NATIVE XML SUPPORT FOR SEMISTRUCTURED PROBABILISTIC DATA MANAGEMENT

A Thesis
Presented to
the Faculty of California Polytechnic State University
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
Evan Pierce Rosson

AUTHORIZATION FOR REPRODUCTION OF MASTER'S THESIS

I reserve the reproduction rights of this thesis for a period of seven years from the date of submission. I waive reproduction rights after the time span has expired.

Signature

Date

APPROVAL PAGE

TITLE: Native XML Support for Semistructured Probabilistic Data Management

AUTHOR: Evan Pierce Rosson

DATE SUBMITTED: June 2007

Dr. Alexander Dekhtyar
Advisor or Committee Chair ______________________________
Signature

Dr. John Clements
Committee Member ______________________________
Signature

Dr. Clark Turner
Committee Member ______________________________
Signature

Abstract

Native XML Support for Semistructured Probabilistic Data Management

Evan Pierce Rosson

Many applications require storing and querying probabilistic information; for example, the risk analysis software used by insurance companies. Probabilistic databases are designed to store such data and support queries using operations based on probability theory. Semistructured databases, often based on XML, allow storage of data that may not strictly conform to a schema, which permits data imported from sources with many different schema. The Semistructured Probabilistic Database Management System (SPDBMS) combines these two approaches. It stores Semistructured Probabilistic Objects (SPOs), probability distributions of variables with discrete and finite domains expressed as XML documents. The SQL-like Semistructured Probabilistic Object Query Language (SPOQL) is used to query and manipulate SPOs using operations based on probability theory.

We present a native XML implementation of SPDBMS, better suited to the semistructured nature of this data than the original relational backend. This is implemented using XQuery, a functional query language for processing XML; and ExistDB, an open-source XML database. The performance of this new implementation is compared with the existing relational implementation. We also implement a new SPDBMS query operation, MIX, and a distinction between LEFT and RIGHT JOIN queries.

Contents

List of Figures

1 Introduction

1.1 Probabilistic Databases

Many applications require storing and querying probabilistic information: stock market prediction, risk analysis software used by insurance companies, and image recognition, to name a few [10] [11]. Such probabilistic data is poorly suited to a traditional relational database - operations common to probabilistic data are not provided by relational databases. A *probabilistic database* allows one to apply transformations to the database's content, based on the laws of probability theory, and perform queries based on these probabilistic transformations. Probabilistic databases have been the subject of much research, and a number of different models have been proposed ([11], [7] and section 2.4).

1.2 Our Semistructured Probabilistic Database

The existing structured probabilistic systems cited above lack the flexibility to work well with real-world data taken from multiple sources, or probabilistic data that otherwise fails to match a common schema [11]. A solution is presented in [9] in the form of a Semistructured Probabilistic Database (SPDB). SPDB is a system for storing probabilistic data as Semistructured Probabilistic Objects, SPOs. An SPO specifies a probability distribution for any number of random variables of finite domain, along with non-probabilistic data; the structure described in detail in section 2.1. Semistructured probabilistic relations store an arbitrary collection of SPOs; as semistructured data, no common schema is needed for the storage of SPOs in a relation.

SPOs are manipulated via a semistructured probabilistic algebra, also described in [9], through which the probability distribution of a set of SPOs can be queried. SP-algebra is implemented via a Semistructured Probabilistic Object Query Language (SPOQL), an SQL-like language. SPOQL and SP-algebra support - among other operations - conditionalizing SPOs to recalculate probability based on known values, and creation of joint probability distributions from multiple SPOs via join and Cartesian product.

1.3 Motivation for an XML backend

XML is particularly well-suited to the semistructured nature of SPO data. In addition, because SPDB currently presents SPOs to the user in XML form, storing SPO data in a relational database is unnatural and the conversion adds a great deal of overhead. We propose an XML backend to SPDB, implemented with an XML-based database and XQuery, to alleviate performance problems and simplify the system as a whole. Our solution is implemented using the open-source XML database ExistDB.

Our primary contribution is the design, implementation, testing, and performance evaluation of the native XML backend described above. We also implement a distinction between LEFT and RIGHT JOIN operations and implement the MIX operation; these operations are specified in earlier works, but were not implemented until now. We also provide a suite of automated tests for verifying the correctness of SPDB.

1.4 Outline

Section 2 provides background information on the project, including the semantics of each SPOQL operation. Section 3 describes the design of the SPDBMS server and its integration with the native XML backend, and section 4 goes into further detail on the design and implementation of the XQuery SPDB library. Section 5 discusses the design and results of experiments which test the performance of the new storage.

ω: SPO-1		
university :	Cal Poly	
CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507
CSC101 = A		

Figure 1: An example of an SPO in table format. "University: Cal Poly" is context information. CSC560 and CSC305 are random variables. The rows of grades for each random variable and the probability for each form the probability table. The precondition of an A in CSC101 is conditional information.

2 Background

2.1 Data Model

SPOs - *Semistructured Probabilistic Objects* - are the structure SPDB uses to represent probabilistic data. An SPO is a tuple consists of four parts: Context, Variables, the Probability Table, and Conditionals [9].

Examples of SPOs in tabular and XML formats are shown in figures 1 and 2, respectively.

More formally, consider the set $\mathcal{R}$ of all possible relational attributes; and $\mathcal{V}$, the set of all possible random variables and their domains. An SPO is a tuple $S = \langle T, V, P, C \rangle$, where: [11, section 3] [9]

- T is a relational tuple of a *semistructured schema* over $\mathcal{R}$, all possible relational attributes.
- $V = \{v_1, \ldots, v_q\} \subseteq \mathcal{V}$ is the set of *participating random variables* in S,

```
<spo>
    <context>
        <elem><name>university</name><val>Cal Poly</val></elem>
    </context>
    <table>
        <variable><name>CSC560</name> <name>CSC305</name></variable>
        <row> <val>A</val> <val>A</val> <P>0.34476</P> </row>
        <row> <val>A</val> <val>B</val> <P>0.32192</P> </row>
        <row> <val>B</val> <val>A</val> <P>0.05507</P> </row>
    </table>
    <conditional>
        <elem><name>CSC101</name> <val>B</val></elem>
    </conditional>
</spo>
```

Figure 2: An example of an SPO in XML format. “University: Cal Poly” is context information. CSC560 and CSC305 are random variables. The rows of grades for each random variable and the probability for each form the probability table. The precondition of an A in CSC101 is conditional information. This SPO is equivalent to the example in table format from figure 1.

where $\mathcal{V}$ is the set of all possible random variables, and $V \neq \emptyset$.

- $P : dom(V) \longrightarrow \mathcal{P}[0,1]$ is the *probability table* of s. P need not be complete: total probability may be less than one.

- $C = \{(u_1, X_1), \ldots (u_s, X_s)\}$, where $\{u_1, \ldots, u_s\} = U \subseteq \mathcal{V}$ and $X_i \subseteq dom(u_i)$, $1 \leq i \leq n$ such that $V \cap U = \emptyset$, is the *conditional* of S.

To elaborate on this definition, less formally:

- *Participating random variables* are the names of each column seen in figures 1 and 2. *Participating* variables are part of the probability table, and distinct from the variables found within an SPO's *conditionals* but not in the table. Each variable has a finite *domain*, represented as the range of values for each variable in the probability table and conditionals. In the above example, CSC560 and CSC305 are all participating random variables.

- The *probability table* specifies the probability distribution for each set of events. Each *column* in the table is a participating random variable. Each *row* in the table contains a set of values for each variable within the domain of that variable, and the probability that this combination of values will occur.

  A probability table is *complete* if it contains a probability for every possible instance - that is, every possible combination of values within the domain of each variable. Incomplete tables may be stored and queried as well. The table in our example above is not complete - observe that a row for (CSC560=B, CSC305=B) is missing, and other possible values for each random variable (in our example, grades of C, D, and F) may account for other missing rows as well.

- *Conditional* information represents random variables with a known value or restricted to a subset of its domain. When analyzing a probability distribution, we often have prior information on the value of some variables. Our definition represents this restriction as (u, X), where u is a random variable and X is a subset of the domain of u. Unlike context, conditionals are subject to change: *conditionalization* queries may add new conditions to the SPO.

- *Context* is any supporting information for this probability distribution. Context may contain any set of name-value pairs whose values are certain, and these values has has no effect on the probability calculations performed elsewhere. In the example of figures 1 and 2, *university* is part of the context - it is known in advance, not a random variable, and plays no part in probability calculations.

2.2 SPOQL and SP-Algebra Operations

SPOQL - Semistructured Probability Object Query Language - is an SQL-like language used to extract information from the database. Like SQL, SPOQL queries have *no side effects* - the operations discussed below do not

Operation	Section	SP-Algebra	SPOQL Syntax
Selection	2.2.1	$\sigma_c(S)$	`SELECT * FROM` S `WHERE` c
Projection	2.2.2	$\pi_f(S)$	`SELECT` f `FROM` S
Conditionalization	2.2.3	$\mu_d(S)$	`SELECT * FROM` S `CONDITIONAL` d
Cartesian Product	2.2.4	$S_1 \times S_2$	`SELECT * FROM` S_1 `TIMES` S_2
Left Join	2.2.5	$S_1 \ltimes S_2$	`SELECT * FROM` S_1 `[LEFT]JOIN` S_2
Right Join	2.2.5	$S_1 \rtimes S_2$	`SELECT * FROM` S_1 `RIGHTJOIN` S_2
Left Mix	2.2.6	$S_1 \otimes_L S_2$	`SELECT * FROM` S_1 `[LEFT]MIX` S_2
Right Mix	2.2.6	$S_1 \otimes_R S_2$	`SELECT * FROM` S_1 `RIGHTMIX` S_2

Figure 3: A summary of conceptual operations on SPOs, the relevant section in this paper, the SP-algebra symbols, and the corresponding SPOQL syntax. For left join/left mix, SPOQL in [brackets] is optional: left is the default direction if none is specified.

modify the contents of the database.

SP-Algebra, specified in [11], is the theoretical basis for SPOQL; similar to relational algebra as a basis for SQL. SPDBMS input is in the form of SPOQL. In the following sections, we'll outline each SP-algebra operation.

The table in figure 3 summarizes each operation, including the SP-algebra symbol, SPOQL syntax, and section in which it is discussed.

2.2.1 Simple Selection

The familiar SELECT statement is used to view the SPOs stored in a given relation.

`WHERE` statements, also familiar from SQL, can be used to restrict the set of SPOs viewed. WHERE conditions may be based on equality or comparisons with the following fields:

- Selection on Variables: Show only SPOs that have the given variable somewhere in their probability table, with any value.

  ```
  SELECT * FROM relation WHERE var.CSC101 IN V
  ```

- Selection on Table: Show only SPOs where the given variable exists with the given value.

  ```
  SELECT * FROM relation WHERE tbl.CSC101 = A
  ```

- Selection on Probability: Show only rows in the probability table where probability is within the given bounds.

  ```
  SELECT * FROM relation WHERE tbl.prob > 0.10
  ```

- Selection on Conditionals: Show only SPOs where the given conditional exists with the given value.

  ```
  SELECT * FROM relation WHERE cnd.CSC101 = A
  ```

- Selection on Context: Show only SPOs containing the given context element.

  ```
  SELECT * FROM relation WHERE cnt.year = 1999
  ```

Multiple `WHERE` conditions may be combined using `AND` and `OR` operators:

```
SELECT * FROM relation WHERE cnt.year = 1999 AND cnd.CSC101 = A
```

ω: S

university : Cal Poly
department : CSC

CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507

CSC101 = A
CSC102 = A

ω: $\pi_{cnt.university}(S)$

university : Cal Poly

CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507

CSC101 = A
CSC102 = A

Figure 4: Example of the projection operation on context information. The context information 'department' is removed and 'university' is selected.

ω: S

university : Cal Poly
department : CSC

CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507

CSC101 = A
CSC102 = A

ω: $\pi_{cnd.CSC101}(S)$

university : Cal Poly
department : CSC

CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507

CSC101 = A

Figure 5: Example of the projection operation on a conditional. The conditional 'CSC101' is selected and 'CSC102' is removed.

2.2.2 Projection

Projection simplifies an SPO by returning the SPO with only a subset of its original variables, conditionals, or context values.

Projection on Conditional and Projection on Context Conditional and context projection are easy to understand: given an initial SPO, these select only a subset of that SPO's conditional or context tuples, respectively. See the examples in figures 4 and 5.

ω: S

university : Cal Poly

CSC560	CSC305	P
A	A	0.34476
A	B	0.32192
B	A	0.05507

CSC101 = A

(Step 1)

ω: $\pi_{var.CSC305}(S)$

university : Cal Poly

CSC305	P
A	*0.34476*
B	0.32192
A	*0.05507*

CSC101 = A

(Step 2)

ω: $\pi_{var.CSC305}(S)$

university : Cal Poly

CSC305	P
A	*0.39983*
B	0.32192

CSC101 = A

Figure 6: Example of the projection operation on a variable, shown in two steps for clarification. Step 1 removes all variables not selected by the projection; step 2 merges all duplicate rows. In step 1, observe the two rows whose events are identical: both have only a single variable with value 'A'. These rows are merged in step 2 by summing their probabilities.

Projection on Variables Much like relational projection, SPO projection on variables modifies the SPO by selecting only a subset of its columns. Projection on variables may not result in duplicate rows - if all values for two rows are identical after a projection, the rows are merged and their probabilities are summed. The example in figure 6 illustrates this process.

Unlike the conditionalization operation (2.2.3), an SPO's conditionals are not modified by projection on a variable.

ω: S

university : Cal Poly

CSC560	CSC305	P
A	A	0.300
A	B	0.100
B	A	0.200

CSC101 = A

(Step 1)

ω: $\mu_{var.CSC560=A}(S)$

university : Cal Poly

CSC560	CSC305	P
A	A	$0.300 * (0.600/0.400)$
A	B	$0.100 * (0.600/0.400)$
B	*A*	—

CSC101 = A
CSC560 = A

(Step 2)

ω: $\mu_{var.CSC560=A}(S)$

university : Cal Poly

CSC560	CSC305	P
A	A	0.450
A	B	0.150

CSC101 = A
CSC560 = A

(Final step)

ω: $\mu_{var.CSC560=A}(S)$

university : Cal Poly

CSC305	P
A	0.450
B	0.150

CSC101 = A
CSC560 = A

Figure 7: Example of the conditionalization operation, shown in four steps for clarity. In the first step, we add the new conditional value 'CSC560 = A' and show our calculations for new probabilities for each existing row. Probabilities are scaled such that the total probability after the conditionalization is equal to the total before the conditionalization: in this example, the pre-conditionalized total is 0.6 and the intermediate total is 0.400 (0.300 + 0.100), so the remaining rows' probability is scaled up by (0.6 / 0.4) = 1.5. Step 2 removes rows that are incompatible with the new conditional (here, rows where CSC560 != A). In the final step, the conditionalized column is finally removed from the probability table.

2.2.3 Conditionalization

Conditionalization fixes the value of a given variable. This has the effect of removing columns from the probability table, like projection, but the variable is not removed from the simulation - it is instead moved to the *conditional* section with the specified value. Rows without the specified value are removed from the probability table - after fixing the value for this variable, these rows need not be analyzed - and the probability of all remaining rows is normalized. An example of this process is shown in figure 7.

If a `CONDITIONAL` operation on variable V is applied to an SPO which does not reference V, that SPO's `conditional` data is updated, but the SPO's probability table is unchanged. This makes sense: requiring a condition to an irrelevant variable should have no effect on the probability of each result.

The sum of probabilities for a complete table is unchanged after conditionalization. For complete tables, the sum of probabilities is always 1. Incomplete tables may be conditionalized as well.

2.2.4 Cartesian Product

A Cartesian product constructs a joint probability distribution from two *product-compatible* SPOs. Two SPOs are product-compatible if they have matching conditionals, and have no random variables in common [11]. The result looks similar to a Cartesian product in a relational database - each pair of rows in the probability table is returned. One important difference from relational databases is, of course, the probability of each row. Probabilities

ω: S_1

university : Cal Poly

CSC560	P
A	0.4
B	0.5

CSC101 = A

ω: S_2

university : Cal Poly

CSC305	P
A	0.1
B	0.8

CSC101 = A

ω: $S_1 \times S_2$

university : Cal Poly

CSC560	CSC305	P
A	A	0.04
A	B	0.32
B	A	0.05
B	B	0.40

CSC101 = A

Figure 8: Example of the Cartesian product operation. We begin with two SPOs for this operation: S_1 and S_2. These are product compatible: *conditionals are identical* and they have *no common participating random variables*. The result contains all participating random variables from both SPOs. Rows are a Cartesian product of rows from the original SPOs, much like a Cartesian product in a relational database; probability for each row is the product of the probabilities from each row that created it.

for each pair of rows are multiplied to find the probability used in the product row. This operation is illustrated in figure 8.

2.2.5 Join

A join is similar to a Cartesian product - it constructs a joint probability distribution from two *join-compatible* SPOs. Two SPOs are join-compatible if their conditionals match, as with product-compatibility; and they must have *at least one* common random variable [11].

To perform a join, one of the two SPOs is *conditionalized* (Conditionalization is explained in section 2.2.3.) Results differ depending on which side is conditionalized. We may specify which side is conditionalized by using `LEFTJOIN` or `RIGHTJOIN`.

2.2.6 Mix

Cartesian product and join are mutually exclusive. Mix chooses the appropriate operation for a pair of SPOs: if the pair is join-compatible, mix executes a join; if the pair is product-compatible,

While a pair of SPOs cannot be *both* product- and join-compatible, note that a pair can be *neither* product- or join-compatible if their conditionals are not equal.

As discussed in the previous section, `JOIN` may have a direction: `LEFTJOIN` or `RIGHTJOIN`. Since `MIX` may execute joins, we specify the direction of joins by using two types of mixes as well: `LEFTMIX` and `RIGHTMIX`. The direction of a mix is irrelevant for a pair of product-compatible SPOs; product does not have a direction.

2.3 Data Modification

All SPOs in the database are stored in a parent *relation*. Each relation is a named grouping of any set of SPOs. Any two SPOs may be part of the same relation - because SPOs are *semistructured* data, no schema must be enforced.

TIMES, JOIN, and MIX operations (see sections 2.2.4, 2.2.5, and 2.2.6 respectively) can operate on any two relations, provided the SPOs each relation contains are product-compatible or join-compatible.

2.3.1 Operations

- CREATE Creates a new relation in the database. A name for the relation must be provided. Relations are empty when created - use *INSERT* to populate relations with SPO data.

- INSERT Adds new SPO data to a given relation. SPO data is provided via an XML file.

- DELETE Remove all SPOs from a relation matching the given WHERE conditions (see section 2.2.1).

- DROP Removes an existing relation, given the name of the relation to drop. All SPOs in the relation are destroyed.

2.4 Related Work

SPDBMS has been the subject of much past work. Zhao and Dekhtyar defined SP-algebra and implemented the original system, including its relational backend [11]. Mathias specified the SPOQL language and implemented it on top of Zhao and Dekhtyar's work [9].

Numerous models for probabilistic relational databases have been developed[11]. A data model proposed by Barbará, Garcia-Molina and Porter [5] extends the relational model to support uncertain data. Each relation must have a deterministic key, and all other attributes may be either deterministic (certain) or stochastic (uncertain). Stochastic attributes have an associated probability. Attributes may also be specified as independent or interdependent: the probability of an independent attribute is not tied to that of any other attribute; the probability of an interdependent attribute is tied to another attribute.

Cavallo and Pittarelli [8] incorporate probability into relational databases as a tuple (V, Δ, dom, p): V is a set of unique attributes, δ is the set of attribute domains, *dom* provides the mapping $V \rightarrow \Delta$, and p is a probability distribution over V. Their model required a sum of 1 for the probabilities of all tuples in a relation. They define projection and join operators.

MystiQ [6] is a relatively recent probabilistic database system, implemented as middle ware on a standard relational database. MystiQ manipulates probabilistic data stored in a standard relational database; however, MystiQ has a different concept of probabilistic data than our work. MystiQ issues queries on a standard relational database using 'fuzzy' queries, where query restrictions are not completely defined. MystiQ will return standard relational rows as results, each row paired with a probability that it was what the user wanted.

These probabilistic databases based on the relational model have inherent weaknesses. Many works describe a single probabilistic object as a single database row with attributes whose values are uncertain. We instead represent a probabilistic object as a set of random variables that form a probability distribution, where the probability of multiple combinations of values can be specified. Another strength of our work over related work is its semistructured nature - SPOs from a single relation, representing the same type of object, need not have identical schema. This allows us to import and work with data from multiple sources with varying schema.

3 SPDBMS Design

3.1 Existing System Design

The original SPDB server design, as described in [9, section 2.2], is shown in figure 9. The SPDB server stores all SPO data in a relational database. Clients connect to the server via TCP/IP, and manipulate stored data or issue SPOQL queries. The SPO Request Dispatcher parses SPOQL queries into SP-algebra strings, further parses SP-algebra strings into an SP-algebra based abstract tree, and passes the final structure to the database adapter for execution. The database adapter constructs DBMS-specific SQL based on this structure to either update the SPOs/relations as requested by an `INSERT`, `DELETE`, `CREATE`, or `DROP` operation; or the SQL required to construct the SPO XML returned by the application for a `SELECT` operation [2]. In the original design, these operations may require the execution of any number of SQL statements against the underlying database.

Decoupling the database adapter from the rest of the application in this design allows new database backends to be constructed with minimum effort - most application logic is not tied to the DBMS backend.

The implementation of this database adapter design required a JDBC interface to the database. All other details of database access are left to the database adapter implementer.

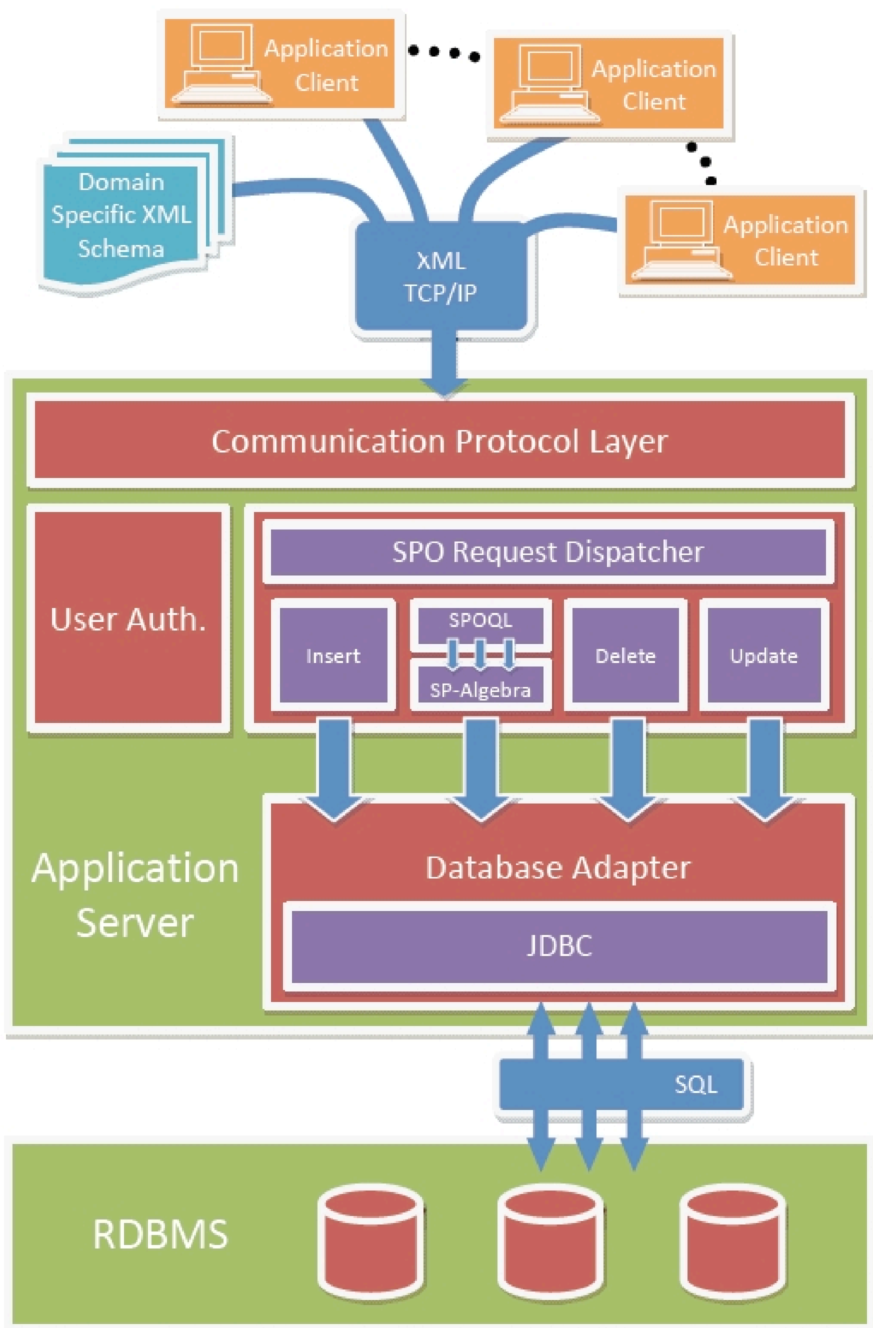


Figure 9: The original SPDBMS system architecture, as described in [9, section 2.2]. (Image source: [2, figure 2])

3.2 XML Database Adapter Design

The SPDB server design (figure 9) specifies a relational database beneath the database adapter, but the implementation does not enforce this; we generalize this design to allow XML databases.

The design for our Exist-DB adapter for SPDB consists of two stages:

- Query planning: given the abstract query tree sent to the database adapter from the query parser, construct a single XQuery statement.
- Query execution: execute the planned XQuery statement.

This is modeled after the query planning and execution stages employed by many relational databases. The advantages of this separation have been shown many times for relational databases, and these advantages (discussed below) are applicable here as well. [1]

One advantage of this approach is that *only one* XQuery statement is constructed per SPOQL query. This minimizes the overhead required by database round-trips. This is a dramatic improvement over the current Oracle implementation's ad-hoc approach to querying the underling database, where many round-trips are required per SPOQL query. An advantage of a native XML backend is that no post-processing of data returned from XQuery is necessary - SPDB returns the same XML data that XQuery excels at processing.

[1] In particular, we appreciated having query execution decoupled from query planning when creating automated tests for query planning.

All SPO-related logic is processed by the XQuery backend as XQuery functions, rather than Java-based SPO server. This has advantages similar to stored procedures in a relational environment: among other things, the caller need not be aware of the XQuery implementation. Removing this logic from the caller also allows us to take full advantage of XQuery's ability to manipulate semistructured XML data.

