# Chapter 4. Prototype Implementation

In this chapter, we implement the benchmark model of this research on a heterogeneous information integration prototype platform. This prototype illustrates the feasibility and validity of the research model. The architecture and design of the prototype is described in the following sections.

## 4.1. Prototype Platform and Architecture

For this research, the prototype is implemented on a heterogeneous information integration prototype platform. There are three data sources of the platform, the Microsoft SQL server, the Tamino XML server, and the Web page repository. The prototype system uses client/server architecture, as Figure 4.1 shows. The client end interface is Web browser, and a user can operate the workload generator prototype through the browser easily. We use Microsoft Internet Explorer 6.0 as Web browser, and Microsoft Internet Information Services 5.0 as Web server. Database servers we adopted are Microsoft SQL server 2000 and Tamino XML server 4.1.4.1. Table 4.1 illustrates the development environment of the prototype.

Table 4.1: The Description of Prototype System Platform Specification

| | |
|---|---|
| **Operation System** | Windows 2000 Server |
| **Web Server** | Microsoft Internet Information Services 5.0 |
| **Database Server** | Microsoft SQL Server 2000<br>Tamino XML Server 4.1.4.1 |
| **Client** | Microsoft Internet Explorer 6.0 |
| **Application Program Language** | Active Server Pages, Java Server Pages, JavaScript |

