(N_1, N_2)$  denote the co-click similarity between two nodes  $N_1$  and  $N_2$  in the similarity graph. If  $N_1$  and  $N_2$  are connected by an edge, we have  $CSim(N_1, N_2) =$