All XQuery commands sent to the server begin with the following, in order to access this XQuery function library

```
import module spdb="http://www.csc.calpoly.edu/~erosson/spdb"
as "resource:spdb.xqm";
```

`spdb.xqm` is the name of our XQuery function library, implemented as an XQuery module. `http://www.csc.calpoly.edu/~erosson/spdb` is the XML namespace required for the module. According to [1] this module is only compiled once, and cached for all queries thereafter. (See our experimental results, section 5.2.1 for further discussion of this.)

In addition, the strategy described above greatly simplifies query planning. For example, consider the following SPOQL statement:

```
SELECT * FROM relation1 LEFTJOIN relation2
```

The SPOQL query planner translates this to reasonably concise and readable XQuery:

```
import module spdb="http://www.csc.calpoly.edu/~erosson/spdb"
```

```
as "resource:spdb.xqm";
spdb:leftjoin(spdb:relation("relation1"), spdb:relation("relation2"))
```

The complex logic behind `JOIN` is abstracted behind an interface of XQuery functions.

3.3 XQuery Interface Design

Each SP-Algebra operation specified in section 2 is implemented as an XQuery function. These XQuery functions provide all of the system's probability analysis features - the SPDB server merely translates SPOQL queries into the matching XQuery calls.

All functions in the XQuery API - that is, any function referenced from the SPDB server corresponding to an SP-Algebra operation - will always accept input and require output of the same form:

```
<spos>
    <spo>
        ...data...
    </spo>
    <spo>
        ...data...
    </spo>
    ...more SPOs...
</spos>
```

This can be verified by examining the method signature for any XQuery function called from SPDB:

```
declare function spdb:sp-some-function($spos as element(spos), ...)
as element(spos);
```

This signature allows the output of any one function to be passed to another. Arbitrary complex SPOQL queries can be planned before any results are computed.

4 XQuery API Implementation

See appendix A for the source of the XQuery module. All functions are written by Evan Pierce Rosson unless another author is noted. To summarize, Dustin Anderson implemented all simple selection WHERE-clauses (section 2.2.1), and projection on context and conditionals (2.2.2). Evan Pierce Rosson implemented all other operations: the current projection on variables (2.2.2); AND and OR conditions used in WHERE-clauses; CONDITIONAL, JOIN, TIMES, and MIX operations (2.2.3, 2.2.4, 2.2.5, and 2.2.6 respectively).

Note that there are a number of auxiliary functions present. Functions called by the SPDB server are differentiated from helper functions only by function name - 'public' functions have names beginning with 'sp-'. All other functions are exposed only to allow for thorough unit testing.

The following sections will discuss the implementation of each function implemented by Evan Pierce Rosson, including relevant code.

4.1 Simple Selection

4.1.1 OR

The implementation of `OR` is straightforward. Each parameter of the OR operation is evaluated, and the union of the results is returned. To avoid returning duplicates - SPOs included in both arguments to OR - the built-in function `distinct-deep` performs a distinct union - SPOS present in both

```
declare function spdb:sp-where-or($spos1 as element(
   spos), $spos2 as element(spos)) as element(spos)
   {
  element spos {$spos1/@*, functx:distinct-deep((
     $spos1/spo, $spos2/spo)) }
};
```

Figure 10: WHERE condition: OR implementation. “WHERE X OR Y” becomes “sp-where-or(X, Y)”

arguments are included only once.

4.1.2 AND

`AND` could have been implemented in our XQuery library as an intersection. Instead, we chose a solution that required no XQuery function implementation, and a simple implementation in the SPDBMS server. The SPOQL query `SELECT * FROM Relation WHERE X AND Y` is translated to XQuery similar to `spdb:X(spdb:Y(spdb:relation("Relation")))`. The requirements of WHERE filter X are applied only to the results of filter Y, not the entire relation. This leads to a more efficient implementation and more concise XQuery.

In the above example, the XQuery statements `spdb:Y(spdb:X(spdb:relation("Relation")))` and `spdb:X(spdb:Y(spdb:relation("Relation")))` are equivalent. One of these may be faster than the other: we would prefer to execute the faster, more selective (fewer results) condition first. This minimizes the number of results that need to be processed by the slower, less selective outer function. Currently we do not attempt this optimization: order of functions in the generated XQuery is based solely on the order of `WHERE` conditions in the SPOQL input. Estimating the cost of each function for this sort of optimization is a good opportunity for future work (section 6).

4.2 Projection on Variables

The general algorithm used in our implementation of projection on variables is:

- Remove all projected columns from the result.
- Merge all *duplicate rows* by summing their probabilities.

Recall from section 2.2.2 that projection on variables must not create duplicate rows. Rows that are identical (except for probability) after removing a column are to be merged by summing their probabilities into a single row.

Identifying duplicate rows proved to be an expensive operation in XQuery. Given n rows, every possible *pair of rows* must be compared, for a total of $\frac{n^2}{2}$, comparisons.

Oracle uses a hash table, with hashed rows as keys, to achieve linear complexity. XQuery, however, lacks the concept of a hash table. We hoped ExistDB's XQuery processor would be able to optimize this case. This, and further optimization possibilities, are further discussed in our experimental results, section 5.2.2.

```
declare function spdb:rows-project-var($rawrows as element(row)*,
$visiblenums as xs:integer*) as element(row)*
{
  let $rowsviscols := for $row in $rawrows
    return element row {$row/@*,
      $row/val[position() = $visiblenums],
      $row/P
    }
  return for $row1 at $i in $rowsviscols
    (: where no previous rows have identical values :)
    where fn:empty($rowsviscols[position() lt $i][not(val != $row1/val)])
    return element row {$row1/@*,
      $row1/val,
      element P {$row1/P/@*,
        (: sum all rows with identical values. skip previous rows: we checked those already :)
        sum(for $p in $rowsviscols[position() ge $i][not(val != $row1/val)]/P
          return xs:decimal($p))
      }
    }
};
```

Figure 11: Implementation of projection on variables.

4.3 Conditionalization

Our XQuery conditionalization implementation is shown in figure 12.

Notice that it's possible for a conditionalization to change an SPO's `<conditional>` data without modifying its probability table, as described in 2.2.3. `conditionalize-table` will return an unmodified probability table if the conditionalized variable is not present in the table. However, its caller `conditionalize` will always modify the SPO's conditional information, even if the probability table is unchanged.

At first, conditionalization appears very similar to projection on variables (sections 2.2.2, 4.2) - both remove a column and multiple rows from the table. Conditionalization, however, has no need to identify and merge duplicate rows. Projection on variables must compare against other rows to determine duplicates; conditionalization simply chooses which rows to remove based on the new condition, and has no need to compare with other rows in the same table. Thus, conditionalization does not have the same complexity problems as projection on variables, and is expected to run in linear time.

4.4 Cartesian Product

The implementation of Cartesian product ("`TIMES`"), specified in section 2.2.4, is shown in figure 13.

Note that `TIMES` has two double `FOR` loops: one iterating each possible pair of SPOs; another iterating each possible pair of rows in each SPO's table. If

```
declare function spdb:conditionalize-table($table as element(table), $name as xs:string, $value as xs:string) as element(table) {
  let $num := spdb:table-column-num($table/variable/name, $name)
  (: If var isn't in table, return the original table.
  This isn't silent failure, but an irrelevant conditionalization. :)
  return if (fn:empty($num)) then $table else
  let $newrows := $table/row[exists(val[$num][text() eq $value])]
  let $oldprob := sum(for $i in $table/row/P return xs:decimal($i))
  let $newprob := sum(for $i in $newrows/P return xs:decimal($i))
  let $probmult := if ($newprob gt 0) then $oldprob div $newprob else 1
  return element table {
    element variable {$table/variable/name[position() ne $num]},
    for $row in $newrows
      let $p := xs:decimal($row/P) * $probmult
      return element row {$row/val[position() ne $num], element P{$p}}
  }
};

declare function spdb:conditionalize($spo as element(spo), $name as xs:string, $value as xs:string) as element(spo) {
  element spo {$spo/@*,
    $spo/context,
    spdb:conditionalize-table($spo/table, $name, $value),
    element conditional {
      $spo/conditional/*,
      element elem {element name{$name}, element val{$value}}
    }
  }
};
```

Figure 12: Conditionalization implementation.

```
declare function spdb:table-product($table1 as element(table), $table2 as element(table)) as element(table) {
  if (not(spdb:common-vars-empty($table1, $table2))) then () else
  element table {
    element variable {$table1/variable/name, $table2/variable/name},
    spdb:rows-product($table1/row, $table2/row)
  }
};

declare function spdb:rows-product($rows1 as element(row)*, $rows2 as element(row)*) as element(row)* {
  for $row1 in $rows1
    for $row2 in $rows2
      let $prob := element P { $row1/P/number() * $row2/P/number() }
      return element row {$row1/val, $row2/val, $prob}
};

declare function spdb:sp-product($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  element spos {
    for $spo1 in $spos1/spo
      for $spo2 in $spos2/spo
        where (spdb:product-compatible($spo1, $spo2))
        return spdb:product($spo1, $spo2)
  }
};
```

Figure 13: Partial Cartesian product (”TIMES”) implementation.

all SPOs are product-compatible and have many rows, this operation can take a very long time. These nested loops are difficult to avoid: the specification of `TIMES` (section 2.2.4) requires that it return $n*m$ SPOs given two relations of size n and m; and $r*s$ rows per SPO given a pair of SPOs with row counts r and s. These nested loops perform exactly this many iterations.

4.5 Join

Our `JOIN` implementation is shown in figure 14. `JOIN` has complexity problems similar to those of `TIMES`. Two double `FOR` loops are required, for each pair of SPOS and each pair of rows.

As with `TIMES`, all SPOs are join-compatible and have many rows, this operation can take a very long time. These nested loops are difficult to avoid: the specification of `JOIN` (section 2.2.5) requires that it return $n * m$ SPOs given two relations of size n and m; and $r * s$ rows per SPO given a pair of SPOs with row counts r and s. These nested loops perform exactly this many iterations.

Our implementation removes all common columns from the `JOIN`ed SPOs by conditionalizing on variables common to both SPOs (`spdb:leftjoin-conditionalize()`, figure 14). Without any common columns, the two SPOs are product-compatible and we perform a Cartesian product.

The implementation of `RIGHTJOIN` is shown in figure 17: the parameters are simply switched to form a left join. This means variables from the second parameter to a `RIGHTJOIN` are displayed before the variables from the first

```
declare function spdb:leftjoin-conditionalize-table($table as element(table), $vars as element(name)*, $vals as element(val)*) as element(table) {
  if (fn:empty($vars)) then $table else
  let $var := $vars[1]
  let $val := $vals[1]
  return spdb:leftjoin-conditionalize-table(element table {$table/@*,
    element variable {$table/variable/name[. ne $var]},
    spdb:conditionalize-table($table, $var, $val)/row
  }, $vars[position() ne 1], $vals[position() ne 1])
};

declare function spdb:table-leftjoin($table1 as element(table), $table2 as element(table)) as element(table)? {
  let $common := spdb:common-vars($table1/variable, $table2/variable)/name
  return if (fn:empty($common)) then () else
  element table {$table1/@*,
    element variable {$table1/variable/name, $table2/variable/name[not(. = $common)]},
    for $row1 in $table1/row
      let $values := spdb:row-column-vals($table1/variable/name, $row1, $common/text())
      let $condtable2 := spdb:leftjoin-conditionalize-table($table2, $common, $values)
      return spdb:rows-product($row1, $condtable2/row)
  }
};
```

Figure 14: Partial JOIN implementation. Some parts are very similar to TIMES and have been omitted from this figure.

```
declare function spdb:sp-rightjoin($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  spdb:sp-leftjoin($spos2, $spos1)
};
```

Figure 15: Right-join implementation, based on left-join.

```
declare function spdb:table-leftmix($table1 as element(table), $table2 as element(table)) as element(table) {
  let $common := spdb:common-vars($table1/variable, $table2/variable)/name
  return if (fn:empty($common))
  then spdb:table-product($table1, $table2)
  else spdb:table-le       $table1, $table2)
};
```

Figure 16: Mix implementation.

parameter in the probability table, which may be unintuitive. This is acceptable, however, as the order in which variables are displayed in the results is not significant.

The names `LEFTJOIN` and `RIGHTJOIN` are used in SPOQL queries, instead of `LEFT JOIN` and `RIGHT JOIN`, to avoid making large changes to the existing SPOQL parser.

4.6 Mix

As described in section 2.2.6, `MIX` simply chooses between `JOIN` and `TIMES` for each pair of SPOs, depending on their compatibility. Note that, while

```
declare function spdb:sp-rightmix($spos1 as element(
    spos), $spos2 as element(spos)) as element(spos)
    {
  spdb:sp-leftmix($spos2, $spos1)
};
```

Figure 17: Right-mix implementation, based on left-mix.

`JOIN` and `TIMES` are mutually exclusive, there exist SPOs which are neither join- nor product-compatible that will not be included in `MIX`'s results.

The implementation of `RIGHTMIX` is similar to that of `RIGHTJOIN` (4.5), as shown in figure 17.

The names `LEFTMIX` and `RIGHTMIX` are used in SPOQL queries, instead of `LEFT MIX` and `RIGHT MIX`, for the same reason as the similar care for `JOIN` operations (section 4.5): to avoid making large changes to the existing SPOQL parser.

5 Experiments

5.1 Design

A major motivation for our native XML SPDBMS backend was improving system performance: the existing Oracle implementation was reported to have trouble with both memory and processing time. We need to justify performance improvements with appropriate experiments. Our experiments focus on measuring the time required to process queries.

5.1.1 Experimental Variables

Factors we planned to vary in experiments include:

- Relation size: The number of SPOs included in each relation
- Number of random variables: The average number of variables in each SPO
- Domain size: the typical number of possible values for each random variable
- Join-compatibility and Product-Compatibility: the number of results expected for JOIN, TIMES, and MIX operations. (Recall the explanations of join- and product-compatibility from sections 2.2.5 and 2.2.4.)

Later, we discovered that execution speed was also dependent on our *XQuery module size* as it influenced the time required to compile the module for every query, and we set up further experiments to show this. See section 5.2.1 for details.

5.1.2 Construction and Execution

A custom XML generator was written to provide test data for experiments varying the above parameters. An existing set of experimental data and queries, provided with the existing source code from [9], was included and adapted to our needs.

To collect timing information, the SPDB server was instrumented to log two pieces of data - time spent performing database queries, and time spent processing the data outside of the database. All times recorded are in milliseconds. Graphs are generated using Gnuplot.

Experiments are automated by a collection of custom shell scripts. Given a set of test data, a list of files containing experimental queries, and a properties file specifying which database backend to use (ExistDB or Oracle): we start the SPDB server, clean out the database, populate the database with test data required throughout the experiment, and run all queries specified for the experiment. After experiments are complete, graphs of the results are generated using custom Gnuplot scripts.

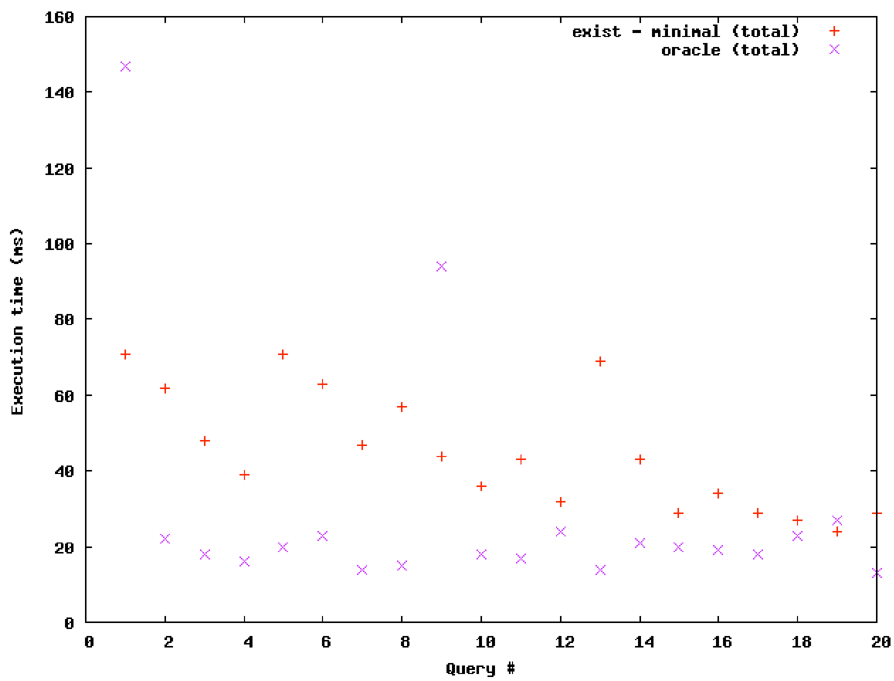


Figure 18: Baseline experiments: running the simplest possible query (SELECT * FROM Relation) on an empty database. Exist tends to lag behind Oracle a bit, but the difference is reasonable.

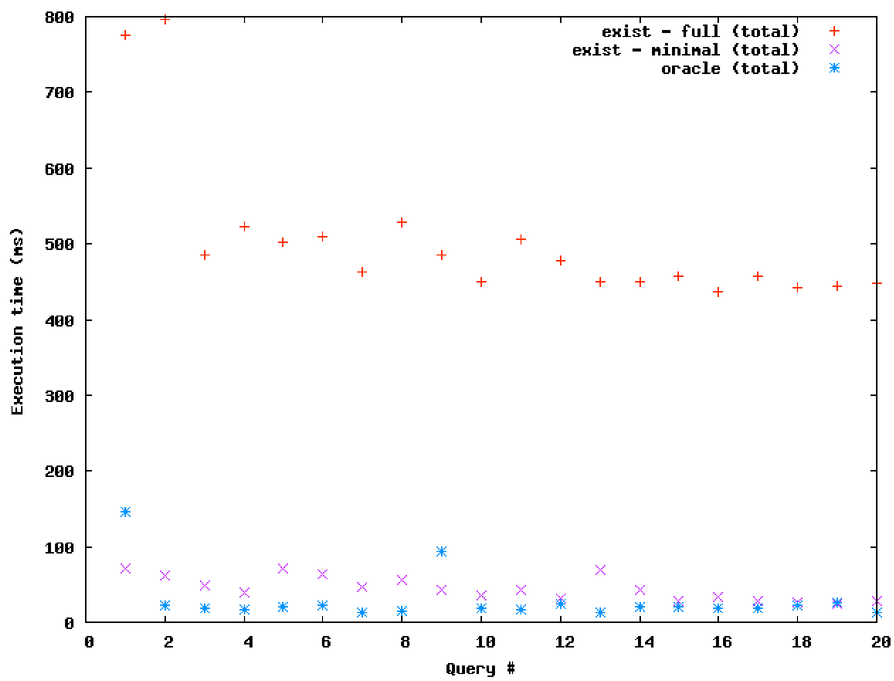


Figure 19: Baseline experiments: running the simplest possible query (SELECT * FROM Relation) on an empty database. "Exist(minimal)" uses the smallest possible XQuery module to run this set of experiments, whereas "Exist(full)" uses the full module supporting all SPDBMS operations. Using the full XQuery module dramatically slows Exist.

5.2 Results and Analysis

5.2.1 Module Compilation Speed

Early experiments showed our ExistDB backend to be dramatically slower than Oracle, even for a baseline measurement - the simplest possible operation, `SELECT * FROM Relation` performed on an empty database. Further experiments, shown in figures 18 and 19, showed that reducing our XQuery module to only a minimum set of functions needed to support this operation dramatically improved our results. ExistDB appeared to be recompiling our XQuery module once for *every XQuery execution*, or once per SPOQL statement!

This module does not change between runs and should only need to be compiled once to be used for all queries. ExistDB's built-in function libraries suggest that this is possible, but we were unable to determine how to implement this ourselves. ExistDB's documentation suggests that this should be done automatically[1]:

> XQuery modules executed via the REST interface, the XQueryServlet or XQueryGenerator are automatically cached: the compiled expression will be added to an internal pool of prepared queries. The next time a query or module is loaded from the same location, it will not be compiled again. Instead, the already compiled code is reused.

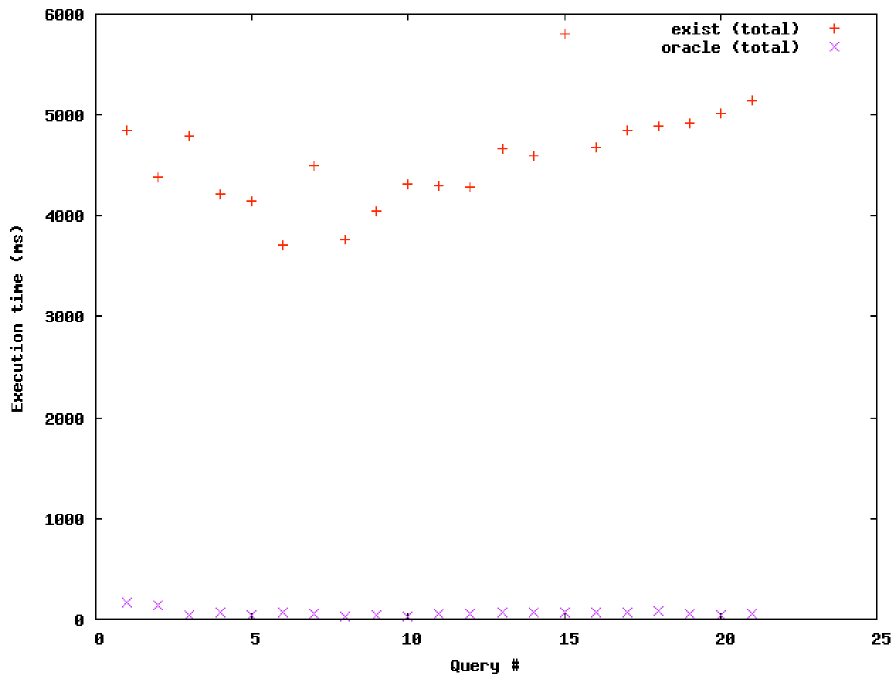


Figure 20: Projection on variables experiments. Performance compared to the original relational implementation was disappointing.

5.2.2 Results for Projection on Variables

Figure 20 shows our XQuery backend's results for the projection on variables operation (section 2.2.2). These results were particularly disappointing. Section 4.2 discusses the relevant code.

When projected columns are removed from the result, some rows will have the same values for all remaining columns - probabilities for these rows must be merged. Identifying and merging these duplicate rows using XQuery is expensive: each *pair of rows* must be checked for equality, leading to a complexity of $O(n^2)$ for n rows. It's certainly possible to optimize this case in XQuery. We could manually construct indexes on discrete values for each column in such a way that each row has a unique integer index based on the values of its random variables. When one column is removed and indexes recalculated, rows with identical values would also have identical indexes. Exist is much more likely to be able to optimize the comparison of a single integer, improving performance and complexity. This has not yet been implemented.

5.2.3 Memory Usage

While experiments focused on execution time rather than memory usage, it was apparent that our XQuery implementation handled memory more efficiently than the existing Oracle implementation. Oracle ran out of memory on some experiments with large joins and products; ExistDB was able to complete the same experiments. This problem with the Oracle implementation existed in previous implementations, and our focus is on the new ExistDB implementation, so we made no attempt to diagnose or repair the Oracle backend's memory problems.

6 Conclusion and Future Work

We have presented a native XML database backend to an XML-centric semistructured probabilistic database. Our original goal of improved execution time over the previous Oracle backend was not met. We analyzed the problems behind these disappointing results and developed several ideas for relevant future work:

- Improve the execution time of all queries by recompiling and reloading our XQuery module, or by compiling only the functions required for a particular operation.
- Port our XQuery module to other XML database backends. Compare their performance.
- Improve the execution time of projection on variables, perhaps using the indexing scheme described in section .
- Implement the `AND` optimization discussed in section 4.1.2.
- Improve the execution time of join and product operations.

7 Bibliography

References

[1] Existdb documentation: Xquery documentation: Xquery caching.

[2] Anderson, D., Reed, A., Rosson, E., and Sideropoulos, A. Native xml support for semistructured probabalistic data management. Tech. rep., Computer Science Department, California Polytechnic State University, Dec 2007.

[3] Anderson, D., Reed, A., Rosson, E., and Sideropoulos, A. Native xml support for semistructured probabalistic data management. Tech. rep., Computer Science Department, California Polytechnic State University, Oct 2007.

[4] Anderson, D., Reed, A., Rosson, E., and Sideropoulos, A. Native xml support for semistructured probabalistic data management: A progress report. Tech. rep., Computer Science Department, California Polytechnic State University, Nov 2007.

[5] Barbará, D., Garcia-Molina, H., and Porter, D. The management of probabilistic data. *IEEE Trans. on Knowledge and Data Engineering 4* (1992), 487–502.

[6] Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., and Suciu, D. Mystiq: A system for finding more answers by using probabilities.

[7] Cavallo, R., and Pittarelli, M. The theory of probabilistic databases. pp. 71–81.

[8] Cavallo, R., and Pittarelli, M. The theory of probabilistic databases. In *Proc. VLDB'87* (1987), pp. 71–81.

[9] Dekhtyar, A., Gutti, P., and Mathias, K. Structured queries for semistructured probabilistic data. *ACM TDM* (2006).

[10] Pearl, J. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[11] Zhao, W., Dekhtyar, A., and Goldsmith, J. A framework for management of semistructured probabilistic data. *Journal of Intelligent Information Systems 25*, 3 (2004), 293–332.