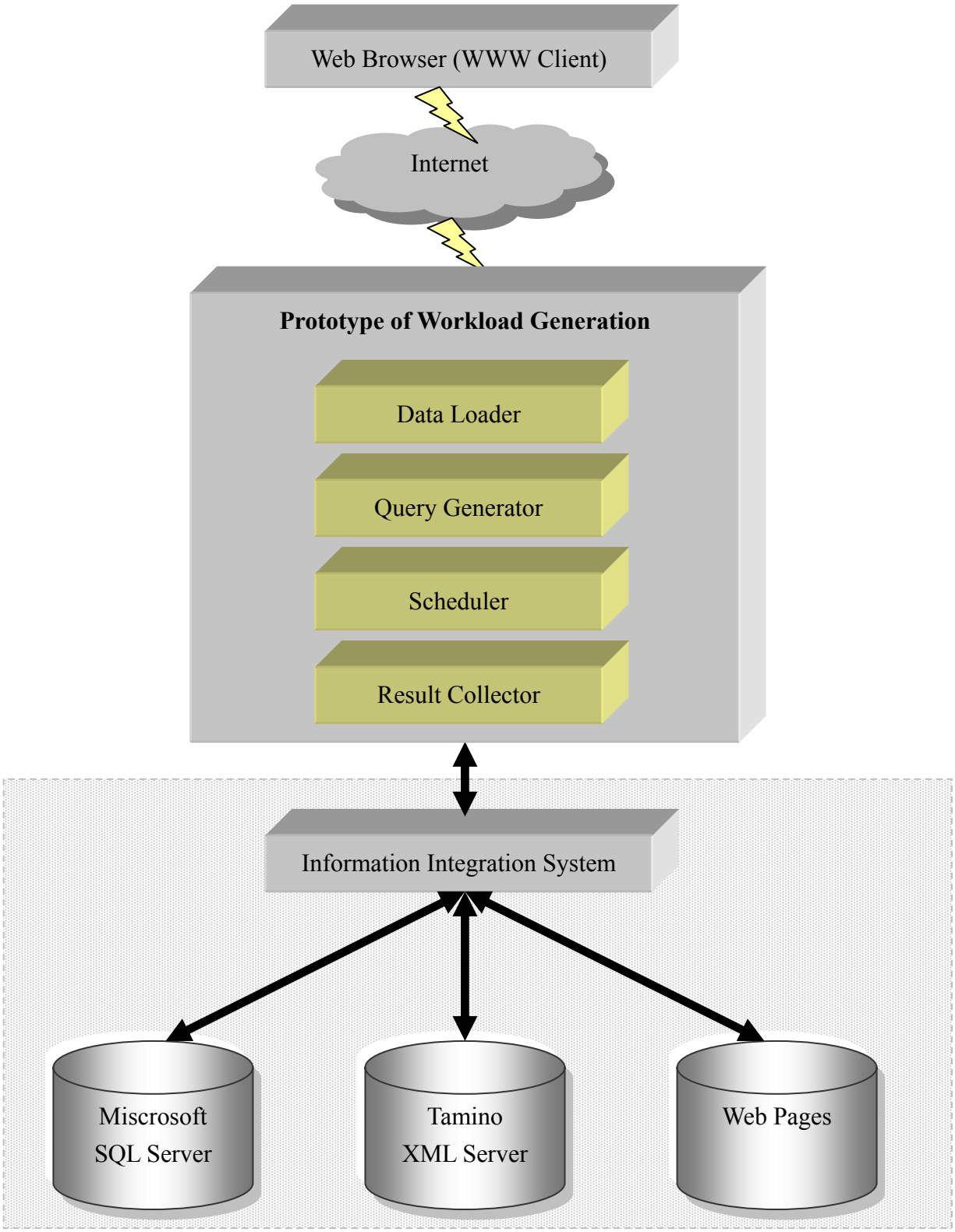



Figure 4.1: Prototype Architecture

## 4.2. Experimental Design

Due to hardware limitations, the prototype and three data sources are located on the same machine. If each of them is as large as the logical size of physical memory, the system may possibly become overloaded. Therefore, we should adjust the logical size of the test database. Table 4.2 shows the logical database size and query response time with a simple query test. The tuple length is 100 bytes on average. The size of the test database can be scaled from 100 bytes to 10 megabytes by varying the number of tuples from 1 to 100,000. The physical storage required to store the database in the DBMS under test may be higher due to non standard implementation of data types and DBMS overhead. We use the query analyzer of the SQL server to execute a simple select query at different size levels. Each query is executed in cold mode. The response time is shown on Table 4.2. When the logical size of the test database is scaled to 10 megabytes, the response time increases to 5 seconds. It is too time-consuming to execute the query on a bigger logical size. For that reason, we choose 10 megabytes as the logical size of the test database.

Table 4.2: Query Response Time at Different Logical Database Sizes

| Number of Tuples | Logical Database Size (bytes) | Response Time (second) |
|---:|---:|---:|
| 1 | 100bytes | 0 |
| 10 | 1Kilobyte | 0 |
| 100 | 10Kilobytes | 1 |
| 1,000 | 100Kilobytes | 1 |
| 10,000 | 1Megabyte | 1 |
| 100,000 | 10Megabytes | 5 |

For the prototype, we assume that the logical database sizes of three data sources

are uniform. For the SQL server, each relation has the same average tuple width and the same number of tuples. Similarly, each schema on the Tamino XML server has the same average instance size and the same number of instances. The file sizes of Web pages are standardized, too.

We use the university campus as the test scenario of the benchmark prototype. It describes universities and departments and the activities that occur at them. The global schema for the information integration prototype system is shown on Appendix A. There are six relations on the SQL server: Department, Teacher, Student, Teaching Assistant, Course, and Study. Their relationships are shown on Figure 4.2. For each relation, the attribute lengths sum up to 100 bytes, as Appendix B shows. In some relations, an additional attribute, fill, is provided to comply with the 100 byte tuple width requirement.