A XQuery

```
module namespace spdb="http://www.csc.calpoly.edu/~
   erosson/spdb";

declare function spdb:sp-select-var($spos as element
   (spos), $varname as xs:string)
as element(spos)
{
    element spos {
        $spos/spo[table/variable/name = $varname]
    }
};

(: @author Dustin Anderson :)
declare function spdb:sp-select-table($sp as element
   (spos),
$varname as xs:string, $value as xs:string, $comp as
    xs:string) as element(spos)
{
let $ret := (
for $spo in $sp//variable/name[text() eq $varname]/
   ancestor::spo
   let $n := ( for $var in $spo//variable
      for $e at $i in $var/name
         return if ( $e eq $varname) then $i else ()
      )
   let $rows := (
     if ($comp eq "=") then
      $spo//row/val[position()=$n and (text() eq
         $value)]/parent::row
     else if ($comp eq "!=") then
      $spo//row/val[position()=$n and (text() ne
         $value)]/parent::row
     else if ($comp eq "<") then
      $spo//row/val[position()=$n and (text() lt
         $value)]/parent::row
```

```
      else if ($comp eq "<=") then
       $spo//row/val[position()=$n and (text() le $value)]/parent::row
      else if ($comp eq ">") then
       $spo//row/val[position()=$n and (text() gt $value)]/parent::row
      else if ($comp eq ">=") then
       $spo//row/val[position()=$n and (text() ge $value)]/parent::row
      else ()
        )
    let $variables := $rows/parent::*/variable
    let $context := $rows/ancestor::*/context
    let $table := element table {$variables, $rows}
    let $conditional := $rows/ancestor::*/conditional
    let $spo := <spo path="{$spo/@path}">{$context, $table, $conditional}</spo>
    return $spo
    )
    return <spos>{$ret}</spos>
};

(: @author Dustin Anderson :)
declare function spdb:sp-select-prob($sp as element(spos), $value as xs:string,
$comp as xs:string) as element(spos)
{
  let $spos := (
  for $node in $sp//spo
   let $context := $node/descendant::context
   let $conditional := $node/descendant::conditional
   let $variables := $node/descendant::variable
   let $rows := (
     if ($comp eq "=") then
       $node/descendant::P[text() eq $value]/parent::row
     else if ($comp eq "!=") then
```

```
        $node/descendant::P[text() ne $value]/parent::row
      else if ($comp eq "<") then
        $node/descendant::P[text() lt $value]/parent::row
      else if ($comp eq "<=") then
        $node/descendant::P[text() le $value]/parent::row
      else if ($comp eq ">") then
        $node/descendant::P[text() gt $value]/parent::row
      else if ($comp eq ">=") then
        $node/descendant::P[text() ge $value]/parent::row
      else ()
      )
    let $table := element table {$variables, $rows}
    let $spo := <spo path="{$node/@path}">{$context, $table, $conditional}</spo>
    return $spo
    )
    return <spos> {$spos} </spos>
};

(: @author Dustin Anderson :)
declare function spdb:sp-select-conditional($sp as element(spos),
$varname as xs:string, $value as xs:string, $comp as xs:string) as element(spos)
{
let $thespos := (
    for $node in $sp//spo
     let $elem := (
      if ($comp eq "=") then
       $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val eq $value)]
      else if ($comp eq "!=") then
```

```
      $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val ne $value)]
     else if ($comp eq "<") then
      $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val lt $value)]
     else if ($comp eq "<=") then
      $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val le $value)]
     else if ($comp eq ">") then
      $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val gt $value)]
     else if ($comp eq ">=") then
      $node/descendant::conditional/child::elem[(descendant::name/text() eq $varname) and (descendant::val ge $value)]
     else()
      )
    let $spo := $elem/ancestor::spo
    return $spo
   )
return <spos> {$thespos} </spos>
};

(: @author Dustin Anderson :)
declare function spdb:sp-select-context($sp as element(spos),
$varname as xs:string, $value as xs:string, $comp as xs:string) as element(spos)
{
let $thespos := (
  for $node in $sp//spo
   let $elem := (
     if ($comp eq "=") then
```

```
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val eq $value]
      else if ($comp eq "!=") then
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val ne $value]
      else if ($comp eq "<") then
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val lt $value]
      else if ($comp eq "<=") then
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val le $value]
      else if ($comp eq ">") then
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val gt $value]
      else if ($comp eq ">=") then
     $node/descendant::context/child::elem[descendant::name/text() eq $varname and descendant::val ge $value]
    else ()
    )
    let $spo := $elem/ancestor::spo
    return $spo
    )

return <spos>{$thespos}</spos>
};

(: @author Dustin Anderson :)
declare function spdb:sp-project-context($sp as element(spos),
$varnames as xs:string*) as element(spos)
{
let $thespos := (
```

```
 for $varname in $varnames
  for $node in $sp//spo
   let $conditional := $node/descendant::conditional
   let $variables := $node/descendant::variable
   let $rows := $node/descendant::row
   let $table := element table {$variables, $rows}
   let $elem := $node/descendant::context/child::
      elem[descendant::name/text() eq $varname]
   let $context := <context> {$elem} </context>
   let $spo := <spo path="{$node/@path}">{$context,
      $table, $conditional}</spo>
   return $spo
   )

   return <spos>{$thespos}</spos>
};

(: @author Dustin Anderson :)
declare function spdb:sp-project-conditional($sp as
   element(spos),
$varnames as xs:string*) as element(spos)
{
let $thespos := (
 for $varname in $varnames
  for $node in $sp//spo
   let $context := $node/descendant::context
   let $variables := $node/descendant::variable
   let $rows := $node/descendant::row
   let $table := element table {$variables, $rows}
   let $elem := $node/descendant::conditional/child
      ::elem[descendant::name/text() eq $varname]
   let $conditional := <conditional> {$elem} </
      conditional>
   let $spo := <spo path="{$node/@path}">{$context,
      $table, $conditional}</spo>
   return $spo
   )
return <spos>{$thespos}</spos>
```

```
};









(: http://www.xqueryfunctions.com/xq/functx_value-
   intersect.html :)
declare namespace functx = "http://www.functx.com";
declare function functx:value-intersect
  ( $arg1 as xs:anyAtomicType* ,
    $arg2 as xs:anyAtomicType* )  as xs:
       anyAtomicType* {

  distinct-values($arg1[.=$arg2])
 } ;

declare function functx:is-node-in-sequence-deep-
   equal
  ( $node as node()? ,
    $seq as node()* )  as xs:boolean {

   some $nodeInSeq in $seq satisfies deep-equal(
      $nodeInSeq,$node)
 } ;

declare function functx:distinct-deep
  ( $nodes as node()* )  as node()* {

    for $seq in (1 to count($nodes))
    return $nodes[$seq][not(functx:is-node-in-
       sequence-deep-equal(
                          .,$nodes[position() < $seq
                              ]))]
```

```
 } ;


declare function spdb:QName($code as xs:string) as xs:QName {
  fn:QName("http://www.csc.calpoly.edu/~erosson/spdb/errors", $code)
};
declare function spdb:error($code as xs:string, $msg as xs:string) as node() {
  fn:error(spdb:QName($code), fn:concat("SPO error: ", $msg))
};
declare function spdb:error($msg as xs:string) as node() {
  spdb:error("???", $msg)
};

declare function spdb:relation($name as xs:string) as element(spos) {
  let $ret := document($name)/spos
  return if (fn:empty($ret)) then spdb:error("relation_dne",fn:concat("Relation does not exist: ", $name)) else
  $ret
};

declare function spdb:cond-normalize($cond as element(conditional)) as element(conditional) {
  element conditional {
    for $elem in $cond/elem
    order by $elem/name
    return $elem
  }
};

(:
Match a SPO's <cond> blocks. Order doesn't matter.
```

```
:)
declare function spdb:cond-match($cond1 as element(conditional), $cond2 as element(conditional)) as xs:boolean {
  deep-equal(spdb:cond-normalize($cond1), spdb:cond-normalize($cond2))
};

declare function spdb:common-vars($vars1 as element(variable), $vars2 as element(variable)) as element(variable) {
  element variable {
    for $name in functx:value-intersect($vars1/name/text(), $vars2/name/text())
    return element name {$name}
  }
};

declare function spdb:common-vars-empty($elem1 as element(), $elem2 as element()) as xs:boolean {
  fn:empty(spdb:common-vars($elem1//variable, $elem2//variable)/name)
};

declare function spdb:product-compatible($spo1 as element(spo), $spo2 as element(spo)) as xs:boolean {
  if (spdb:cond-match($spo1/conditional, $spo2/conditional))
  then spdb:common-vars-empty($spo1, $spo2)
  else false()
};

declare function spdb:join-compatible($spo1 as element(spo), $spo2 as element(spo)) as xs:boolean {
  if (spdb:cond-match($spo1/conditional, $spo2/conditional))
```

```
  then not(spdb:common-vars-empty($spo1, $spo2))
  else false()
};

declare function spdb:table-product($table1 as element(table), $table2 as element(table)) as element(table) {
  if (not(spdb:common-vars-empty($table1, $table2))) then () else
  element table {
    element variable {$table1/variable/name, $table2/variable/name},
    spdb:rows-product($table1/row, $table2/row)
  }
};

declare function spdb:rows-product($rows1 as element(row)*, $rows2 as element(row)*) as element(row)* {
  for $row1 in $rows1
    for $row2 in $rows2
      let $prob := element P { $row1/P/number() * $row2/P/number() }
      return element row {$row1/val, $row2/val, $prob}
};

declare function spdb:context-merge($con1 as element(context), $con2 as element(context)) as element(context) {
  element context {
    functx:distinct-deep(($con1/elem, $con2/elem))
  }
};

declare function spdb:binary-spo-wrap($spo1 as element(spo), $spo2 as element(spo), $table as element(table)) as element(spo) {
```

```
  element spo {
    spdb:context-merge($spo1/context, $spo2/context),
    $table,
    $spo1/conditional
  }
};

declare function spdb:product($spo1 as element(spo), $spo2 as element(spo)) as element(spo)? {
  if (not(spdb:product-compatible($spo1, $spo2))) then () else
  spdb:binary-spo-wrap($spo1, $spo2, spdb:table-product($spo1/table, $spo2/table))
};

declare function spdb:sp-product($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  element spos {
    for $spo1 in $spos1/spo
      for $spo2 in $spos2/spo
        return spdb:product($spo1, $spo2)
  }
};


declare function spdb:table-column-num($vars as element(name)*, $name as xs:string) as xs:integer? {
  spdb:table-column-nums($vars, $name)
};
declare function spdb:table-column-nums($vars as element(name)*, $names as xs:string*) as xs:integer* {
  for $var at $index in $vars
    where $var/text() = $names
    return $index
```

```
};

declare function spdb:table-column-vals-by-nums($rows as element(row)*, $num as xs:integer) as element(val)* {
  $rows/val[position() eq $num]
};

declare function spdb:row-column-vals-by-nums($row as element(row), $nums as xs:integer*) as element(val)* {
  $row/val[position() = $nums]
};

declare function spdb:table-column-vals($table as element(table), $name as xs:string) as element(val)* {
  let $num := spdb:table-column-num($table/variable/name, $name)
  return if (fn:empty($num)) then $table/row/val else
  spdb:table-column-vals-by-nums($table/row, $num)
};

declare function spdb:row-column-vals($vars as element(name)*, $row as element(row), $names as xs:string*) as element(val)* {
  spdb:row-column-vals-by-nums($row, spdb:table-column-nums($vars, $names))
};

declare function spdb:conditionalize-table($table as element(table), $name as xs:string, $value as xs:string) as element(table) {
  let $num := spdb:table-column-num($table/variable/name, $name)
  (: If var isn't in table, return the original table.
```

```
  This isn't silent failure, but an irrelevant conditionalization. :)
  return if (fn:empty($num)) then $table else
  let $newrows := $table/row[exists(val[$num][text() eq $value])]
  let $oldprob := sum(for $i in $table/row/P return xs:decimal($i))
  let $newprob := sum(for $i in $newrows/P return xs:decimal($i))
  let $probmult := if ($newprob gt 0) then $oldprob div $newprob else 1
  return element table {
    element variable {$table/variable/name[position() ne $num]},
    for $row in $newrows
      let $p := xs:decimal($row/P) * $probmult
      return element row {$row/val[position() ne $num], element P{$p}}
  }
};

declare function spdb:conditionalize($spo as element(spo), $name as xs:string, $value as xs:string) as element(spo) {
  element spo {$spo/@*,
    $spo/context,
    spdb:conditionalize-table($spo/table, $name, $value),
    element conditional {
      $spo/conditional/*,
      element elem {element name{$name}, element val{$value}}
    }
  }
};

declare function spdb:sp-conditionalize($spos as element(spos), $name as xs:string, $value as xs:
```

```
    string) as element(spos) {
  element spos {$spos/@*,
    for $spo in $spos/spo
      return spdb:conditionalize($spo, $name, $value
        )
  }
};
declare function spdb:leftjoin-conditionalize-table(
    $table as element(table), $vars as element(name)
    *, $vals as element(val)*) as element(table) {
  if (fn:empty($vars)) then $table else
  let $var := $vars[1]
  let $val := $vals[1]
  return spdb:leftjoin-conditionalize-table(element
      table {$table/@*,
    element variable {$table/variable/name[. ne $var
      ]},
    spdb:conditionalize-table($table, $var, $val)/
      row
  }, $vars[position() ne 1], $vals[position() ne 1])
};

declare function spdb:table-leftjoin($table1 as
    element(table), $table2 as element(table)) as
    element(table)? {
  let $common := spdb:common-vars($table1/variable,
      $table2/variable)/name
  return if (fn:empty($common)) then () else
  element table {$table1/@*,
    element variable {$table1/variable/name, $table2
      /variable/name[not(. = $common)]},
    for $row1 in $table1/row
      let $values := spdb:row-column-vals($table1/
        variable/name, $row1, $common/text())
      let $condtable2 := spdb:leftjoin-
        conditionalize-table($table2, $common,
        $values)
```

```
      return spdb:rows-product($row1, $condtable2/row)
  }
};

declare function spdb:leftjoin($spo1 as element(spo), $spo2 as element(spo)) as element(spo)? {
  if (not(spdb:join-compatible($spo1, $spo2))) then () else
  spdb:binary-spo-wrap($spo1, $spo2, spdb:table-leftjoin($spo1/table, $spo2/table))
};

declare function spdb:sp-leftjoin($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  element spos {
    for $spo1 in $spos1/spo
      for $spo2 in $spos2/spo
        return spdb:leftjoin($spo1, $spo2)
  }
};

declare function spdb:sp-rightjoin($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  spdb:sp-leftjoin($spos2, $spos1)
};

declare function spdb:table-leftmix($table1 as element(table), $table2 as element(table)) as element(table) {
  let $common := spdb:common-vars($table1/variable, $table2/variable)/name
  return if (fn:empty($common)) then spdb:table-product($table1, $table2) else spdb:table-l        $table1, $table2)
};
```

```

declare function spdb:leftmix($spo1 as element(spo), $spo2 as element(spo)) as element(spo)? {
  if (not(spdb:cond-match($spo1/conditional, $spo2/conditional))) then () else
  spdb:binary-spo-wrap($spo1, $spo2, spdb:table-leftmix($spo1/table, $spo2/table))
};

declare function spdb:sp-leftmix($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  element spos {
    for $spo1 in $spos1/spo
      for $spo2 in $spos2/spo
        return spdb:leftmix($spo1, $spo2)
  }
};

declare function spdb:sp-rightmix($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  spdb:sp-leftmix($spos2, $spos1)
};

declare function spdb:sp-where-or($spos1 as element(spos), $spos2 as element(spos)) as element(spos) {
  element spos {$spos1/@*, functx:distinct-deep(($spos1/spo, $spos2/spo)) }
};

(: Existdb-specific: http://exist.sourceforge.net/update_ext.html :)
declare function spdb:sp-update-insert($dest as element(spos), $data as element(spos)) as node()* {
  update insert $data/spo into $dest
```

```
};

declare function spdb:sp-update-delete($dest as xs:string, $matches as element(spos)) as node()* {
  for $match in $matches/spo
    for $spo in spdb:relation($dest)/spo
      where deep-equal($match, $spo)
      return update delete $spo
};

declare function spdb:rows-project-var($rawrows as element(row)*, $visiblenums as xs:integer*) as element(row)* {
  let $rowsviscols := for $row in $rawrows
    return element row {$row/@*,
      $row/val[position() = $visiblenums],
      $row/P
    }
  return for $row1 at $i in $rowsviscols
    (: where no previous rows have identical values :)
    where fn:empty($rowsviscols[position() lt $i][not(val != $row1/val)])
    return element row {$row1/@*,
      $row1/val,
      element P {$row1/P/@*,
        (: sum all rows with identical values. skip previous rows: we checked those already :)
        sum(for $p in $rowsviscols[position() ge $i][not(val != $row1/val)]/P
          return xs:decimal($p))
      }
    }
};

declare function spdb:table-project-var($table as element(table), $visiblevars as xs:string*) as
```

```
    element(table) {
  let $visiblenums := spdb:table-column-nums($table/variable/name, $visiblevars)
  return element table {$table/@*,
    element variable {$table/variable/@*,
      $table/variable/name[position() = $visiblenums]
    },
    spdb:rows-project-var($table/row, $visiblenums)
  }
};

declare function spdb:project-var($spo as element(spo), $visiblevars as xs:string*) as element(spo) {
  element spo {$spo/@*,
    $spo/context,
    spdb:table-project-var($spo/table, $visiblevars),
    $spo/conditional
  }
};

declare function spdb:sp-project-var($spos as element(spos), $visiblevars as xs:string*) as element(spos) {
  element spos {$spos/@*,
    for $spo in $spos/spo
      return spdb:project-var($spo, $visiblevars)
  }
};

declare function spdb:rows-project-var-2($rawrows as element(row)*, $visiblenums as xs:integer*) as element(row)* {
  let $rowsviscols := for $row in $rawrows
    return element row {$row/@*,
      $row/val[position() = $visiblenums],
```

```
      $row/P
    }
  return for $row1 at $i in $rowsviscols
    (: where no previous rows have identical values :)
    where fn:empty($rowsviscols[position() lt $i][deep-equal(val, $row1/val)])
    return element row {$row1/@*,
      $row1/val,
      element P {$row1/P/@*,
        (: sum all rows with identical values. skip previous rows: we checked those already :)
        sum(for $p in $rowsviscols[position() ge $i][deep-equal(val, $row1/val)]/P
          return xs:decimal($p))
      }
    }
};

declare function spdb:table-project-var-2($table as element(table), $visiblevars as xs:string*) as element(table) {
  let $visiblenums := spdb:table-column-nums($table/variable/name, $visiblevars)
  return element table {$table/@*,
    element variable {$table/variable/@*,
      $table/variable/name[position() = $visiblenums]
    },
    spdb:rows-project-var-2($table/row, $visiblenums)
  }
};

declare function spdb:project-var-2($spo as element(spo), $visiblevars as xs:string*) as element(spo) {
```

```
  element spo {$spo/@*,
    $spo/context,
    spdb:table-project-var-2($spo/table,
       $visiblevars),
    $spo/conditional
  }
};

declare function spdb:sp-project-var-2($spos as
   element(spos), $visiblevars as xs:string*) as
   element(spos) {
  element spos {$spos/@*,
    for $spo in $spos/spo
      return spdb:project-var-2($spo, $visiblevars)
  }
};

(: @author Dustin Anderson :)
declare function spdb:sp-project-var-3($sp as
   element(spos), $visiblevars as xs:string*) as
   element(spos)
{
let $hiddenvars := $sp//variable/name[. !=
   $visiblevars]

let $thespos := (
          (:Only iterate over spos that have CS113
             in them as <name> elements:)
for $varname in $hiddenvars
for $node in $sp/descendant::variable/name[text() eq
    $varname]/ancestor::spo
          (: Get the position of 'CS113' in the <
             names> :)
    let $n := ( for $w in $node//variable
      for $e at $i in $w/child::name
                        return
                        if ( $e eq $varname) then $i
                           else ()
```

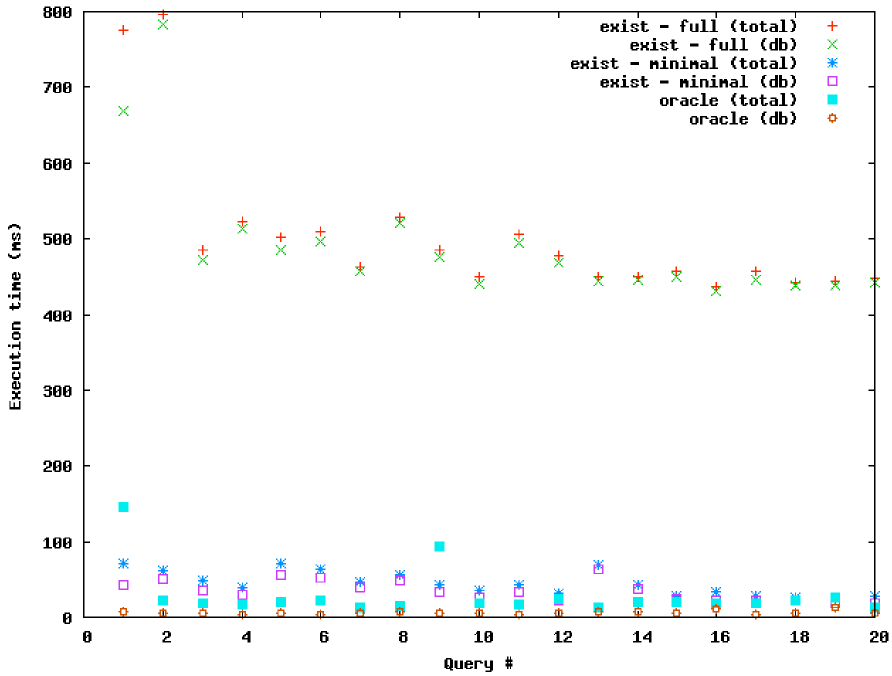
```
               )

            (:Remove $varnames from the <variable>
               element:)
         let $variable := (
            for $v in $node//table/child::variable
            let $name := $v/child::name[position() !=
                $n]
                return element variable{$name}
         )

            (:Remove elements in table that match
               with position of $varnames:)
         let $rowswithP := (
            for $r in $node//table/row
            let $vals := $r/child::val[position() !=
               $n]
                return element row{$vals, $r/child::P
                   }
         )

            (:Remove elements in table that match
               with position of $varnames:)
         let $rowswithoutP := (
            for $r in $node//table/row
            let $vals := $r/child::val[position() !=
               $n]
                return element row{$vals}
         )

         let $table := (
         for $xx at $ii in $rowswithoutP
          for $yy at $jj in $rowswithoutP
            return if (($xx eq $yy) and ($jj > $ii))
               then
              element row {$xx/val, <P>{($rowswithP[
                 position()=$ii]/P + $rowswithP[
                 position()=$jj]/P)}</P>}
```

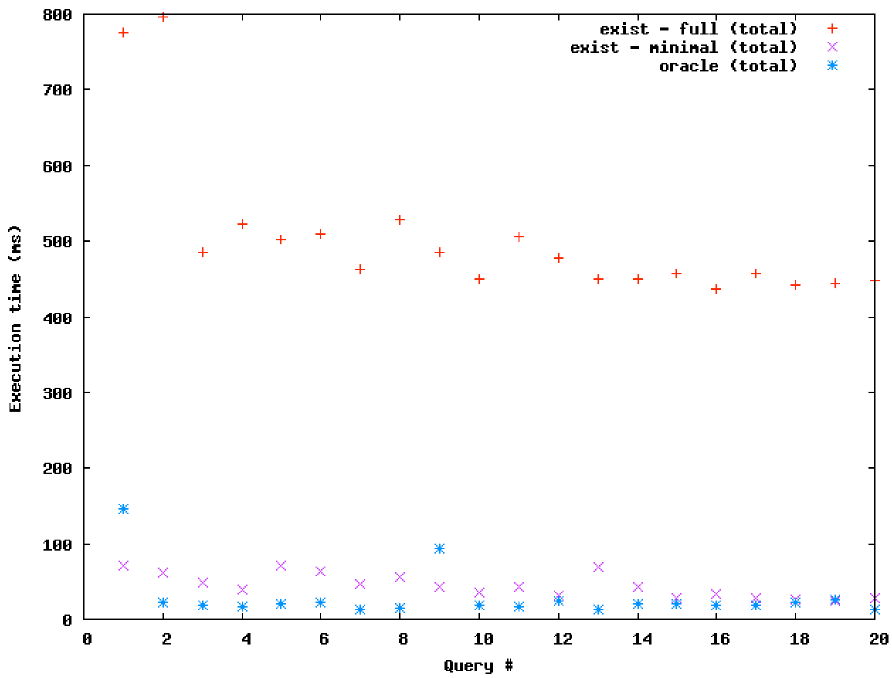
```
            else
                ()
          )

         let $context := $node//context
         let $conditional := $node//conditional

    return <spo path="{$node/@path}">{$context, <
       table>{$variable, $table}</table>,
       $conditional}</spo>
     )

  return <spos>{$thespos}</spos>
};
```