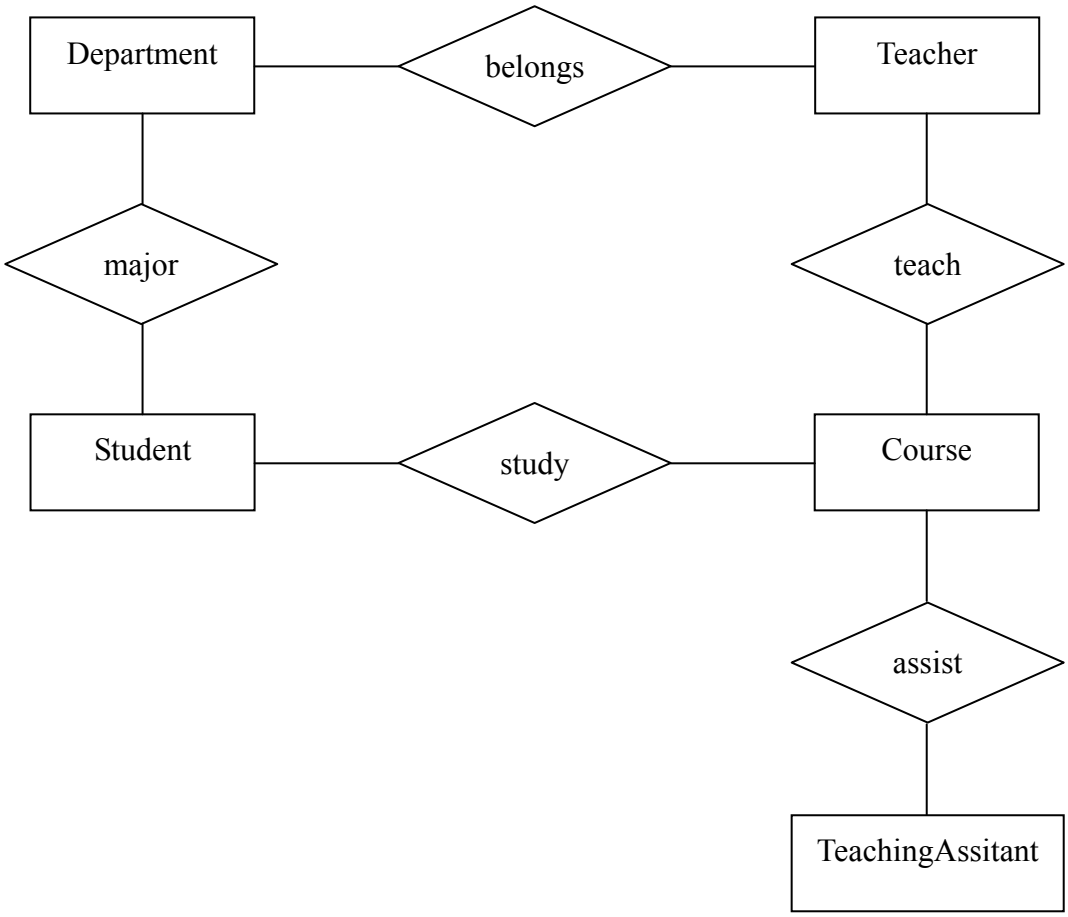



Figure 4.2: ER Diagram of SQL Server

For the Tamino XML server, we have defined six schemas: Department, Faculty, Student, Grad Student, Course, and Publication. The structure of each schema is shown on Figure 4.3 to Figure 4.8. These schema details are shown on Appendix C. When generating each schema's instance, we fix the average size at 1 kilobyte.