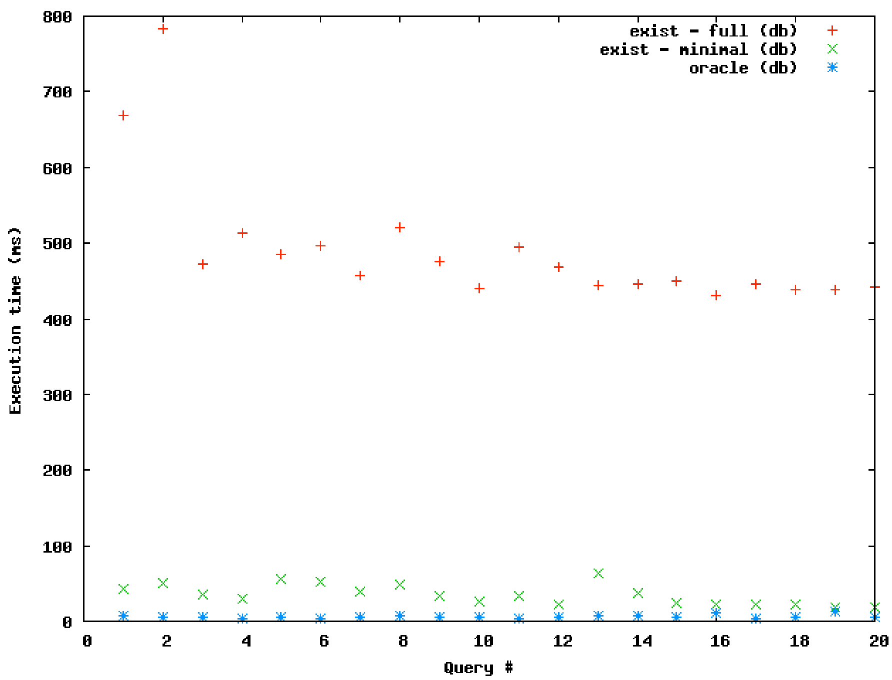
B Experimental Results

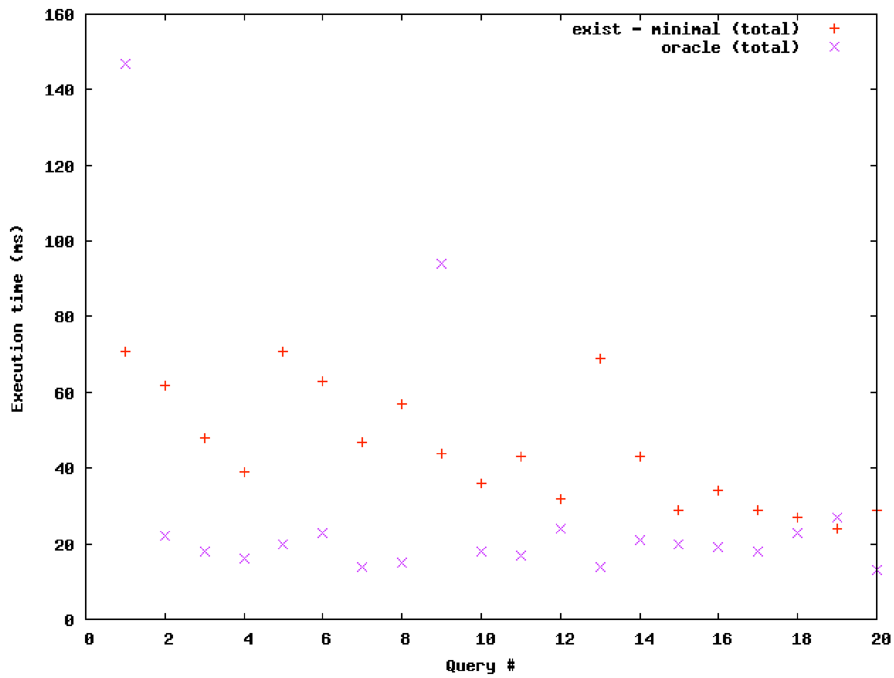
B.1 Baseline experiments

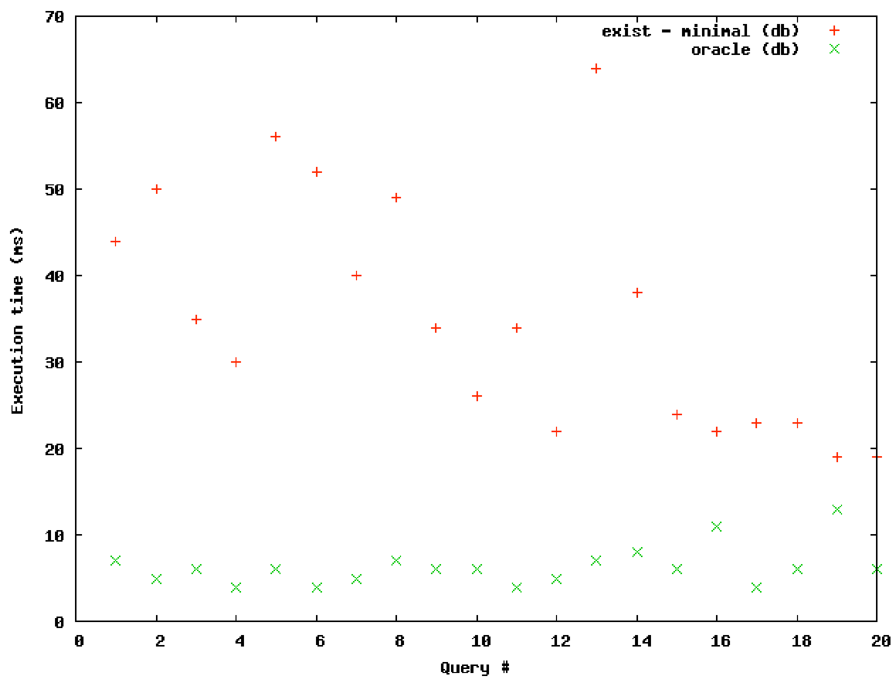
Comparison




exist - full (total)
exist - minimal (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
2
4
6
8
10
12
14
16
18
20


exist - full (db)
exist - minimal (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
2
4
6
8
10
12
14
16
18
20


exist - minimal (total)
oracle (total)
Execution time (ms)
Query #
0
20
40
60
80
100
120
140
160
0
2
4
6
8
10
12
14
16
18
20


exist - minimal (db)
oracle (db)
Execution time (ms)
Query #
0
10
20
30
40
50
60
70
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from empty: (time: 147 db: 7)
QUERY XML select * from empty: (time: 22 db: 5)
QUERY XML select * from empty: (time: 18 db: 6)
QUERY XML select * from empty: (time: 16 db: 4)
QUERY XML select * from empty: (time: 20 db: 6)
QUERY XML select * from empty: (time: 23 db: 4)
QUERY XML select * from empty: (time: 14 db: 5)
QUERY XML select * from empty: (time: 15 db: 7)
QUERY XML select * from empty: (time: 94 db: 6)
QUERY XML select * from empty: (time: 18 db: 6)
QUERY XML select * from empty: (time: 17 db: 4)
QUERY XML select * from empty: (time: 24 db: 5)
QUERY XML select * from empty: (time: 14 db: 7)
QUERY XML select * from empty: (time: 21 db: 8)
QUERY XML select * from empty: (time: 20 db: 6)
QUERY XML select * from empty: (time: 19 db: 11)
QUERY XML select * from empty: (time: 18 db: 4)
QUERY XML select * from empty: (time: 23 db: 6)
QUERY XML select * from empty: (time: 27 db: 13)
QUERY XML select * from empty: (time: 13 db: 6)
```

Exist (minimal) Data

```
QUERY XML select * from empty: (time: 71 db: 44)
QUERY XML select * from empty: (time: 62 db: 50)
QUERY XML select * from empty: (time: 48 db: 35)
QUERY XML select * from empty: (time: 39 db: 30)
QUERY XML select * from empty: (time: 71 db: 56)
QUERY XML select * from empty: (time: 63 db: 52)
QUERY XML select * from empty: (time: 47 db: 40)
QUERY XML select * from empty: (time: 57 db: 49)
QUERY XML select * from empty: (time: 44 db: 34)
QUERY XML select * from empty: (time: 36 db: 26)
QUERY XML select * from empty: (time: 43 db: 34)
QUERY XML select * from empty: (time: 32 db: 22)
QUERY XML select * from empty: (time: 69 db: 64)
QUERY XML select * from empty: (time: 43 db: 38)
QUERY XML select * from empty: (time: 29 db: 24)
QUERY XML select * from empty: (time: 34 db: 22)
```

```
QUERY XML select * from empty: (time: 29 db: 23)
QUERY XML select * from empty: (time: 27 db: 23)
QUERY XML select * from empty: (time: 24 db: 19)
QUERY XML select * from empty: (time: 29 db: 19)
```

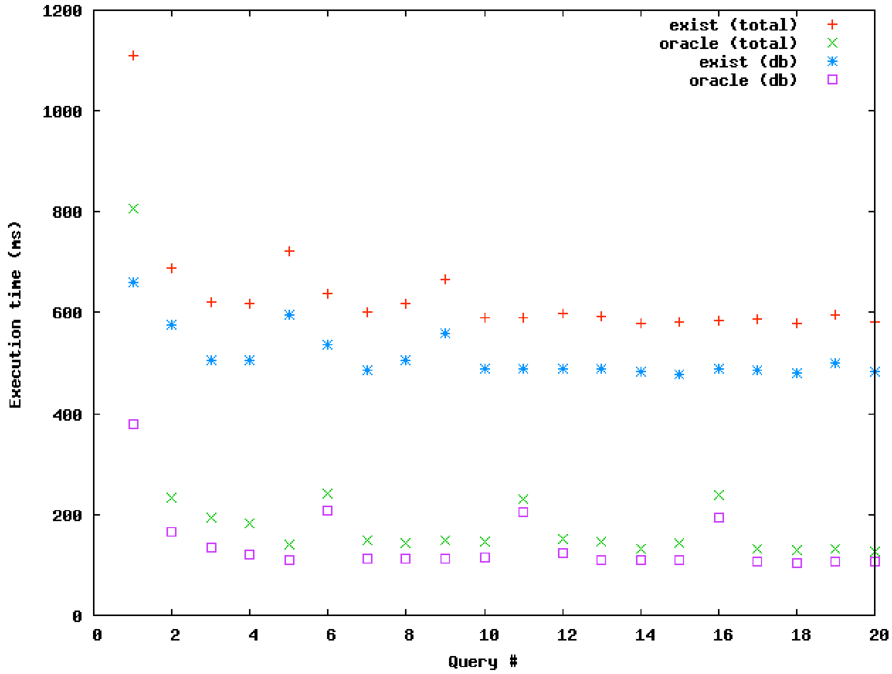
Exist (full) Data

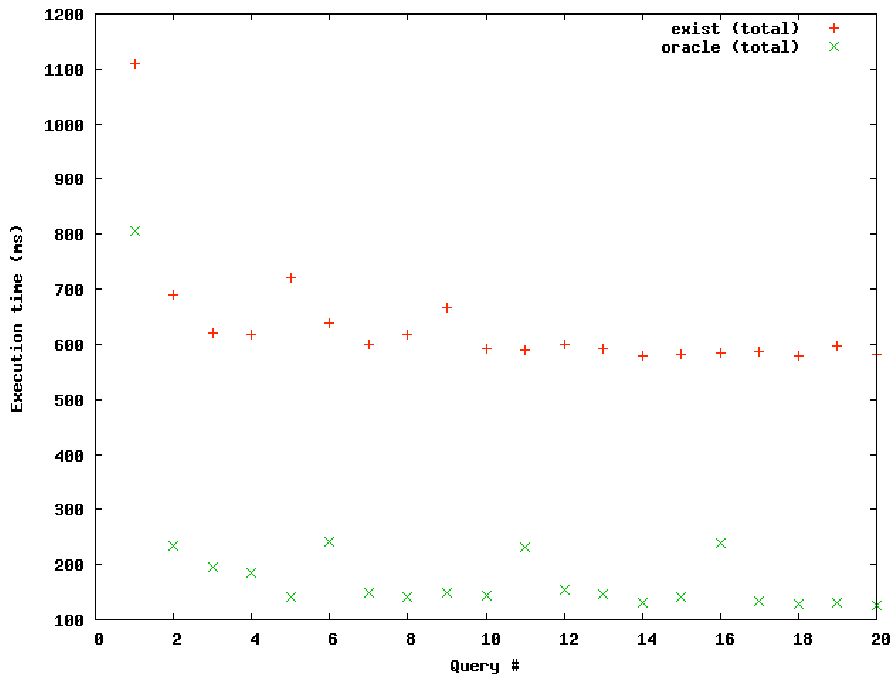
```
QUERY XML select * from empty: (time: 776 db: 668)
QUERY XML select * from empty: (time: 796 db: 784)
QUERY XML select * from empty: (time: 486 db: 472)
QUERY XML select * from empty: (time: 523 db: 513)
QUERY XML select * from empty: (time: 502 db: 486)
QUERY XML select * from empty: (time: 509 db: 497)
QUERY XML select * from empty: (time: 463 db: 458)
QUERY XML select * from empty: (time: 528 db: 520)
QUERY XML select * from empty: (time: 486 db: 476)
QUERY XML select * from empty: (time: 450 db: 440)
QUERY XML select * from empty: (time: 506 db: 494)
QUERY XML select * from empty: (time: 477 db: 468)
QUERY XML select * from empty: (time: 450 db: 444)
QUERY XML select * from empty: (time: 449 db: 445)
QUERY XML select * from empty: (time: 457 db: 450)
QUERY XML select * from empty: (time: 436 db: 430)
QUERY XML select * from empty: (time: 457 db: 445)
QUERY XML select * from empty: (time: 443 db: 438)
QUERY XML select * from empty: (time: 444 db: 439)
QUERY XML select * from empty: (time: 447 db: 442)
```

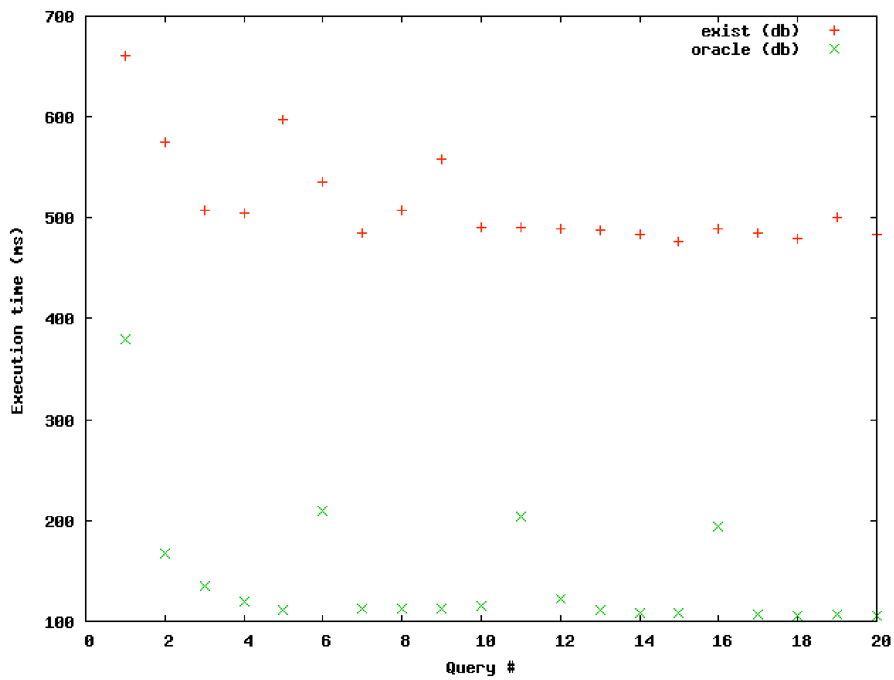
B.2 2 variables, 100 SPOs

B.2.1 Simple Selection

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First: (time: 806 db: 379)
QUERY XML select * from First: (time: 234 db: 167)
QUERY XML select * from First: (time: 195 db: 135)
QUERY XML select * from First: (time: 184 db: 120)
QUERY XML select * from First: (time: 141 db: 111)
QUERY XML select * from Second: (time: 241 db: 209)
QUERY XML select * from Second: (time: 148 db: 113)
QUERY XML select * from Second: (time: 142 db: 113)
QUERY XML select * from Second: (time: 148 db: 113)
QUERY XML select * from Second: (time: 145 db: 116)
QUERY XML select * from Third: (time: 231 db: 204)
QUERY XML select * from Third: (time: 153 db: 123)
QUERY XML select * from Third: (time: 146 db: 111)
QUERY XML select * from Third: (time: 131 db: 109)
QUERY XML select * from Third: (time: 142 db: 109)
QUERY XML select * from Fourth: (time: 239 db: 194)
QUERY XML select * from Fourth: (time: 133 db: 107)
QUERY XML select * from Fourth: (time: 129 db: 105)
QUERY XML select * from Fourth: (time: 131 db: 107)
QUERY XML select * from Fourth: (time: 127 db: 106)
```

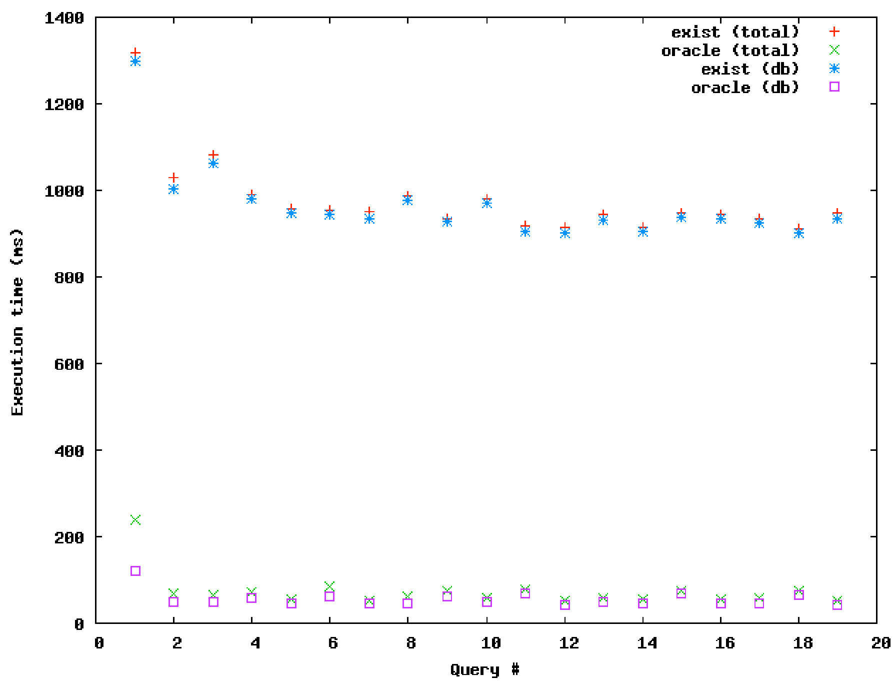
Exist Data

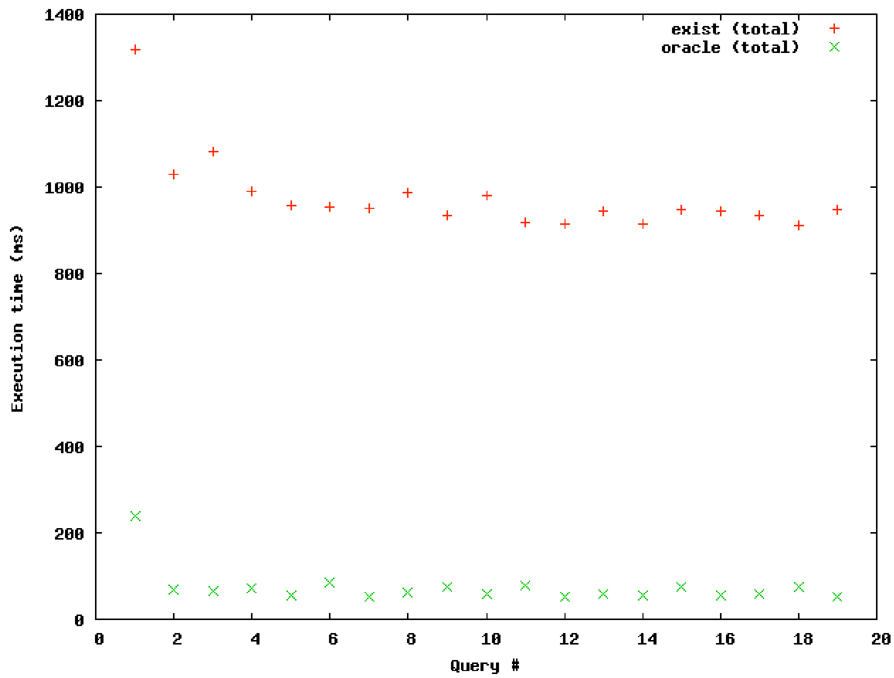
```
QUERY XML select * from First: (time: 1109 db: 660)
QUERY XML select * from First: (time: 689 db: 575)
QUERY XML select * from First: (time: 620 db: 507)
QUERY XML select * from First: (time: 617 db: 505)
QUERY XML select * from First: (time: 722 db: 597)
QUERY XML select * from Second: (time: 639 db: 536)
QUERY XML select * from Second: (time: 601 db: 485)
QUERY XML select * from Second: (time: 618 db: 507)
QUERY XML select * from Second: (time: 667 db: 558)
QUERY XML select * from Second: (time: 591 db: 490)
QUERY XML select * from Third: (time: 589 db: 490)
QUERY XML select * from Third: (time: 599 db: 489)
QUERY XML select * from Third: (time: 593 db: 488)
QUERY XML select * from Third: (time: 580 db: 483)
QUERY XML select * from Third: (time: 581 db: 477)
QUERY XML select * from Fourth: (time: 584 db: 489)
```

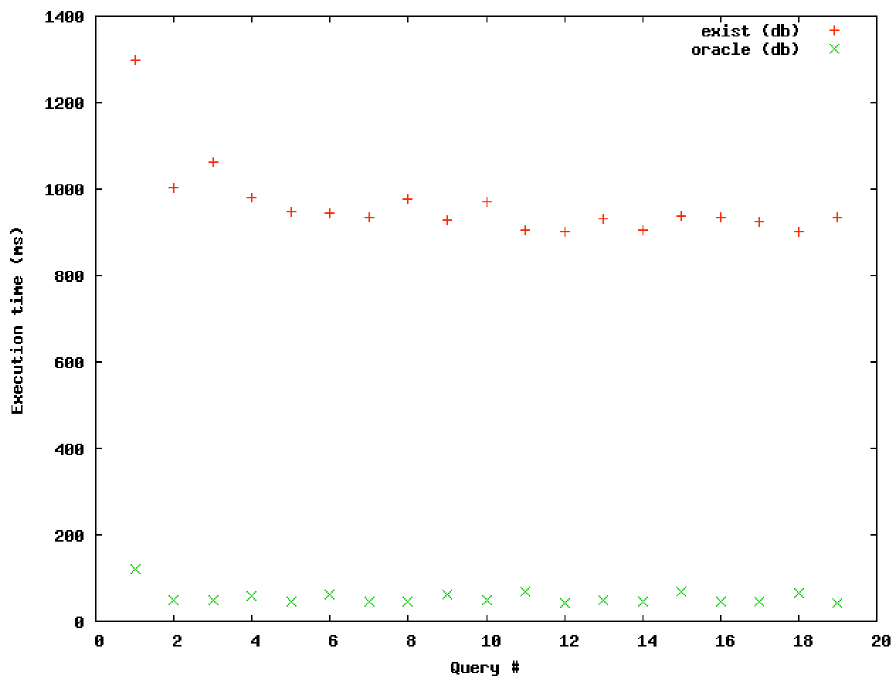
```
17 QUERY XML select * from Fourth: (time: 588 db: 485)
18 QUERY XML select * from Fourth: (time: 579 db: 480)
19 QUERY XML select * from Fourth: (time: 597 db: 500)
20 QUERY XML select * from Fourth: (time: 581 db: 483)
```

B.2.2 Select on Context

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First where cnt.college =XX:
    (time: 238 db: 120)
QUERY XML select * from First where cnt.comments=OO:
    (time: 69 db: 50)
QUERY XML select * from First where cnt.year=1976: (
   time: 64 db: 49)
QUERY XML select * from First where cnt.year=1995: (
   time: 71 db: 60)
QUERY XML select * from First where cnt.comments=GG:
    (time: 55 db: 47)
QUERY XML select * from Second where cnt.semester =
   PP: (time: 86 db: 63)
QUERY XML select * from Second where cnt.major=XX: (
   time: 54 db: 45)
QUERY XML select * from Second where cnt.major=II: (
   time: 62 db: 46)
QUERY XML select * from Second where cnt.year=1999:
   (time: 74 db: 61)
QUERY XML select * from Second where cnt.semester=FF
   : (time: 58 db: 48)
QUERY XML select * from Third where cnt.major=MM: (
   time: 80 db: 70)
QUERY XML select * from Third where cnt.instructor=
   HH: (time: 53 db: 42)
QUERY XML select * from Third where cnt.semester=TT:
    (time: 60 db: 50)
QUERY XML select * from Third where cnt.comments=PP:
    (time: 56 db: 45)
QUERY XML select * from Fourth where cnt.comments=WW
   : (time: 77 db: 68)
QUERY XML select * from Fourth where cnt.year=1981:
   (time: 56 db: 45)
QUERY XML select * from Fourth where cnt.major=DD: (
   time: 58 db: 45)
QUERY XML select * from Fourth where cnt.major=WW: (
   time: 76 db: 64)
QUERY XML select * from Fourth where cnt.semester=GG
   : (time: 54 db: 44)
```

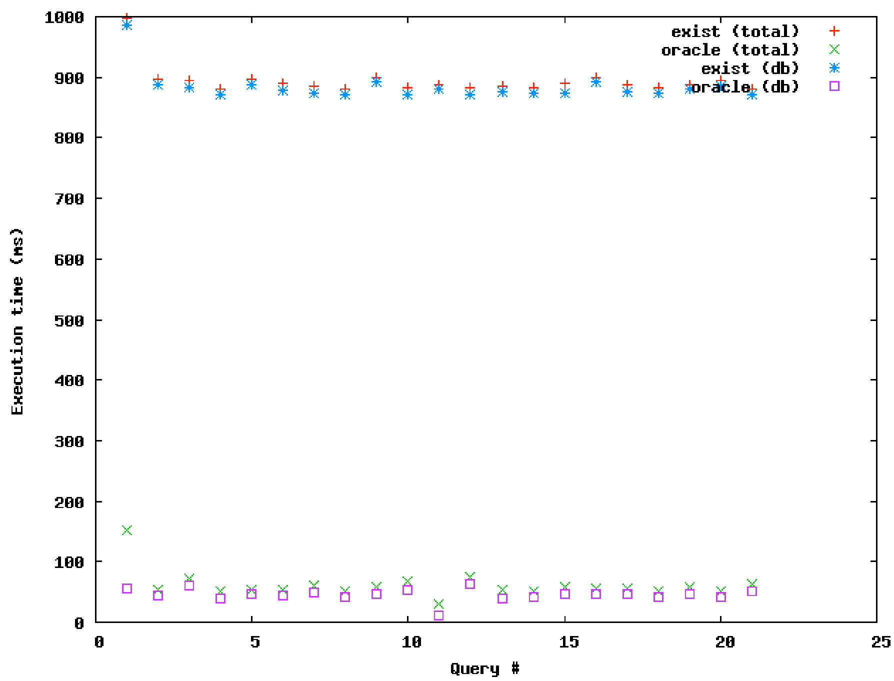
Exist Data

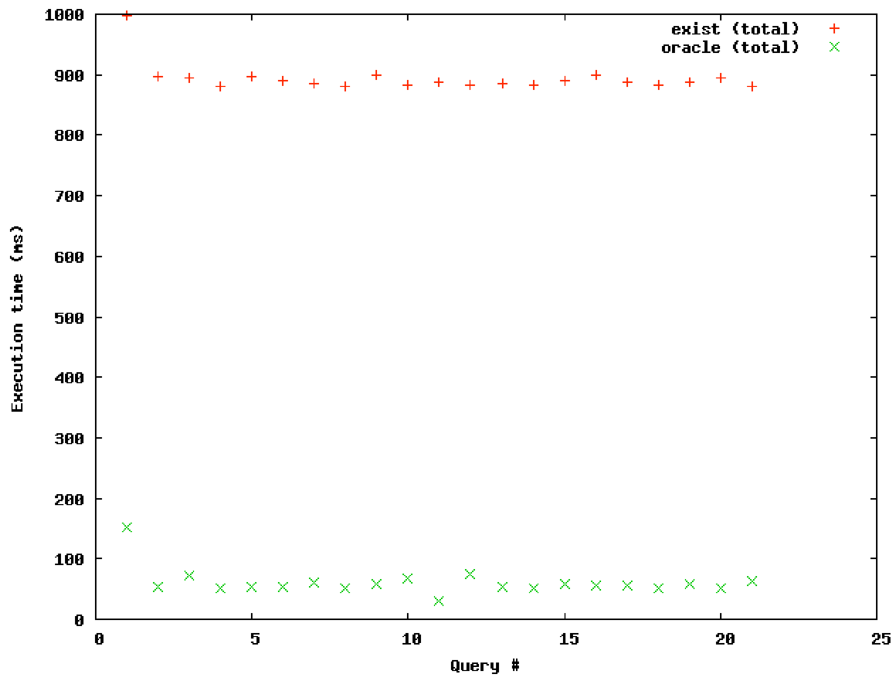
```
QUERY XML select * from First where cnt.college =XX: (time: 1318 db: 1298)
QUERY XML select * from First where cnt.comments=OO: (time: 1029 db: 1004)
QUERY XML select * from First where cnt.year=1976: (time: 1083 db: 1061)
QUERY XML select * from First where cnt.year=1995: (time: 989 db: 980)
QUERY XML select * from First where cnt.comments=GG: (time: 958 db: 948)
QUERY XML select * from Second where cnt.semester = PP: (time: 955 db: 943)
QUERY XML select * from Second where cnt.major=XX: (time: 950 db: 936)
QUERY XML select * from Second where cnt.major=II: (time: 986 db: 977)
QUERY XML select * from Second where cnt.year=1999: (time: 936 db: 928)
QUERY XML select * from Second where cnt.semester=FF: (time: 979 db: 971)
QUERY XML select * from Third where cnt.major=MM: (time: 918 db: 906)
QUERY XML select * from Third where cnt.instructor=HH: (time: 916 db: 902)
QUERY XML select * from Third where cnt.semester=TT: (time: 944 db: 931)
QUERY XML select * from Third where cnt.comments=PP: (time: 914 db: 904)
QUERY XML select * from Fourth where cnt.comments=WW: (time: 947 db: 937)
QUERY XML select * from Fourth where cnt.year=1981: (time: 943 db: 934)
QUERY XML select * from Fourth where cnt.major=DD: (time: 934 db: 926)
QUERY XML select * from Fourth where cnt.major=WW: (time: 913 db: 902)
```

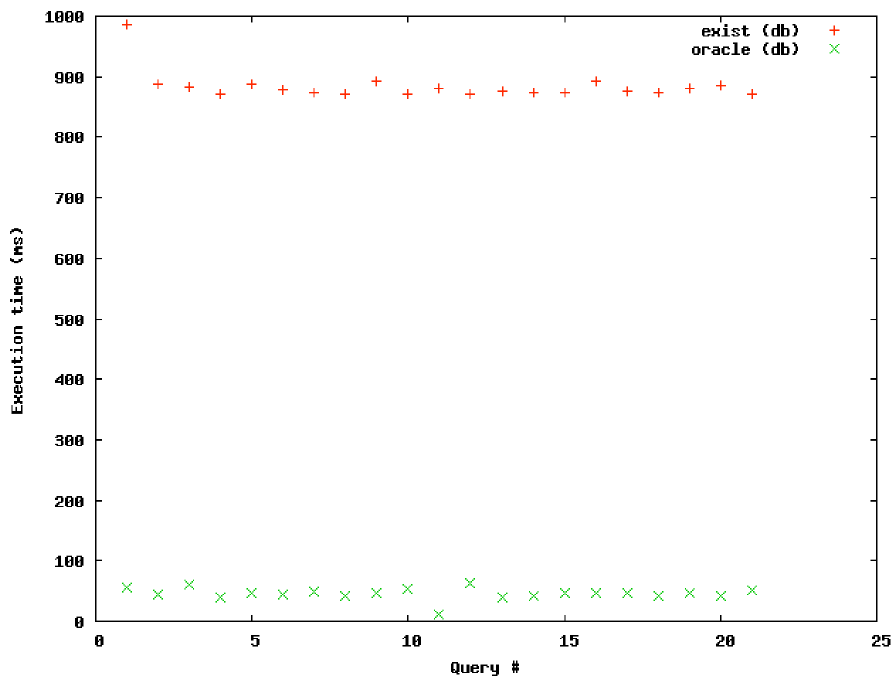
```
19 QUERY XML select * from Fourth where cnt.semester=GG
   : (time: 947 db: 935)
```

B.2.3 Select on Conditional

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
1000
0
5
10
15
20
25


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
1000
0
5
10
15
20
25

Oracle Data

```
QUERY XML select * from First where cnd.CS120=B: (
   time: 153 db: 57)
QUERY XML select * from First where cnd.CS101=B: (
   time: 53 db: 44)
QUERY XML select * from First where cnd.CS106=A: (
   time: 73 db: 60)
QUERY XML select * from First where cnd.CS143=B: (
   time: 52 db: 40)
QUERY XML select * from First where cnd.CS103=B: (
   time: 55 db: 47)
QUERY XML select * from Second where cnd.CS143=B: (
   time: 53 db: 44)
QUERY XML select * from Second where cnd.CS146=A: (
   time: 62 db: 50)
QUERY XML select * from Second where cnd.CS135=A: (
   time: 51 db: 43)
QUERY XML select * from Second where cnd.CS102=A: (
   time: 59 db: 47)
QUERY XML select * from Second where cnd.CS127=B: (
   time: 67 db: 55)
QUERY XML select * from Second where cnd.CS144=A: (
   time: 30 db: 12)
QUERY XML select * from Third where cnd.CS144=A: (
   time: 75 db: 63)
QUERY XML select * from Third where cnd.CS138=A: (
   time: 54 db: 40)
QUERY XML select * from Third where cnd.CS112=B: (
   time: 51 db: 42)
QUERY XML select * from Third where cnd.CS140=A: (
   time: 59 db: 48)
QUERY XML select * from Third where cnd.CS122=A: (
   time: 57 db: 46)
QUERY XML select * from Fourth where cnd.CS117=B: (
   time: 57 db: 46)
QUERY XML select * from Fourth where cnd.CS124=A: (
   time: 51 db: 41)
QUERY XML select * from Fourth where cnd.CS107=A: (
   time: 58 db: 46)
```

```
QUERY XML select * from Fourth where cnd.CS128=A: (
    time: 51 db: 42)
QUERY XML select * from Fourth where cnd.CS130=B: (
    time: 63 db: 51)
```

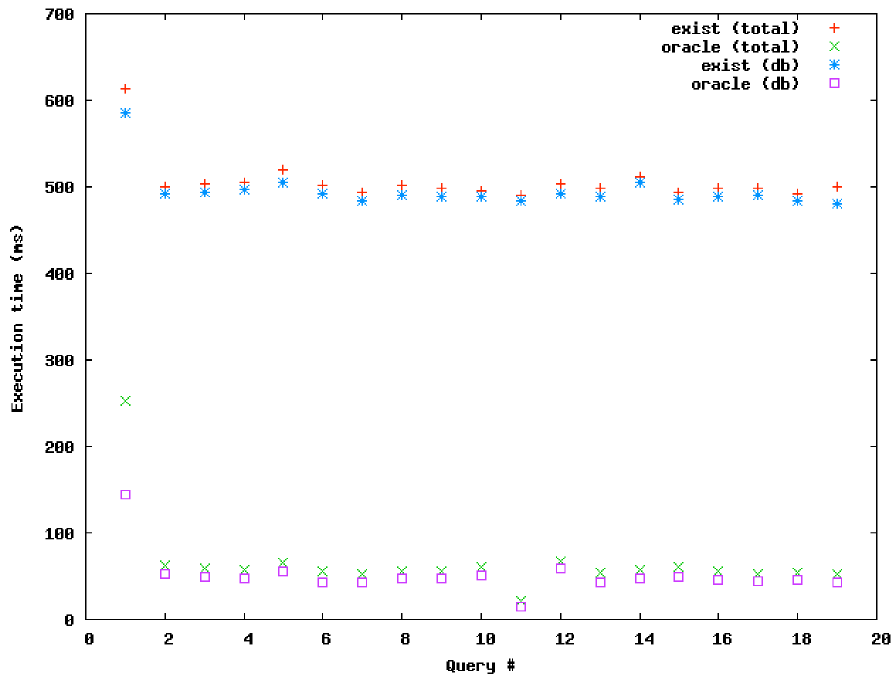
Exist Data

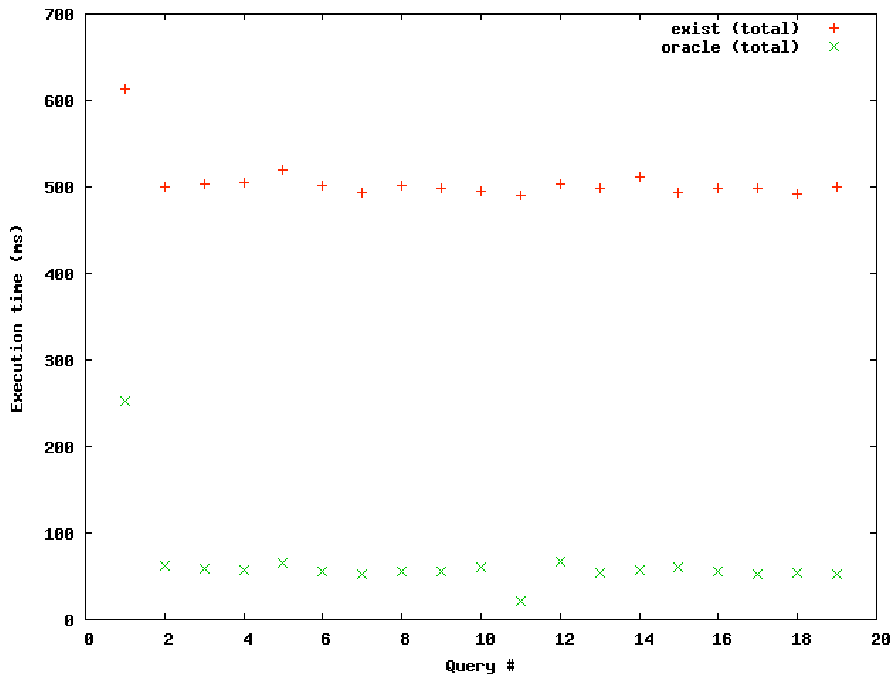
```
QUERY XML select * from First where cnd.CS120=B: (
    time: 998 db: 987)
QUERY XML select * from First where cnd.CS101=B: (
    time: 896 db: 887)
QUERY XML select * from First where cnd.CS106=A: (
    time: 894 db: 883)
QUERY XML select * from First where cnd.CS143=B: (
    time: 880 db: 872)
QUERY XML select * from First where cnd.CS103=B: (
    time: 896 db: 888)
QUERY XML select * from Second where cnd.CS143=B: (
    time: 890 db: 878)
QUERY XML select * from Second where cnd.CS146=A: (
    time: 885 db: 874)
QUERY XML select * from Second where cnd.CS135=A: (
    time: 880 db: 872)
QUERY XML select * from Second where cnd.CS102=A: (
    time: 899 db: 892)
QUERY XML select * from Second where cnd.CS127=B: (
    time: 883 db: 871)
QUERY XML select * from Second where cnd.CS144=A: (
    time: 887 db: 880)
QUERY XML select * from Third where cnd.CS144=A: (
    time: 883 db: 871)
QUERY XML select * from Third where cnd.CS138=A: (
    time: 885 db: 877)
QUERY XML select * from Third where cnd.CS112=B: (
    time: 882 db: 873)
QUERY XML select * from Third where cnd.CS140=A: (
    time: 890 db: 873)
QUERY XML select * from Third where cnd.CS122=A: (
    time: 900 db: 892)
```

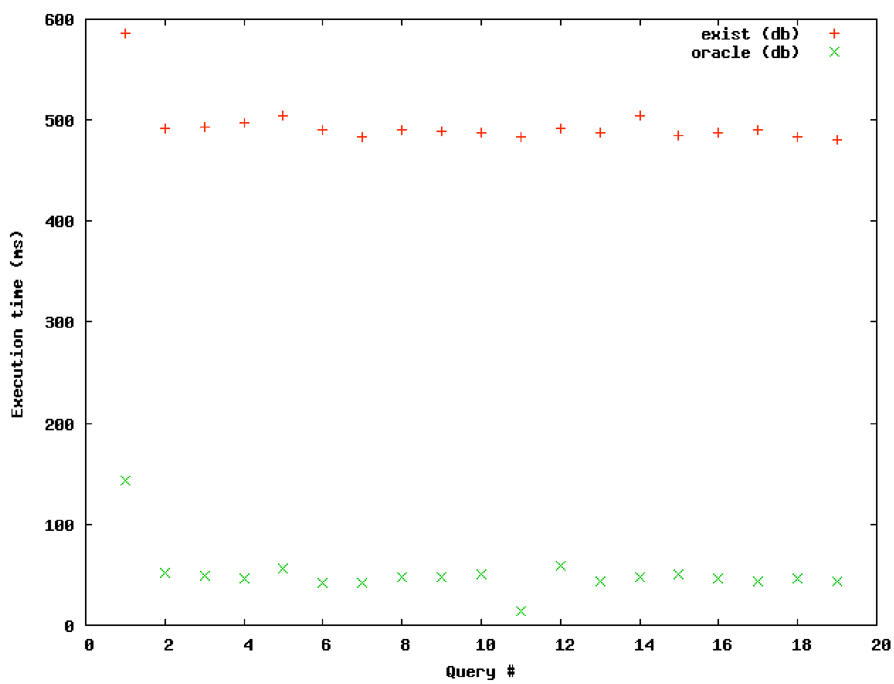
```
QUERY XML select * from Fourth where cnd.CS117=B: (
   time: 888 db: 877)
QUERY XML select * from Fourth where cnd.CS124=A: (
   time: 882 db: 874)
QUERY XML select * from Fourth where cnd.CS107=A: (
   time: 888 db: 880)
QUERY XML select * from Fourth where cnd.CS128=A: (
   time: 894 db: 885)
QUERY XML select * from Fourth where cnd.CS130=B: (
   time: 881 db: 872)
```

B.2.4 Select on Variables

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First where var.CS299 in V:
    (time: 252 db: 144)
QUERY XML select * from First where var.CS205 in V:
    (time: 63 db: 52)
QUERY XML select * from First where var.CS286 in V:
    (time: 59 db: 49)
QUERY XML select * from First where var.CS268 in V:
    (time: 57 db: 47)
QUERY XML select * from Second where var.CS211 in V:
     (time: 65 db: 56)
QUERY XML select * from Second where var.CS141 in V:
     (time: 56 db: 42)
QUERY XML select * from Second where var.CS279 in V:
     (time: 52 db: 42)
QUERY XML select * from Second where var.CS269 in V:
     (time: 56 db: 48)
QUERY XML select * from Second where var.CS251 in V:
     (time: 55 db: 48)
QUERY XML select * from Third where var.CS127 in V:
    (time: 61 db: 51)
QUERY XML select * from Third where var.CS200 in V:
    (time: 21 db: 14)
QUERY XML select * from Third where var.CS112 in V:
    (time: 67 db: 59)
QUERY XML select * from Third where var.CS224 in V:
    (time: 54 db: 43)
QUERY XML select * from Third where var.CS239 in V:
    (time: 57 db: 48)
QUERY XML select * from Fourth where var.CS124 in V:
     (time: 60 db: 50)
QUERY XML select * from Fourth where var.CS157 in V:
     (time: 56 db: 46)
QUERY XML select * from Fourth where var.CS270 in V:
     (time: 52 db: 44)
QUERY XML select * from Fourth where var.CS129 in V:
     (time: 54 db: 46)
QUERY XML select * from Fourth where var.CS166 in V:
     (time: 52 db: 43)
```

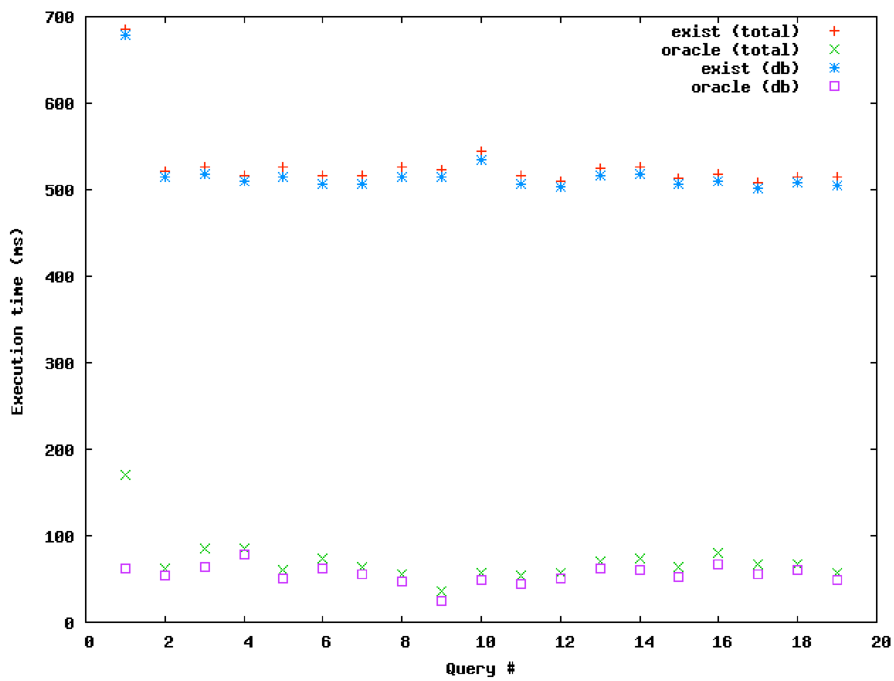
Exist Data

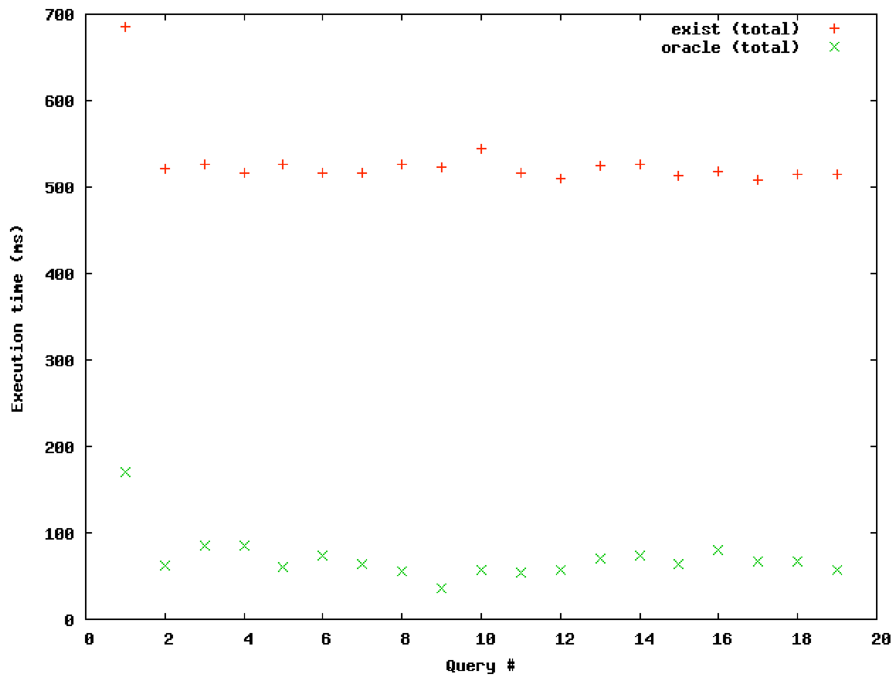
```
QUERY XML select * from First where var.CS299 in V:
    (time: 613 db: 586)
QUERY XML select * from First where var.CS205 in V:
    (time: 500 db: 492)
QUERY XML select * from First where var.CS286 in V:
    (time: 504 db: 493)
QUERY XML select * from First where var.CS268 in V:
    (time: 505 db: 497)
QUERY XML select * from Second where var.CS211 in V:
    (time: 519 db: 505)
QUERY XML select * from Second where var.CS141 in V:
    (time: 502 db: 491)
QUERY XML select * from Second where var.CS279 in V:
    (time: 494 db: 484)
QUERY XML select * from Second where var.CS269 in V:
    (time: 502 db: 490)
QUERY XML select * from Second where var.CS251 in V:
    (time: 498 db: 489)
QUERY XML select * from Third where var.CS127 in V:
    (time: 495 db: 488)
QUERY XML select * from Third where var.CS200 in V:
    (time: 490 db: 483)
QUERY XML select * from Third where var.CS112 in V:
    (time: 503 db: 492)
QUERY XML select * from Third where var.CS224 in V:
    (time: 499 db: 488)
QUERY XML select * from Third where var.CS239 in V:
    (time: 512 db: 505)
QUERY XML select * from Fourth where var.CS124 in V:
    (time: 494 db: 485)
QUERY XML select * from Fourth where var.CS157 in V:
    (time: 498 db: 488)
QUERY XML select * from Fourth where var.CS270 in V:
    (time: 498 db: 490)
QUERY XML select * from Fourth where var.CS129 in V:
    (time: 492 db: 483)
```

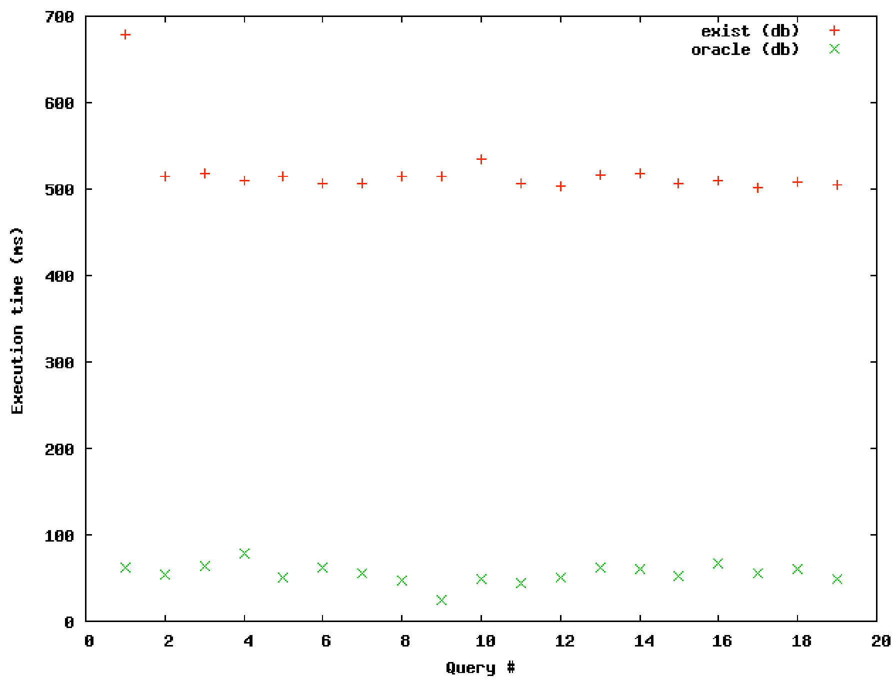
19 QUERY XML select * from Fourth where var.CS166 in V:
(time: 500 db: 481)

B.2.5 Select on Table

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First where tbl.CS239=B: (
   time: 171 db: 62)
QUERY XML select * from First where tbl.CS255=A: (
   time: 63 db: 54)
QUERY XML select * from First where tbl.CS211=A: (
   time: 85 db: 64)
QUERY XML select * from First where tbl.CS217=B: (
   time: 86 db: 78)
QUERY XML select * from First where tbl.CS219=A: (
   time: 60 db: 51)
QUERY XML select * from Second where tbl.CS292=B: (
   time: 73 db: 63)
QUERY XML select * from Second where tbl.CS107=A: (
   time: 64 db: 55)
QUERY XML select * from Second where tbl.CS244=B: (
   time: 55 db: 47)
QUERY XML select * from Second where tbl.CS279=B: (
   time: 36 db: 24)
QUERY XML select * from Second where tbl.CS183=B: (
   time: 58 db: 50)
QUERY XML select * from Third where tbl.CS277=A: (
   time: 54 db: 45)
QUERY XML select * from Third where tbl.CS228=B: (
   time: 58 db: 51)
QUERY XML select * from Third where tbl.CS298=A: (
   time: 71 db: 63)
QUERY XML select * from Third where tbl.CS281=B: (
   time: 73 db: 60)
QUERY XML select * from Third where tbl.CS183=B: (
   time: 64 db: 52)
QUERY XML select * from Fourth where tbl.CS136=A: (
   time: 81 db: 67)
QUERY XML select * from Fourth where tbl.CS251=B: (
   time: 67 db: 56)
QUERY XML select * from Fourth where tbl.CS277=B: (
   time: 68 db: 60)
QUERY XML select * from Fourth where tbl.CS249=B: (
   time: 58 db: 50)
```

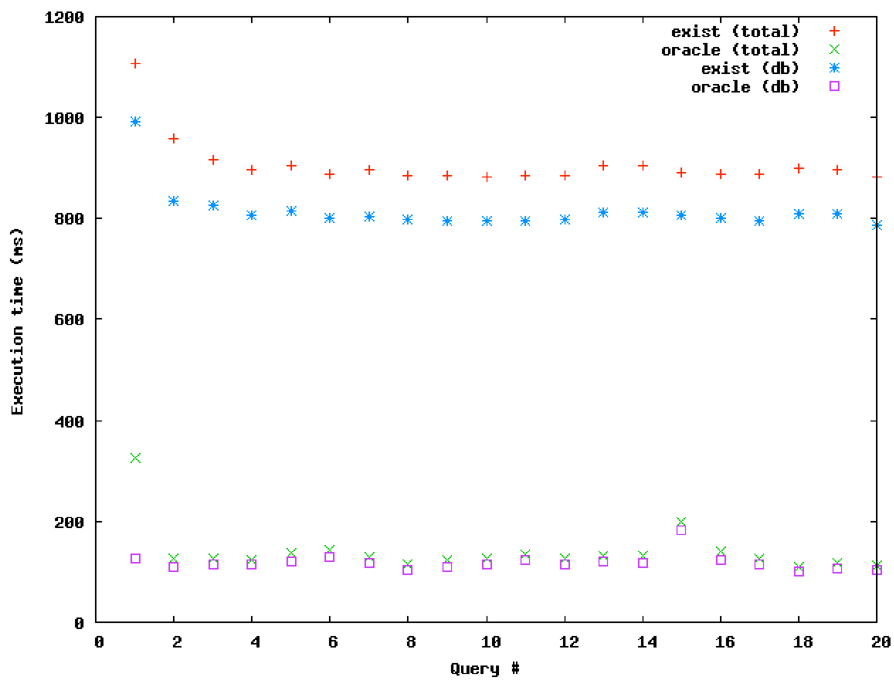
Exist Data

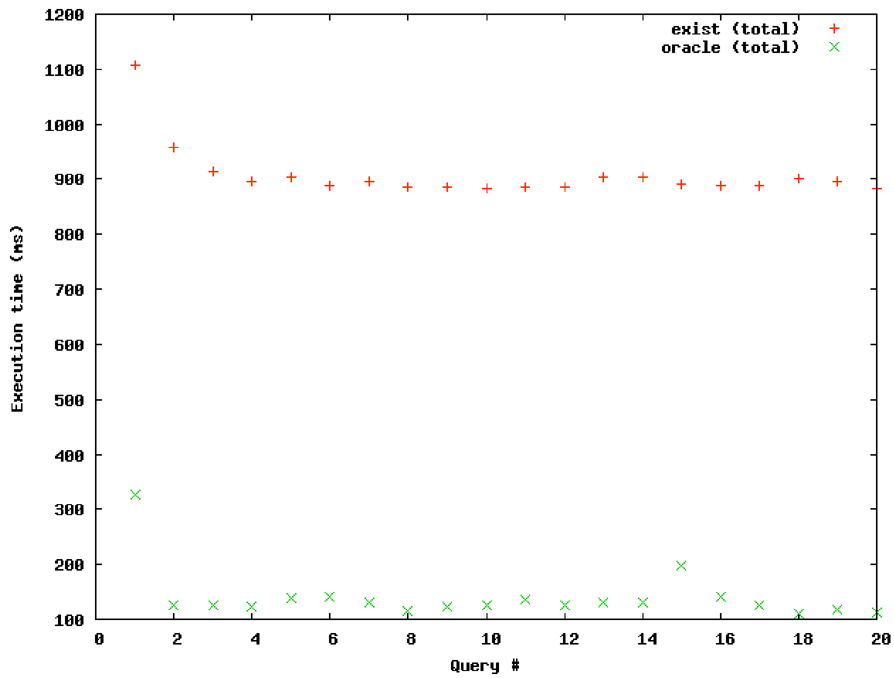
```
QUERY XML select * from First where tbl.CS239=B: (
    time: 686 db: 679)
QUERY XML select * from First where tbl.CS255=A: (
    time: 521 db: 514)
QUERY XML select * from First where tbl.CS211=A: (
    time: 527 db: 518)
QUERY XML select * from First where tbl.CS217=B: (
    time: 517 db: 510)
QUERY XML select * from First where tbl.CS219=A: (
    time: 527 db: 515)
QUERY XML select * from Second where tbl.CS292=B: (
    time: 516 db: 507)
QUERY XML select * from Second where tbl.CS107=A: (
    time: 516 db: 507)
QUERY XML select * from Second where tbl.CS244=B: (
    time: 526 db: 514)
QUERY XML select * from Second where tbl.CS279=B: (
    time: 523 db: 515)
QUERY XML select * from Second where tbl.CS183=B: (
    time: 545 db: 534)
QUERY XML select * from Third where tbl.CS277=A: (
    time: 516 db: 506)
QUERY XML select * from Third where tbl.CS228=B: (
    time: 510 db: 504)
QUERY XML select * from Third where tbl.CS298=A: (
    time: 525 db: 517)
QUERY XML select * from Third where tbl.CS281=B: (
    time: 527 db: 518)
QUERY XML select * from Third where tbl.CS183=B: (
    time: 513 db: 506)
QUERY XML select * from Fourth where tbl.CS136=A: (
    time: 518 db: 510)
QUERY XML select * from Fourth where tbl.CS251=B: (
    time: 509 db: 501)
QUERY XML select * from Fourth where tbl.CS277=B: (
    time: 515 db: 508)
```

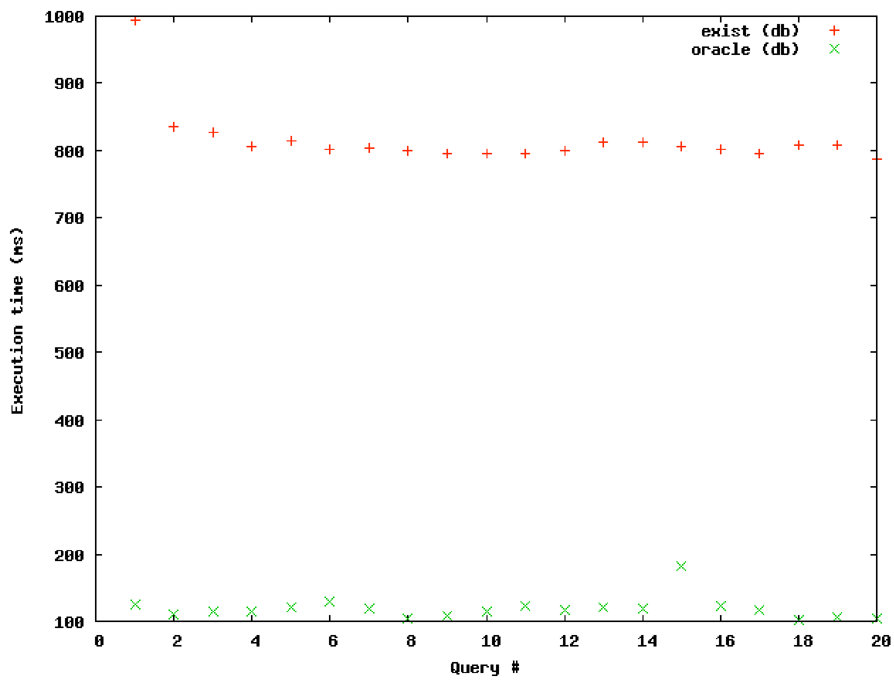
```
19 QUERY XML select * from Fourth where tbl.CS249=B: (
   time: 514 db: 505)
```

B.2.6 Project on Context

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
1000
900
800
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select cnt.year from First: (time: 327 db: 126)
QUERY XML select cnt.college from First: (time: 126 db: 110)
QUERY XML select cnt.comments from First: (time: 127 db: 114)
QUERY XML select cnt.major from First: (time: 124 db: 115)
QUERY XML select cnt.semester from First: (time: 138 db: 121)
QUERY XML select cnt.instructor from Second: (time: 142 db: 129)
QUERY XML select cnt.comments from Second: (time: 130 db: 118)
QUERY XML select cnt.semester from Second: (time: 115 db: 104)
QUERY XML select cnt.college from Second: (time: 123 db: 109)
QUERY XML select cnt.major from Second: (time: 126 db: 115)
QUERY XML select cnt.year from Third: (time: 135 db: 123)
QUERY XML select cnt.major from Third: (time: 127 db: 116)
QUERY XML select cnt.instructor from Third: (time: 131 db: 121)
QUERY XML select cnt.college from Third: (time: 132 db: 118)
QUERY XML select cnt.comments from Third: (time: 199 db: 182)
QUERY XML select cnt.instructor from Fourth: (time: 140 db: 124)
QUERY XML select cnt.year from Fourth: (time: 127 db: 116)
QUERY XML select cnt.comments from Fourth: (time: 111 db: 102)
QUERY XML select cnt.major from Fourth: (time: 117 db: 107)
```

```
QUERY XML select cnt.college from Fourth: (time: 113 db: 104)
```

Exist Data

```
QUERY XML select cnt.year from First: (time: 1107 db: 993)
QUERY XML select cnt.college from First: (time: 958 db: 836)
QUERY XML select cnt.comments from First: (time: 915 db: 827)
QUERY XML select cnt.major from First: (time: 896 db: 807)
QUERY XML select cnt.semester from First: (time: 904 db: 814)
QUERY XML select cnt.instructor from Second: (time: 889 db: 802)
QUERY XML select cnt.comments from Second: (time: 897 db: 804)
QUERY XML select cnt.semester from Second: (time: 886 db: 799)
QUERY XML select cnt.college from Second: (time: 885 db: 795)
QUERY XML select cnt.major from Second: (time: 882 db: 795)
QUERY XML select cnt.year from Third: (time: 885 db: 796)
QUERY XML select cnt.major from Third: (time: 886 db: 799)
QUERY XML select cnt.instructor from Third: (time: 905 db: 813)
QUERY XML select cnt.college from Third: (time: 905 db: 813)
QUERY XML select cnt.comments from Third: (time: 891 db: 806)
QUERY XML select cnt.instructor from Fourth: (time: 888 db: 802)
QUERY XML select cnt.year from Fourth: (time: 889 db: 796)
```

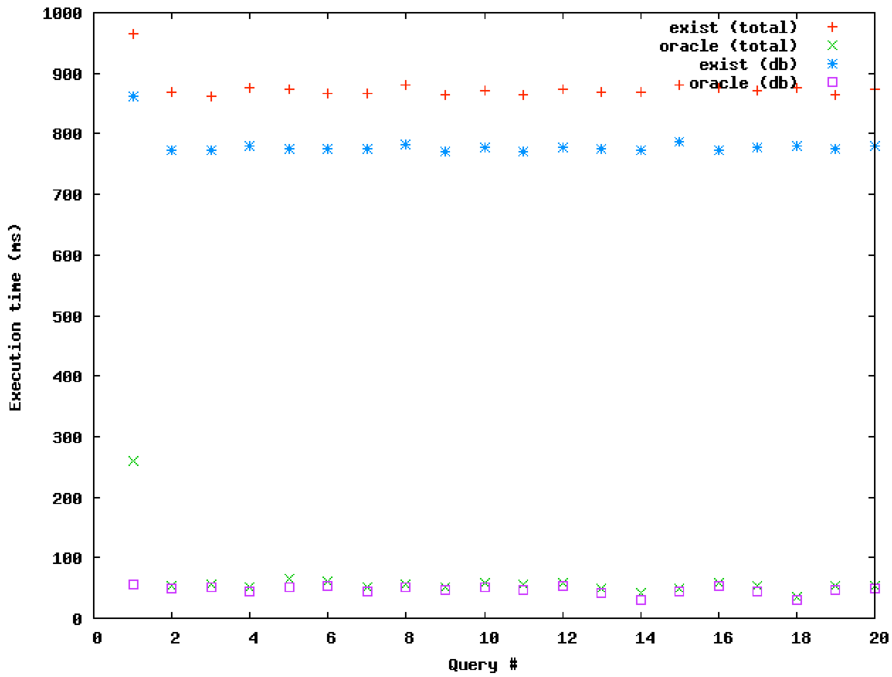
18 QUERY XML select cnt.comments from Fourth: (time: 900 db: 809)

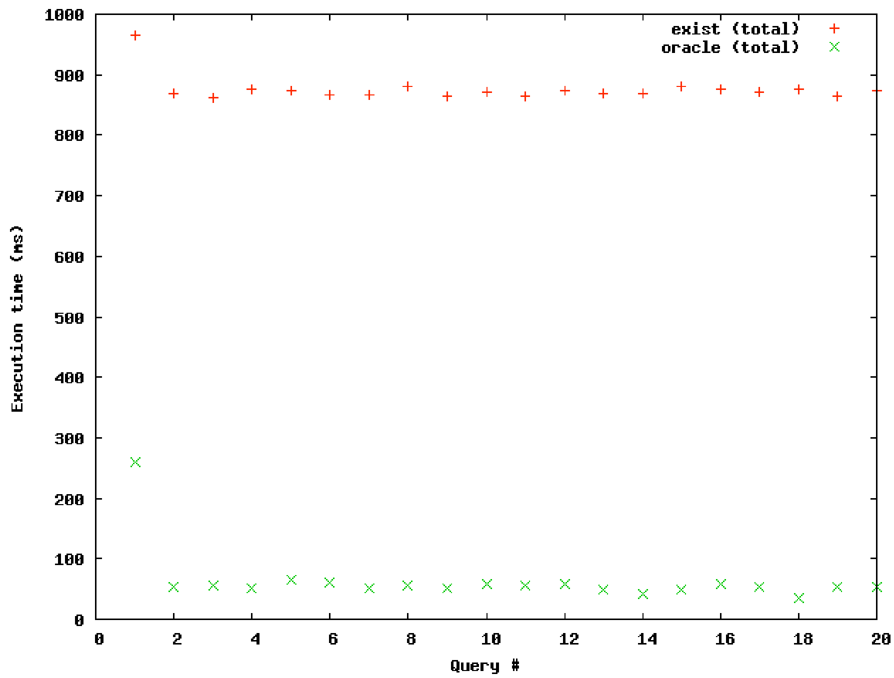
19 QUERY XML select cnt.major from Fourth: (time: 896 db: 808)

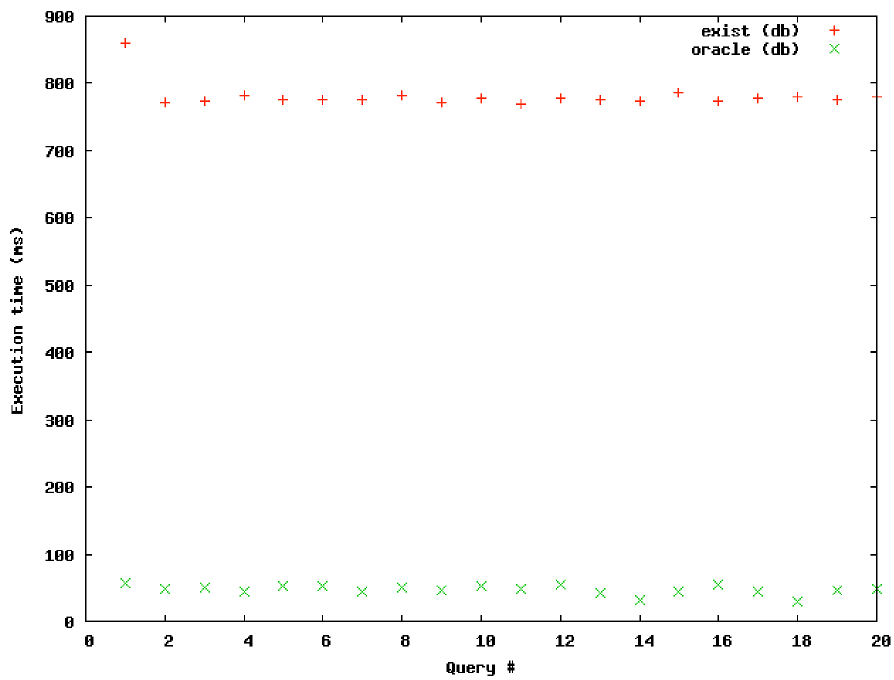
20 QUERY XML select cnt.college from Fourth: (time: 882 db: 787)

B.2.7 Project on Conditional

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
1000
900
800
700
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
900
800
700
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select cnd.CS115 from First: (time: 260 db: 57)
QUERY XML select cnd.CS146 from First: (time: 55 db: 49)
QUERY XML select cnd.CS105 from First: (time: 57 db: 51)
QUERY XML select cnd.CS134 from First: (time: 51 db: 44)
QUERY XML select cnd.CS130 from First: (time: 65 db: 52)
QUERY XML select cnd.CS132 from Second: (time: 60 db: 53)
QUERY XML select cnd.CS146 from Second: (time: 52 db: 45)
QUERY XML select cnd.CS109 from Second: (time: 56 db: 51)
QUERY XML select cnd.CS100 from Second: (time: 51 db: 46)
QUERY XML select cnd.CS122 from Second: (time: 59 db: 52)
QUERY XML select cnd.CS105 from Third: (time: 56 db: 48)
QUERY XML select cnd.CS117 from Third: (time: 58 db: 54)
QUERY XML select cnd.CS147 from Third: (time: 49 db: 42)
QUERY XML select cnd.CS122 from Third: (time: 41 db: 31)
QUERY XML select cnd.CS123 from Third: (time: 49 db: 44)
QUERY XML select cnd.CS111 from Fourth: (time: 59 db: 54)
QUERY XML select cnd.CS119 from Fourth: (time: 53 db: 45)
QUERY XML select cnd.CS128 from Fourth: (time: 36 db: 30)
QUERY XML select cnd.CS132 from Fourth: (time: 53 db: 47)
```

```
QUERY XML select cnd.CS116 from Fourth: (time: 54 db
    : 49)
```

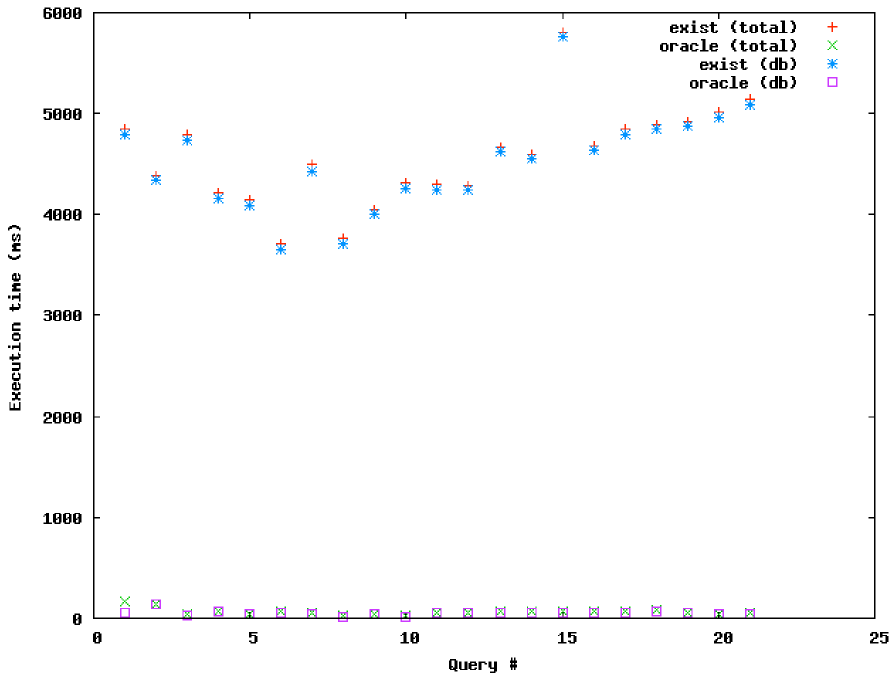
Exist Data

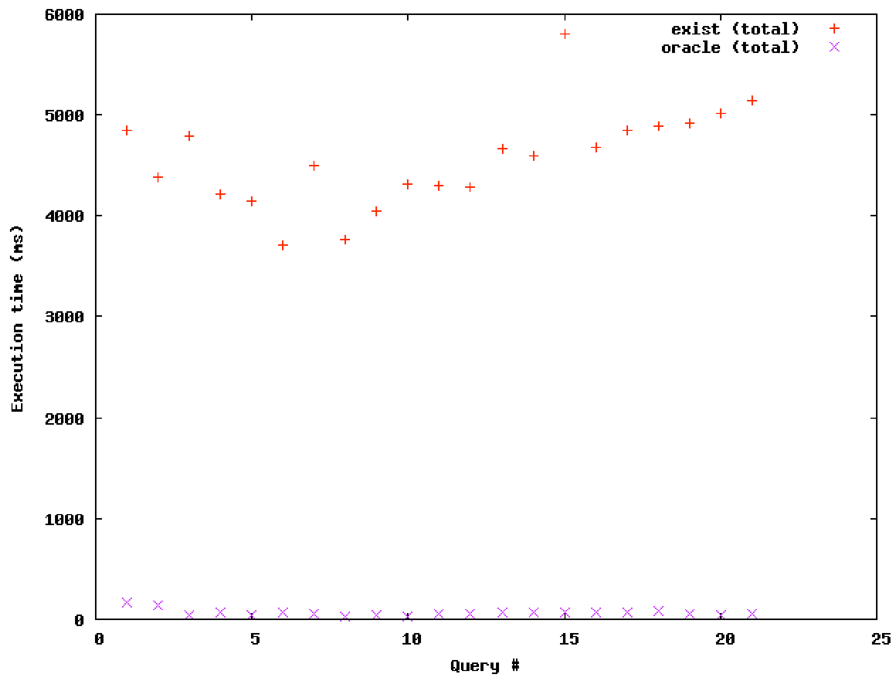
```
QUERY XML select cnd.CS115 from First: (time: 965 db
    : 861)
QUERY XML select cnd.CS146 from First: (time: 869 db
    : 772)
QUERY XML select cnd.CS105 from First: (time: 862 db
    : 774)
QUERY XML select cnd.CS134 from First: (time: 875 db
    : 781)
QUERY XML select cnd.CS130 from First: (time: 874 db
    : 776)
QUERY XML select cnd.CS132 from Second: (time: 866
    db: 776)
QUERY XML select cnd.CS146 from Second: (time: 867
    db: 776)
QUERY XML select cnd.CS109 from Second: (time: 881
    db: 783)
QUERY XML select cnd.CS100 from Second: (time: 864
    db: 771)
QUERY XML select cnd.CS122 from Second: (time: 871
    db: 777)
QUERY XML select cnd.CS105 from Third: (time: 864 db
    : 770)
QUERY XML select cnd.CS117 from Third: (time: 873 db
    : 777)
QUERY XML select cnd.CS147 from Third: (time: 869 db
    : 776)
QUERY XML select cnd.CS122 from Third: (time: 868 db
    : 774)
QUERY XML select cnd.CS123 from Third: (time: 880 db
    : 787)
QUERY XML select cnd.CS111 from Fourth: (time: 875
    db: 773)
QUERY XML select cnd.CS119 from Fourth: (time: 872
    db: 778)
```

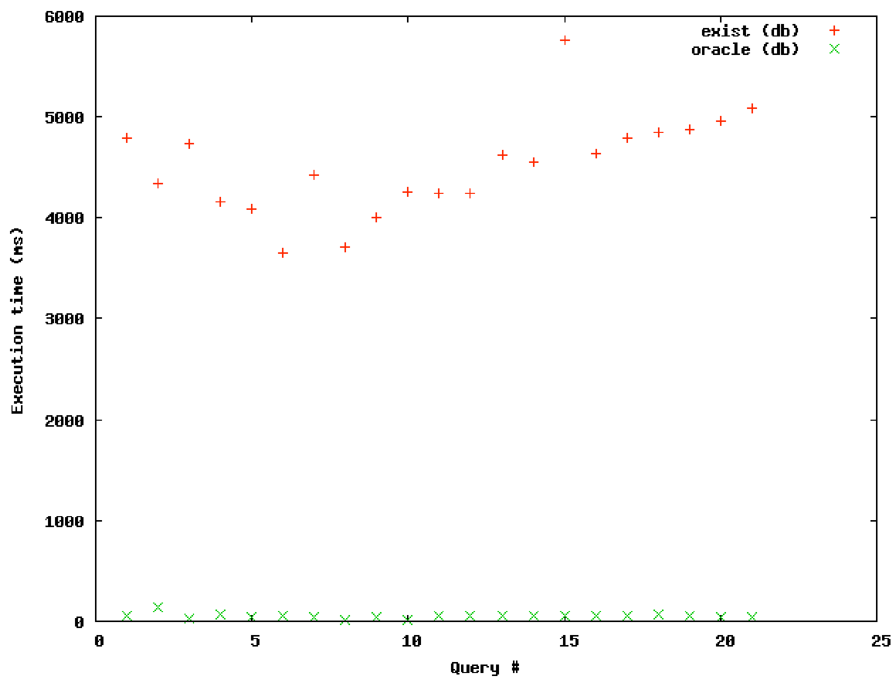
```
18 QUERY XML select cnd.CS128 from Fourth: (time: 875
   db: 779)
19 QUERY XML select cnd.CS132 from Fourth: (time: 865
   db: 776)
20 QUERY XML select cnd.CS116 from Fourth: (time: 873
   db: 779)
```

B.2.8 Project on Variables

Comparison




Execution time (ms)
Query #
exist (total)
oracle (total)
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25


Execution time (ms)
Query #
exist (db)
oracle (db)
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25

Oracle Data

```
QUERY XML select var.CS214 from First: (time: 171 db
    : 61)
QUERY XML select var.CS249 from First: (time: 145 db
    : 139)
QUERY XML select var.CS239 from First: (time: 38 db:
     31)
QUERY XML select var.CS282 from First: (time: 76 db:
     69)
QUERY XML select var.CS299 from First: (time: 41 db:
     36)
QUERY XML select var.CS297 from Second: (time: 67 db
    : 60)
QUERY XML select var.CS265 from Second: (time: 56 db
    : 49)
QUERY XML select var.CS265 from Second: (time: 25 db
    : 18)
QUERY XML select var.CS244 from Second: (time: 45 db
    : 39)
QUERY XML select var.CS244 from Second: (time: 24 db
    : 19)
QUERY XML select var.CS128 from Second: (time: 58 db
    : 53)
QUERY XML select var.CS244 from Third: (time: 61 db:
     56)
QUERY XML select var.CS299 from Third: (time: 64 db:
     57)
QUERY XML select var.CS113 from Third: (time: 65 db:
     58)
QUERY XML select var.CS192 from Third: (time: 66 db:
     60)
QUERY XML select var.CS169 from Third: (time: 68 db:
     60)
QUERY XML select var.CS126 from Fourth: (time: 64 db
    : 59)
QUERY XML select var.CS210 from Fourth: (time: 83 db
    : 76)
QUERY XML select var.CS164 from Fourth: (time: 56 db
    : 50)
```

```
QUERY XML select var.CS270 from Fourth: (time: 46 db
    : 39)
QUERY XML select var.CS213 from Fourth: (time: 56 db
    : 49)
```

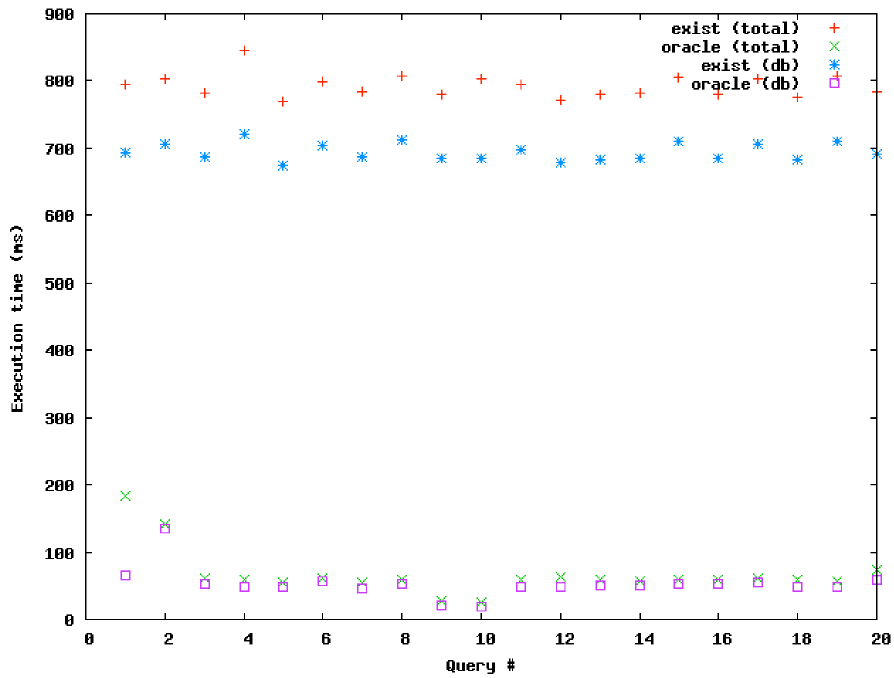
Exist Data

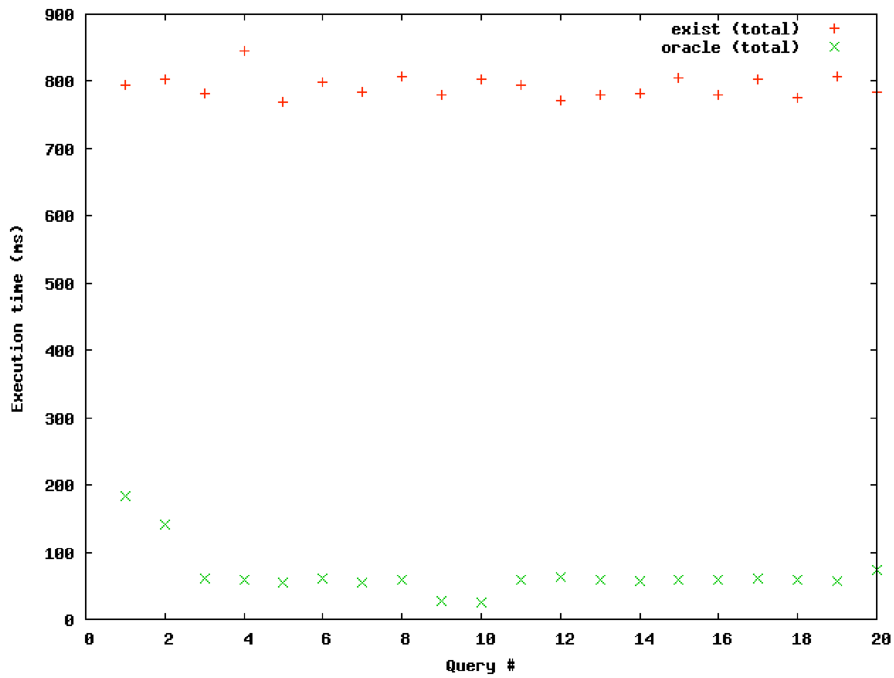
```
QUERY XML select var.CS214 from First: (time: 4844
    db: 4795)
QUERY XML select var.CS249 from First: (time: 4384
    db: 4338)
QUERY XML select var.CS239 from First: (time: 4798
    db: 4741)
QUERY XML select var.CS282 from First: (time: 4214
    db: 4153)
QUERY XML select var.CS299 from First: (time: 4149
    db: 4092)
QUERY XML select var.CS297 from Second: (time: 3714
    db: 3654)
QUERY XML select var.CS265 from Second: (time: 4491
    db: 4431)
QUERY XML select var.CS265 from Second: (time: 3764
    db: 3716)
QUERY XML select var.CS244 from Second: (time: 4044
    db: 3999)
QUERY XML select var.CS244 from Second: (time: 4309
    db: 4254)
QUERY XML select var.CS128 from Second: (time: 4305
    db: 4240)
QUERY XML select var.CS244 from Third: (time: 4284
    db: 4240)
QUERY XML select var.CS299 from Third: (time: 4666
    db: 4620)
QUERY XML select var.CS113 from Third: (time: 4595
    db: 4546)
QUERY XML select var.CS192 from Third: (time: 5810
    db: 5767)
QUERY XML select var.CS169 from Third: (time: 4679
    db: 4635)
```

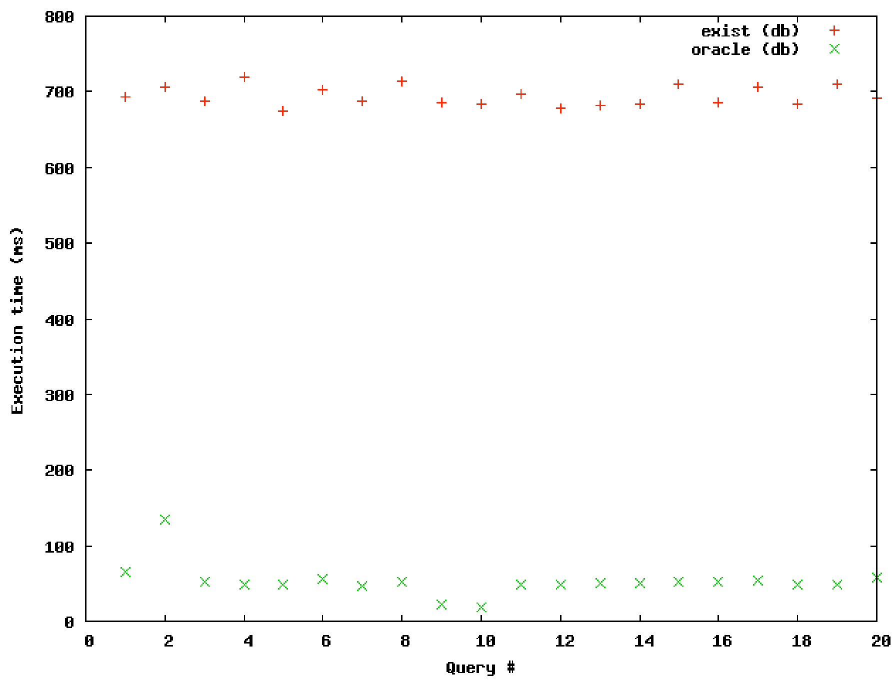
```
QUERY XML select var.CS126 from Fourth: (time: 4843
   db: 4798)
QUERY XML select var.CS210 from Fourth: (time: 4894
   db: 4851)
QUERY XML select var.CS164 from Fourth: (time: 4919
   db: 4870)
QUERY XML select var.CS270 from Fourth: (time: 5014
   db: 4965)
QUERY XML select var.CS213 from Fourth: (time: 5137
   db: 5091)
```

B.2.9 Conditional

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First conditional var.CS172=A: (time: 184 db: 65)
QUERY XML select * from First conditional var.CS113=A: (time: 142 db: 135)
QUERY XML select * from First conditional var.CS229=B: (time: 61 db: 53)
QUERY XML select * from First conditional var.CS251=B: (time: 58 db: 48)
QUERY XML select * from First conditional var.CS256=B: (time: 55 db: 49)
QUERY XML select * from Second conditional var.CS233=A: (time: 62 db: 56)
QUERY XML select * from Second conditional var.CS219=B: (time: 55 db: 47)
QUERY XML select * from Second conditional var.CS190=B: (time: 58 db: 52)
QUERY XML select * from Second conditional var.CS297=B: (time: 28 db: 22)
QUERY XML select * from Second conditional var.CS297=B: (time: 26 db: 18)
QUERY XML select * from Third conditional var.CS138=A: (time: 58 db: 49)
QUERY XML select * from Third conditional var.CS251=A: (time: 64 db: 48)
QUERY XML select * from Third conditional var.CS202=B: (time: 60 db: 51)
QUERY XML select * from Third conditional var.CS144=A: (time: 57 db: 51)
QUERY XML select * from Third conditional var.CS240=B: (time: 59 db: 53)
QUERY XML select * from Fourth conditional var.CS112=B: (time: 60 db: 53)
QUERY XML select * from Fourth conditional var.CS206=A: (time: 62 db: 55)
QUERY XML select * from Fourth conditional var.CS217=A: (time: 59 db: 49)
QUERY XML select * from Fourth conditional var.CS275=A: (time: 56 db: 49)
```

```
QUERY XML select * from Fourth conditional var.CS111=A: (time: 73 db: 59)
```

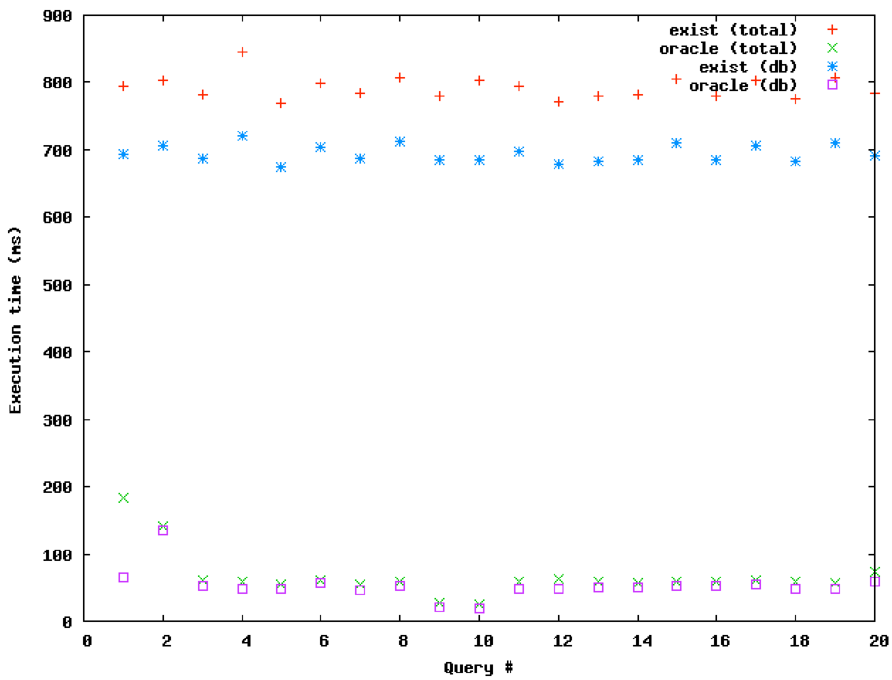
Exist Data

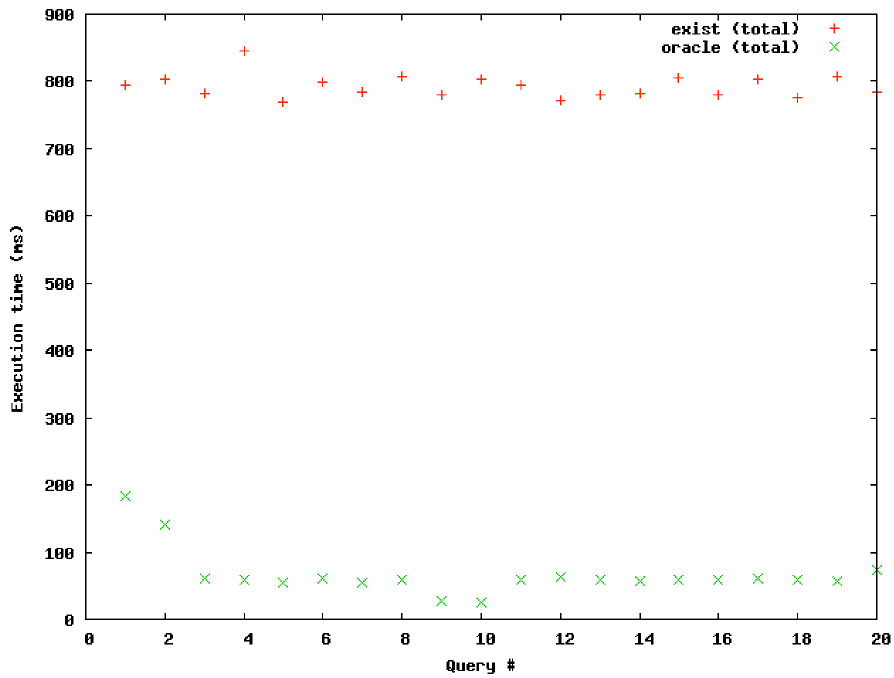
```
QUERY XML select * from First conditional var.CS172=A: (time: 795 db: 694)
QUERY XML select * from First conditional var.CS113=A: (time: 802 db: 707)
QUERY XML select * from First conditional var.CS229=B: (time: 781 db: 688)
QUERY XML select * from First conditional var.CS251=B: (time: 846 db: 720)
QUERY XML select * from First conditional var.CS256=B: (time: 770 db: 675)
QUERY XML select * from Second conditional var.CS233=A: (time: 799 db: 703)
QUERY XML select * from Second conditional var.CS219=B: (time: 785 db: 688)
QUERY XML select * from Second conditional var.CS190=B: (time: 807 db: 713)
QUERY XML select * from Second conditional var.CS297=B: (time: 780 db: 686)
QUERY XML select * from Second conditional var.CS297=B: (time: 803 db: 684)
QUERY XML select * from Third conditional var.CS138=A: (time: 794 db: 697)
QUERY XML select * from Third conditional var.CS251=A: (time: 772 db: 679)
QUERY XML select * from Third conditional var.CS202=B: (time: 779 db: 682)
QUERY XML select * from Third conditional var.CS144=A: (time: 782 db: 684)
QUERY XML select * from Third conditional var.CS240=B: (time: 805 db: 710)
QUERY XML select * from Fourth conditional var.CS112=B: (time: 780 db: 686)
QUERY XML select * from Fourth conditional var.CS206=A: (time: 803 db: 707)
```

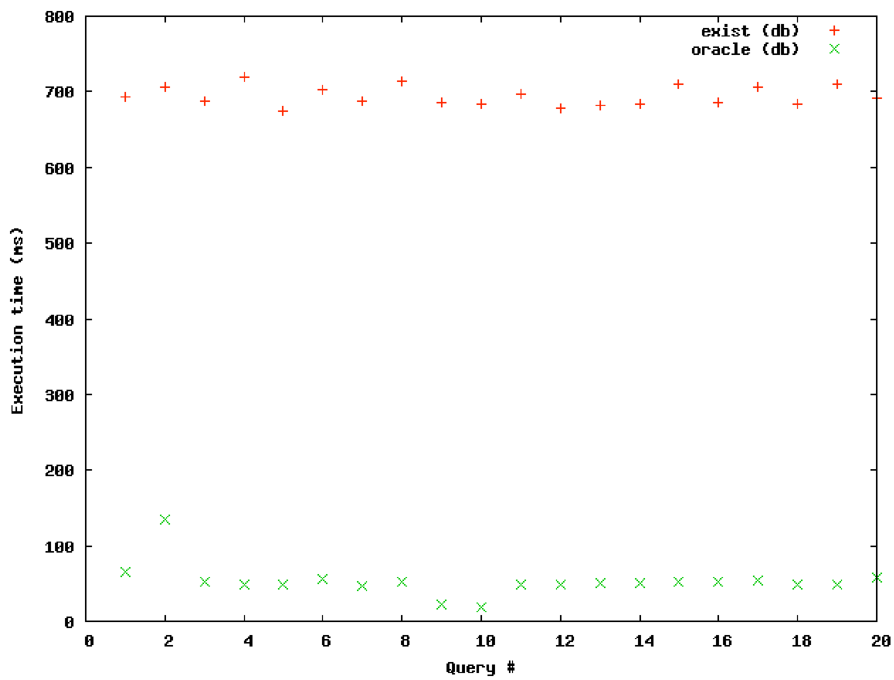
```
18 QUERY XML select * from Fourth conditional var.CS217
     =A: (time: 776 db: 683)
19 QUERY XML select * from Fourth conditional var.CS275
     =A: (time: 807 db: 710)
20 QUERY XML select * from Fourth conditional var.CS111
     =A: (time: 785 db: 691)
```

B.2.10 Conditional

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML select * from First conditional var.CS172=A: (time: 184 db: 65)
QUERY XML select * from First conditional var.CS113=A: (time: 142 db: 135)
QUERY XML select * from First conditional var.CS229=B: (time: 61 db: 53)
QUERY XML select * from First conditional var.CS251=B: (time: 58 db: 48)
QUERY XML select * from First conditional var.CS256=B: (time: 55 db: 49)
QUERY XML select * from Second conditional var.CS233=A: (time: 62 db: 56)
QUERY XML select * from Second conditional var.CS219=B: (time: 55 db: 47)
QUERY XML select * from Second conditional var.CS190=B: (time: 58 db: 52)
QUERY XML select * from Second conditional var.CS297=B: (time: 28 db: 22)
QUERY XML select * from Second conditional var.CS297=B: (time: 26 db: 18)
QUERY XML select * from Third conditional var.CS138=A: (time: 58 db: 49)
QUERY XML select * from Third conditional var.CS251=A: (time: 64 db: 48)
QUERY XML select * from Third conditional var.CS202=B: (time: 60 db: 51)
QUERY XML select * from Third conditional var.CS144=A: (time: 57 db: 51)
QUERY XML select * from Third conditional var.CS240=B: (time: 59 db: 53)
QUERY XML select * from Fourth conditional var.CS112=B: (time: 60 db: 53)
QUERY XML select * from Fourth conditional var.CS206=A: (time: 62 db: 55)
QUERY XML select * from Fourth conditional var.CS217=A: (time: 59 db: 49)
QUERY XML select * from Fourth conditional var.CS275=A: (time: 56 db: 49)
```

```
20 QUERY XML select * from Fourth conditional var.CS111=A: (time: 73 db: 59)
```

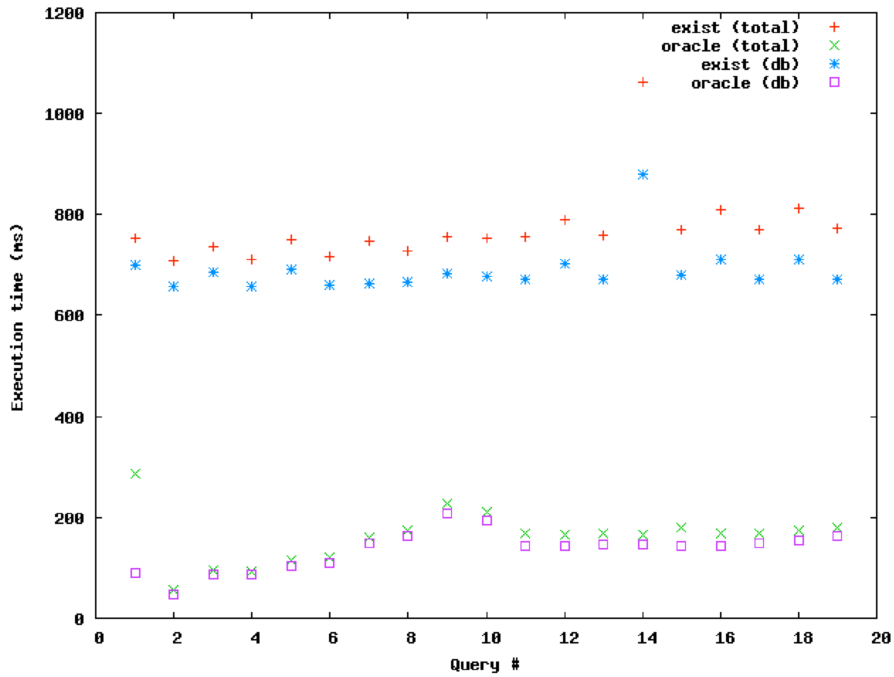
Exist Data

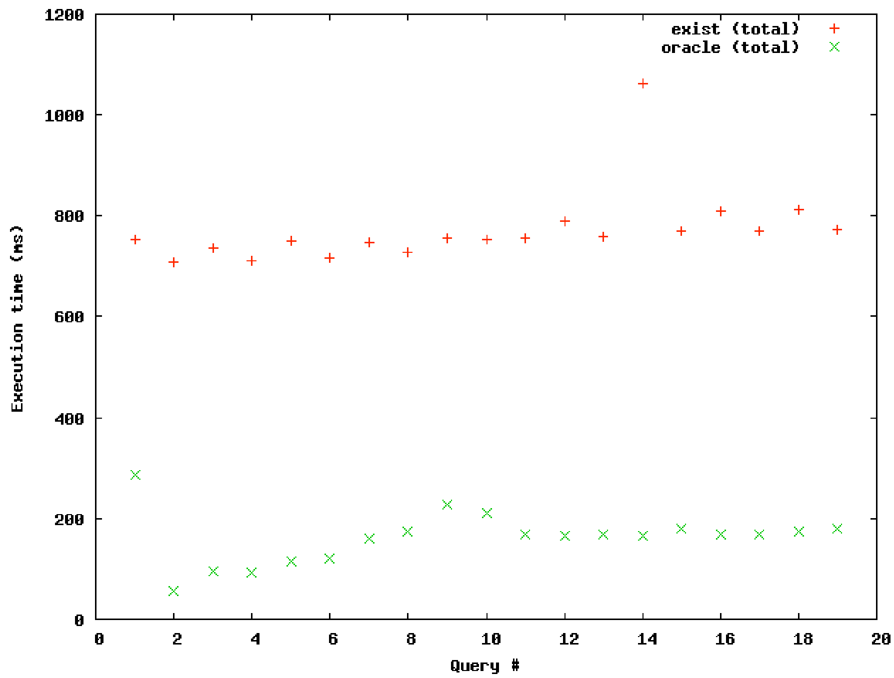
```
1 QUERY XML select * from First conditional var.CS172=A: (time: 795 db: 694)
2 QUERY XML select * from First conditional var.CS113=A: (time: 802 db: 707)
3 QUERY XML select * from First conditional var.CS229=B: (time: 781 db: 688)
4 QUERY XML select * from First conditional var.CS251=B: (time: 846 db: 720)
5 QUERY XML select * from First conditional var.CS256=B: (time: 770 db: 675)
6 QUERY XML select * from Second conditional var.CS233=A: (time: 799 db: 703)
7 QUERY XML select * from Second conditional var.CS219=B: (time: 785 db: 688)
8 QUERY XML select * from Second conditional var.CS190=B: (time: 807 db: 713)
9 QUERY XML select * from Second conditional var.CS297=B: (time: 780 db: 686)
10 QUERY XML select * from Second conditional var.CS297=B: (time: 803 db: 684)
11 QUERY XML select * from Third conditional var.CS138=A: (time: 794 db: 697)
12 QUERY XML select * from Third conditional var.CS251=A: (time: 772 db: 679)
13 QUERY XML select * from Third conditional var.CS202=B: (time: 779 db: 682)
14 QUERY XML select * from Third conditional var.CS144=A: (time: 782 db: 684)
15 QUERY XML select * from Third conditional var.CS240=B: (time: 805 db: 710)
16 QUERY XML select * from Fourth conditional var.CS112=B: (time: 780 db: 686)
17 QUERY XML select * from Fourth conditional var.CS206=A: (time: 803 db: 707)
```

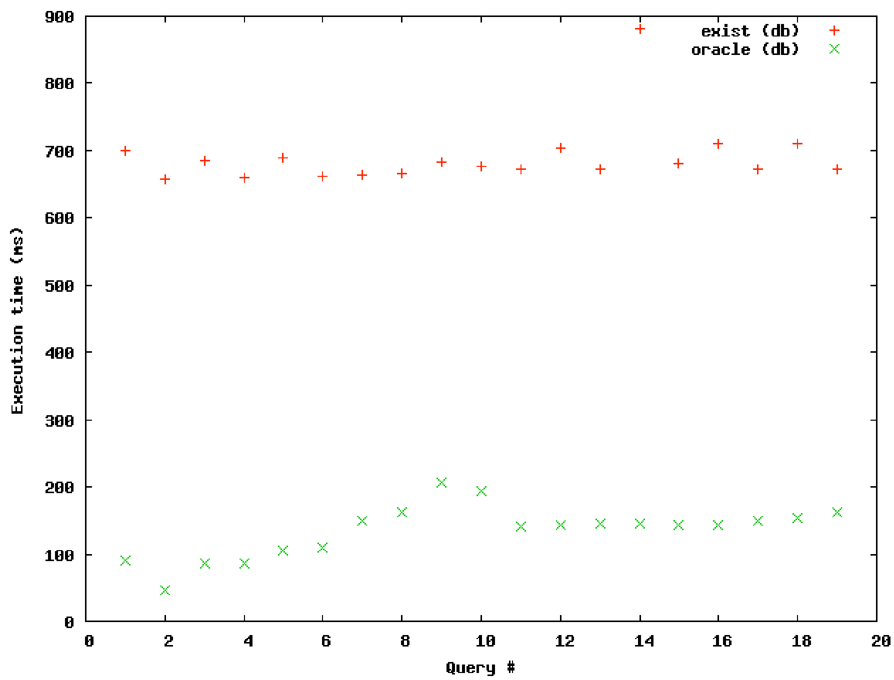
```
18 QUERY XML select * from Fourth conditional var.CS217
   =A: (time: 776 db: 683)
19 QUERY XML select * from Fourth conditional var.CS275
   =A: (time: 807 db: 710)
20 QUERY XML select * from Fourth conditional var.CS111
   =A: (time: 785 db: 691)
```

B.2.11 Select on Probability

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #


exist (db)
oracle (db)
Execution time (ms)
Query #

Oracle Data

```
QUERY XML select * from First where tbl.prob>0.64: (
   time: 286 db: 91)
QUERY XML select * from First where tbl.prob>0.60: (
   time: 56 db: 47)
QUERY XML select * from First where tbl.prob>0.56: (
   time: 95 db: 87)
QUERY XML select * from First where tbl.prob>0.52: (
   time: 94 db: 86)
QUERY XML select * from First where tbl.prob>0.48: (
   time: 115 db: 105)
QUERY XML select * from First where tbl.prob>0.44: (
   time: 120 db: 110)
QUERY XML select * from First where tbl.prob>0.40: (
   time: 159 db: 149)
QUERY XML select * from First where tbl.prob>0.36: (
   time: 175 db: 162)
QUERY XML select * from First where tbl.prob>0.28: (
   time: 228 db: 207)
QUERY XML select * from First where tbl.prob>0.24: (
   time: 212 db: 194)
QUERY XML select * from First where tbl.prob>0.20: (
   time: 169 db: 142)
QUERY XML select * from First where tbl.prob>0.18: (
   time: 165 db: 144)
QUERY XML select * from First where tbl.prob>0.16: (
   time: 168 db: 145)
QUERY XML select * from First where tbl.prob>0.14: (
   time: 166 db: 146)
QUERY XML select * from First where tbl.prob>0.12: (
   time: 180 db: 144)
QUERY XML select * from First where tbl.prob>0.10: (
   time: 169 db: 144)
QUERY XML select * from First where tbl.prob>0.08: (
   time: 170 db: 149)
QUERY XML select * from First where tbl.prob>0.04: (
   time: 174 db: 154)
QUERY XML select * from First where tbl.prob>0.02: (
   time: 180 db: 163)
```

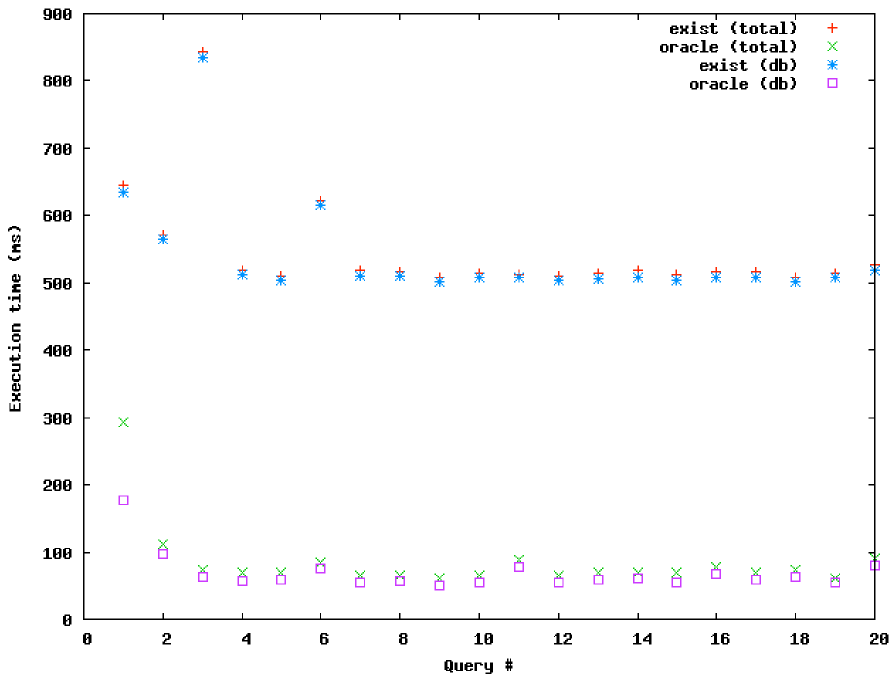
Exist Data

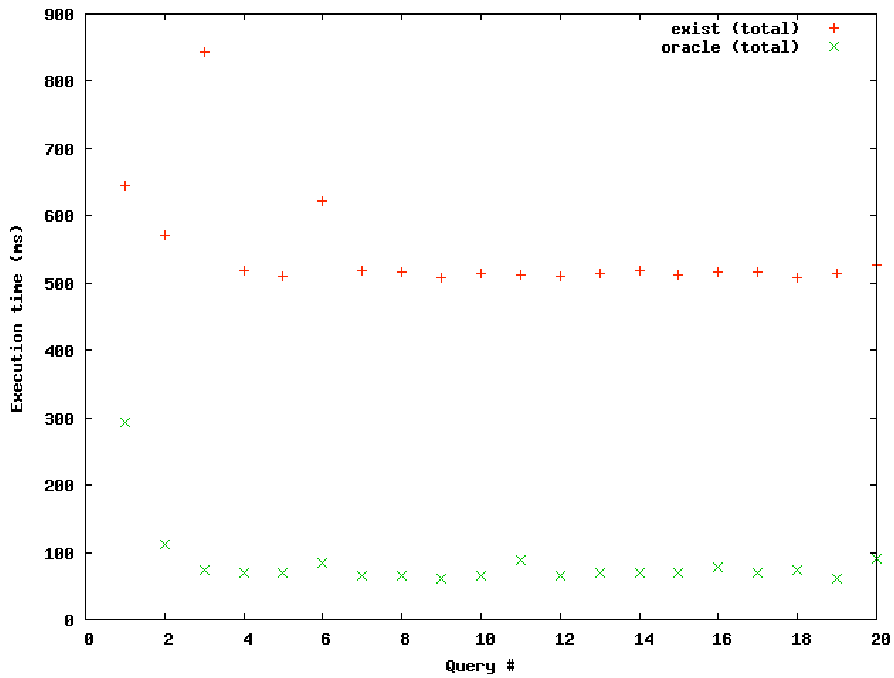
```
QUERY XML select * from First where tbl.prob>0.64: (
    time: 754 db: 700)
QUERY XML select * from First where tbl.prob>0.60: (
    time: 708 db: 658)
QUERY XML select * from First where tbl.prob>0.56: (
    time: 737 db: 685)
QUERY XML select * from First where tbl.prob>0.52: (
    time: 711 db: 659)
QUERY XML select * from First where tbl.prob>0.48: (
    time: 750 db: 690)
QUERY XML select * from First where tbl.prob>0.44: (
    time: 716 db: 661)
QUERY XML select * from First where tbl.prob>0.40: (
    time: 748 db: 664)
QUERY XML select * from First where tbl.prob>0.36: (
    time: 728 db: 666)
QUERY XML select * from First where tbl.prob>0.28: (
    time: 757 db: 682)
QUERY XML select * from First where tbl.prob>0.24: (
    time: 753 db: 676)
QUERY XML select * from First where tbl.prob>0.20: (
    time: 757 db: 672)
QUERY XML select * from First where tbl.prob>0.18: (
    time: 790 db: 703)
QUERY XML select * from First where tbl.prob>0.16: (
    time: 758 db: 672)
QUERY XML select * from First where tbl.prob>0.14: (
    time: 1062 db: 881)
QUERY XML select * from First where tbl.prob>0.12: (
    time: 771 db: 681)
QUERY XML select * from First where tbl.prob>0.10: (
    time: 808 db: 711)
QUERY XML select * from First where tbl.prob>0.08: (
    time: 769 db: 673)
QUERY XML select * from First where tbl.prob>0.04: (
    time: 813 db: 710)
```

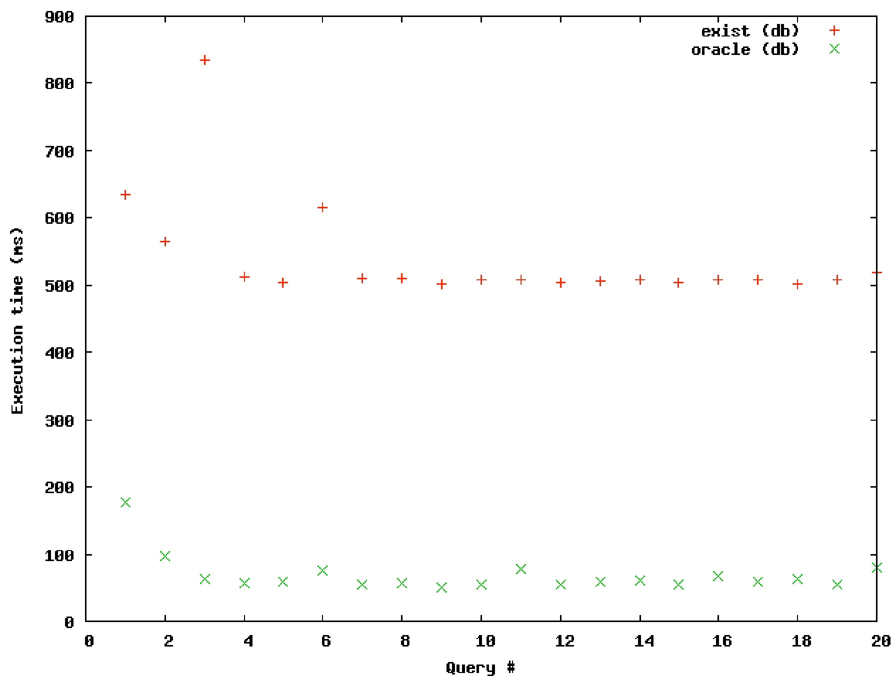
```
QUERY XML select * from First where tbl.prob>0.02: (
  time: 773 db: 672)
```

B.2.12 Complex Select Conditions

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
100
200
300
400
500
600
700
800
900
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML Select * from First where First.tbl.CS115="A" and First.tbl.prob>0.01: (time: 292 db: 177)
QUERY XML Select * from First where First.tbl.CS242="B" and First.tbl.prob>0.03: (time: 111 db: 98)
QUERY XML Select * from First where First.tbl.CS219="A" and First.tbl.prob>0.06: (time: 74 db: 63)
QUERY XML Select * from First where First.tbl.CS211="B" and First.tbl.prob>0.08: (time: 70 db: 56)
QUERY XML Select * from First where First.tbl.CS172="B" and First.tbl.prob>0.09: (time: 70 db: 60)
QUERY XML Select * from Second where Second.tbl.CS292="B" and Second.tbl.prob>0.01: (time: 84 db: 76)
QUERY XML Select * from Second where Second.tbl.CS107="A" and Second.tbl.prob>0.03: (time: 65 db: 54)
QUERY XML Select * from Second where Second.tbl.CS244="B" and Second.tbl.prob>0.06: (time: 66 db: 56)
QUERY XML Select * from Second where Second.tbl.CS279="B" and Second.tbl.prob>0.08: (time: 61 db: 50)
QUERY XML Select * from Second where Second.tbl.CS183="B" and Second.tbl.prob>0.09: (time: 65 db: 55)
QUERY XML Select * from Third where Third.tbl.CS277="A" and Third.tbl.prob>0.01: (time: 89 db: 79)
QUERY XML Select * from Third where Third.tbl.CS228="B" and Third.tbl.prob>0.03: (time: 65 db: 54)
QUERY XML Select * from Third where Third.tbl.CS298="A" and Third.tbl.prob>0.06: (time: 70 db: 60)
QUERY XML Select * from Third where Third.tbl.CS281="B" and Third.tbl.prob>0.08: (time: 70 db: 62)
QUERY XML Select * from Third where Third.tbl.CS183="B" and Third.tbl.prob>0.09: (time: 69 db: 54)
QUERY XML Select * from Fourth where Fourth.tbl.CS136="A" and Fourth.tbl.prob>0.09: (time: 77 db: 68)
```

```
QUERY XML Select * from Fourth where Fourth.tbl.CS251="B" and Fourth.tbl.prob>0.09: (time: 70 db: 59)
QUERY XML Select * from Fourth where Fourth.tbl.CS277="B" and Fourth.tbl.prob>0.09: (time: 73 db: 63)
QUERY XML Select * from Fourth where Fourth.tbl.CS249="B" and Fourth.tbl.prob>0.09: (time: 62 db: 54)
QUERY XML Select * from Fourth where Fourth.tbl.CS271="A" and Fourth.tbl.prob>0.09: (time: 90 db: 81)
```

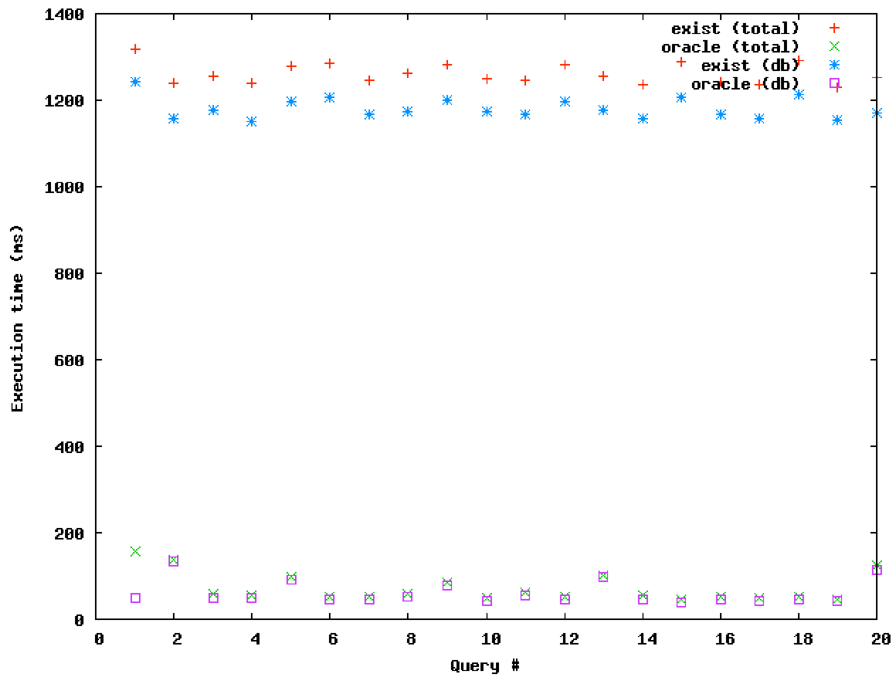
Exist Data

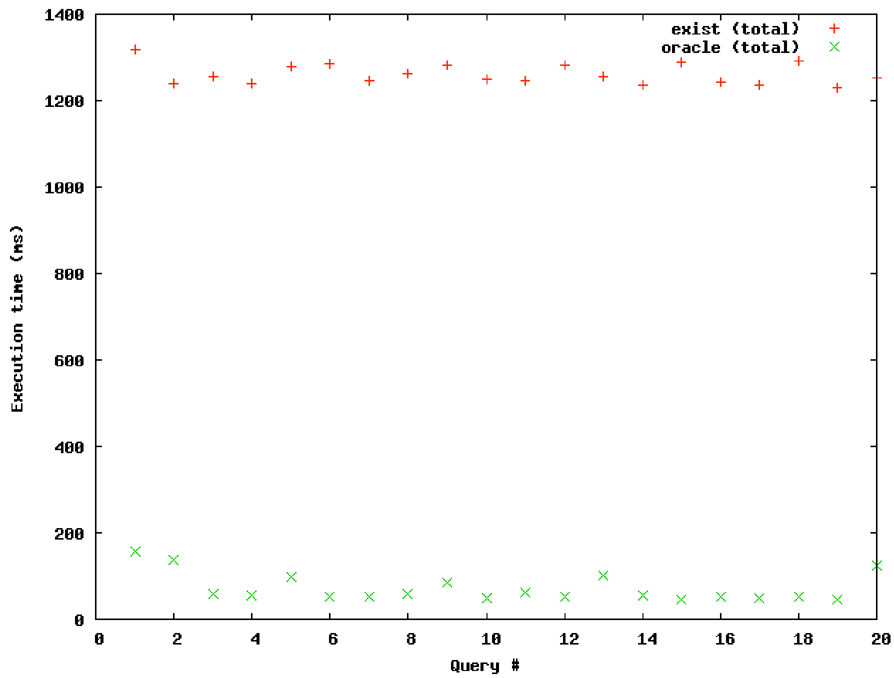
```
QUERY XML Select * from First where First.tbl.CS115="A" and First.tbl.prob>0.01: (time: 645 db: 635)
QUERY XML Select * from First where First.tbl.CS242="B" and First.tbl.prob>0.03: (time: 571 db: 564)
QUERY XML Select * from First where First.tbl.CS219="A" and First.tbl.prob>0.06: (time: 844 db: 834)
QUERY XML Select * from First where First.tbl.CS211="B" and First.tbl.prob>0.08: (time: 518 db: 512)
QUERY XML Select * from First where First.tbl.CS172="B" and First.tbl.prob>0.09: (time: 511 db: 504)
QUERY XML Select * from Second where Second.tbl.CS292="B" and Second.tbl.prob>0.01: (time: 622 db: 615)
QUERY XML Select * from Second where Second.tbl.CS107="A" and Second.tbl.prob>0.03: (time: 518 db: 511)
QUERY XML Select * from Second where Second.tbl.CS244="B" and Second.tbl.prob>0.06: (time: 517 db: 510)
QUERY XML Select * from Second where Second.tbl.CS279="B" and Second.tbl.prob>0.08: (time: 509 db: 502)
QUERY XML Select * from Second where Second.tbl.CS183="B" and Second.tbl.prob>0.09: (time: 514 db
```

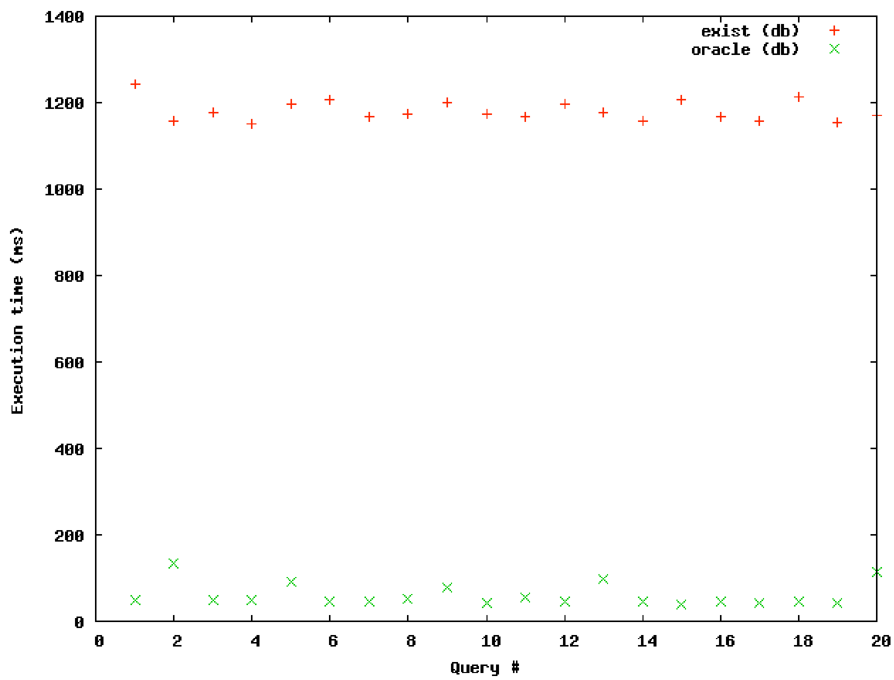
```
   : 507)
QUERY XML Select * from Third where Third.tbl.CS277
   ="A" and Third.tbl.prob>0.01: (time: 513 db: 507)
QUERY XML Select * from Third where Third.tbl.CS228
   ="B" and Third.tbl.prob>0.03: (time: 510 db: 503)
QUERY XML Select * from Third where Third.tbl.CS298
   ="A" and Third.tbl.prob>0.06: (time: 514 db: 506)
QUERY XML Select * from Third where Third.tbl.CS281
   ="B" and Third.tbl.prob>0.08: (time: 518 db: 509)
QUERY XML Select * from Third where Third.tbl.CS183
   ="B" and Third.tbl.prob>0.09: (time: 512 db: 504)
QUERY XML Select * from Fourth where Fourth.tbl.
   CS136="A" and Fourth.tbl.prob>0.09: (time: 516 db
   : 509)
QUERY XML Select * from Fourth where Fourth.tbl.
   CS251="B" and Fourth.tbl.prob>0.09: (time: 516 db
   : 507)
QUERY XML Select * from Fourth where Fourth.tbl.
   CS277="B" and Fourth.tbl.prob>0.09: (time: 509 db
   : 502)
QUERY XML Select * from Fourth where Fourth.tbl.
   CS249="B" and Fourth.tbl.prob>0.09: (time: 514 db
   : 508)
QUERY XML Select * from Fourth where Fourth.tbl.
   CS271="A" and Fourth.tbl.prob>0.09: (time: 526 db
   : 519)
```

B.2.13 Complex Project Conditions

Comparison




exist (total)
oracle (total)
Execution time (ms)
Query #
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12
14
16
18
20


exist (db)
oracle (db)
Execution time (ms)
Query #
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12
14
16
18
20

Oracle Data

```
QUERY XML Select First.cnt.college,First.cnd.CS101
    from First: (time: 159 db: 50)
QUERY XML Select First.cnt.comments,First.cnd.CS106
    from First: (time: 139 db: 133)
QUERY XML Select First.cnt.year,First.cnd.CS143 from
     First: (time: 58 db: 50)
QUERY XML Select First.cnt.semester,First.cnd.CS103
    from First: (time: 55 db: 49)
QUERY XML Select First.cnt.instructor,First.cnd.
    CS121 from First: (time: 100 db: 93)
QUERY XML Select Second.cnt.semester,Second.cnd.
    CS143 from Second: (time: 53 db: 46)
QUERY XML Select Second.cnt.major,Second.cnd.CS146
    from Second: (time: 53 db: 46)
QUERY XML Select Second.cnt.major,Second.cnd.CS135
    from Second: (time: 58 db: 51)
QUERY XML Select Second.cnt.year,Second.cnd.CS102
    from Second: (time: 85 db: 78)
QUERY XML Select Second.cnt.semester,Second.cnd.
    CS127 from Second: (time: 50 db: 44)
QUERY XML Select Third.cnt.major,Third.cnd.CS144
    from Third: (time: 63 db: 55)
QUERY XML Select Third.cnt.instructor,Third.cnd.
    CS138 from Third: (time: 54 db: 47)
QUERY XML Select Third.cnt.semester,Third.cnd.CS112
    from Third: (time: 102 db: 97)
QUERY XML Select Third.cnt.comments,Third.cnd.CS140
    from Third: (time: 55 db: 46)
QUERY XML Select Third.cnt.comments,Third.cnd.CS122
    from Third: (time: 47 db: 40)
QUERY XML Select Fourth.cnt.comments,Fourth.cnd.
    CS117 from Fourth: (time: 54 db: 47)
QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS124
    from Fourth: (time: 48 db: 42)
QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS107
    from Fourth: (time: 51 db: 46)
QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS128
    from Fourth: (time: 46 db: 41)
```

```
20 QUERY XML Select Fourth.cnt.semester,Fourth.cnd.
      CS130 from Fourth: (time: 124 db: 116)
```

Exist Data

```
1 QUERY XML Select First.cnt.college,First.cnd.CS101
      from First: (time: 1319 db: 1241)
2 QUERY XML Select First.cnt.comments,First.cnd.CS106
      from First: (time: 1239 db: 1158)
3 QUERY XML Select First.cnt.year,First.cnd.CS143 from
       First: (time: 1255 db: 1177)
4 QUERY XML Select First.cnt.semester,First.cnd.CS103
      from First: (time: 1238 db: 1151)
5 QUERY XML Select First.cnt.instructor,First.cnd.
      CS121 from First: (time: 1280 db: 1198)
6 QUERY XML Select Second.cnt.semester,Second.cnd.
      CS143 from Second: (time: 1284 db: 1205)
7 QUERY XML Select Second.cnt.major,Second.cnd.CS146
      from Second: (time: 1245 db: 1167)
8 QUERY XML Select Second.cnt.major,Second.cnd.CS135
      from Second: (time: 1262 db: 1175)
9 QUERY XML Select Second.cnt.year,Second.cnd.CS102
      from Second: (time: 1283 db: 1199)
10 QUERY XML Select Second.cnt.semester,Second.cnd.
      CS127 from Second: (time: 1250 db: 1174)
11 QUERY XML Select Third.cnt.major,Third.cnd.CS144
      from Third: (time: 1246 db: 1167)
12 QUERY XML Select Third.cnt.instructor,Third.cnd.
      CS138 from Third: (time: 1282 db: 1196)
13 QUERY XML Select Third.cnt.semester,Third.cnd.CS112
      from Third: (time: 1255 db: 1176)
14 QUERY XML Select Third.cnt.comments,Third.cnd.CS140
      from Third: (time: 1237 db: 1156)
15 QUERY XML Select Third.cnt.comments,Third.cnd.CS122
      from Third: (time: 1288 db: 1206)
16 QUERY XML Select Fourth.cnt.comments,Fourth.cnd.
      CS117 from Fourth: (time: 1243 db: 1166)
17 QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS124
      from Fourth: (time: 1237 db: 1156)
```

```
QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS107
   from Fourth: (time: 1293 db: 1213)
QUERY XML Select Fourth.cnt.major,Fourth.cnd.CS128
   from Fourth: (time: 1229 db: 1153)
QUERY XML Select Fourth.cnt.semester,Fourth.cnd.
   CS130 from Fourth: (time: 1253 db: 1169)
```

C SPO Schema Document

This is the Structured Probability Object XSD, used to validate XML describing an SPO. The original document is available at `http://www.csr.uky.edu/wtw/schema/spo.xsd`. This document was *not* created as part of this thesis.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.csr.uky.edu/wtw/schema"
  elementFormDefault="qualified"
  xmlns:spo="http://www.csr.uky.edu/wtw/schema">
<!--  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" -->


<xsd:annotation><xsd:documentation>
This is the definition for SPO.
</xsd:documentation></xsd:annotation>
<!--
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
<xsd:simpleType name="nameType">
        <xsd:restriction base = "xsd:string">
        </xsd:restriction>
  <xsd:attribute name="ID" type="xsd:ID" use="optional"/>
</xsd:simpleType>
-->

<xsd:complexType name="nameType">
  <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <!--  <xsd:attribute name="ID" type="xsd:NCName" use="optional"/> -->
      </xsd:extension>
  </xsd:simpleContent>
```

```
</xsd:complexType>

<xsd:simpleType name="valType">
        <xsd:restriction base = "xsd:string">
<!--
                <xsd:pattern value="[A-Z|0-9]+"/>
-->
        </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="elemType">
  <xsd:sequence>
    <xsd:element name="name" minOccurs="1" maxOccurs
       ="1" type="spo:nameType"/>
    <xsd:element name="val" minOccurs="1" maxOccurs
       ="unbounded" type="spo:valType"/>
  </xsd:sequence>
  <xsd:attribute name="IDREF" type="xsd:NCName" use
     ="optional"/>
</xsd:complexType>

<xsd:complexType name="contextType">
  <xsd:sequence>
    <xsd:element name="elem" minOccurs="0" maxOccurs
       ="unbounded" type="spo:elemType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="variableType">
  <xsd:sequence>
    <xsd:element name="name" minOccurs="1" maxOccurs
       ="unbounded" type="spo:nameType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="probabilityType">
  <xsd:restriction base="xsd:float">
    <xsd:minInclusive value="0.0000000"/>
```

```
    <xsd:maxInclusive value="1.0000000"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="rowType">
  <xsd:sequence>
    <xsd:element name="val" minOccurs="1" maxOccurs="unbounded" type="spo:valType"/>
    <xsd:element name="P" type="spo:probabilityType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tableType">
  <xsd:sequence>
    <xsd:element name ="variable" type="spo:variableType"/>
          <xsd:element name="row" minOccurs="1" maxOccurs="unbounded" type="spo:rowType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="conditionalType">
  <xsd:sequence>
    <xsd:element name="elem" minOccurs="0" maxOccurs="unbounded" type="spo:elemType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="pathType">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="spoType">
      <xsd:sequence>
    <xsd:element name ="context" type="spo:contextType"/>
```

```
    <xsd:element name ="table" type="spo:tableType"/>
    <xsd:element name ="conditional" type="spo:conditionalType"/>
      </xsd:sequence>
      <xsd:attribute name="path" type="spo:pathType" use="required"/>
</xsd:complexType>

<xsd:complexType name="sposType">
      <xsd:sequence>
    <xsd:element name="spo" minOccurs="1" maxOccurs="unbounded" type="spo:spoType">
      <xsd:key name="myId">
        <xsd:selector xpath="./table/variable"/>
        <xsd:field xpath="name"/>
      </xsd:key>

      <xsd:keyref name="myIdref" refer="spo:myId"> <!-- Note spo:myId -->
        <xsd:selector xpath="./context/ele"/>
              <xsd:field xpath="@IDREF"/>
      </xsd:keyref>
    </xsd:element>
      </xsd:sequence>
</xsd:complexType>

<xsd:element name="spos" type="spo:sposType">

</xsd:element>

</xsd:schema>
```