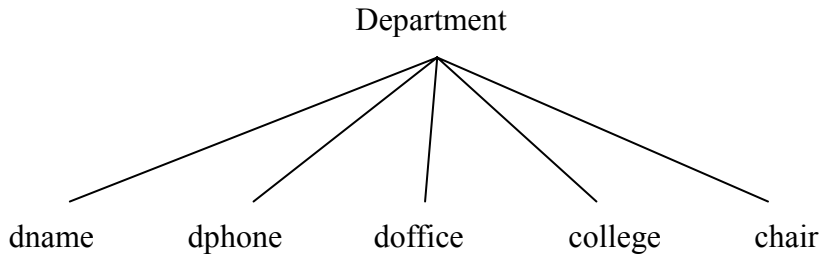



Figure 4.3: Tree Structure of Department Schema

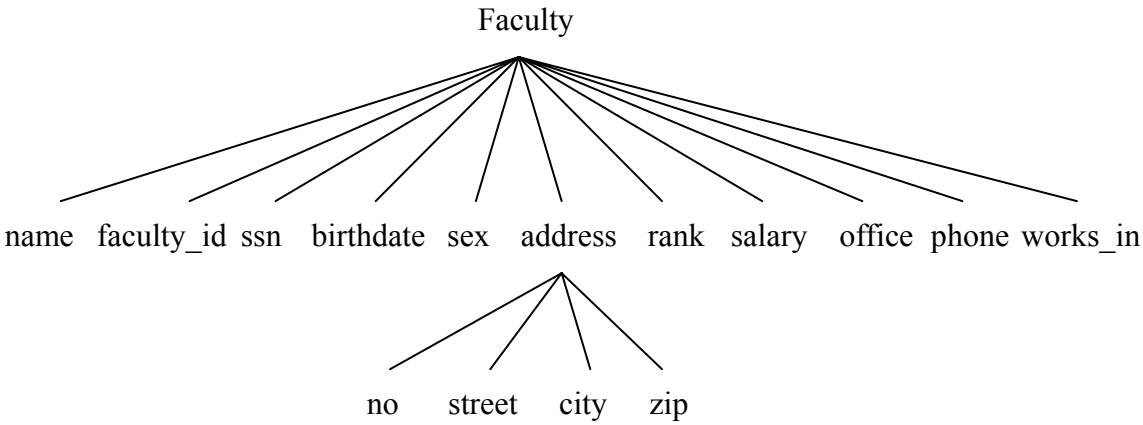



Figure 4.4: Tree Structure of Faculty Schema

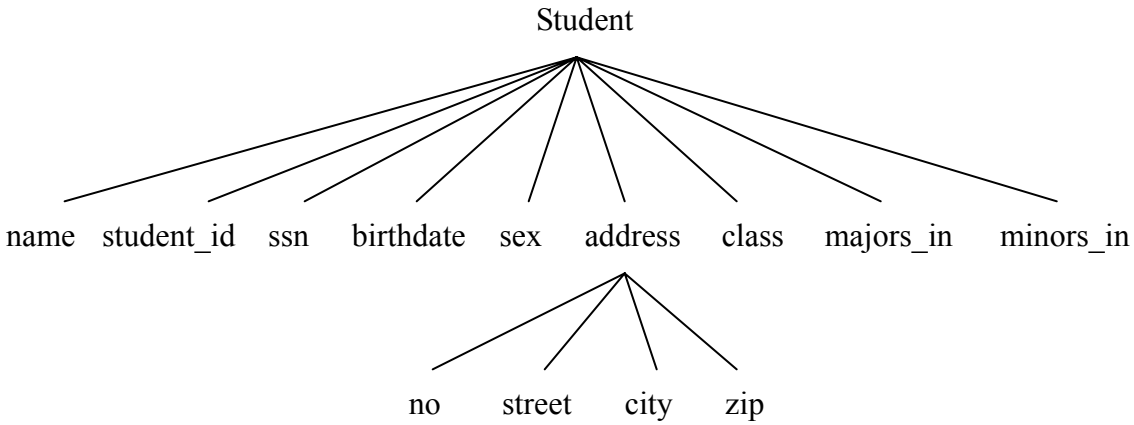



Figure 4.5: Tree Structure of Student Schema

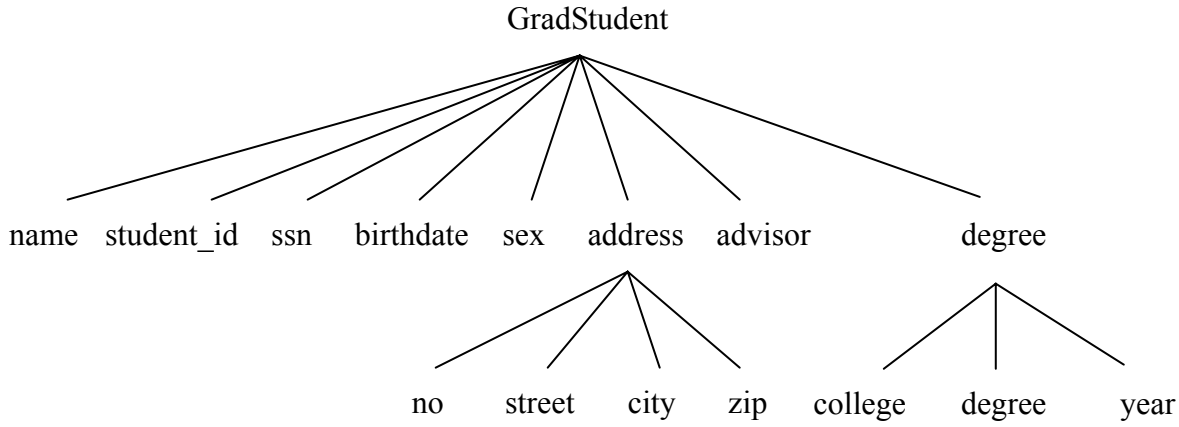



Figure 4.6: Tree Structure of GradStudent Schema

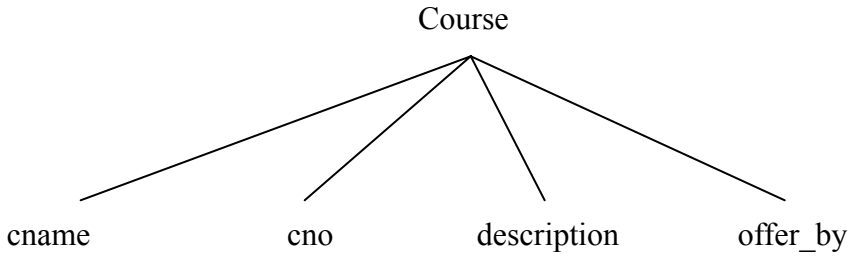



Figure 4.7: Tree Structure of Course Schema

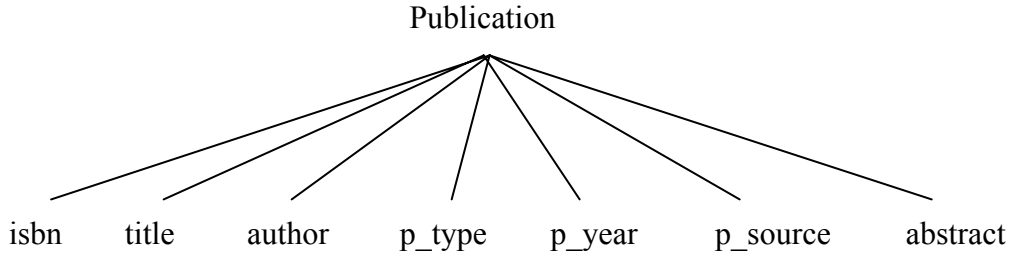



Figure 4.8: Tree Structure of Publication Schema

For the Web page repository, each Web page is 1 kilobyte in size on average.

## 4.3. Prototype Functions

We must follow the research model to implement the prototype system. Therefore, the prototype features several primary functions including the data loader, the query generator, the scheduler, and the result collector. Figure 4.9 shows the system functions and Figure 4.10 shows the main menu of the prototype system. Each function is described in the following sections.

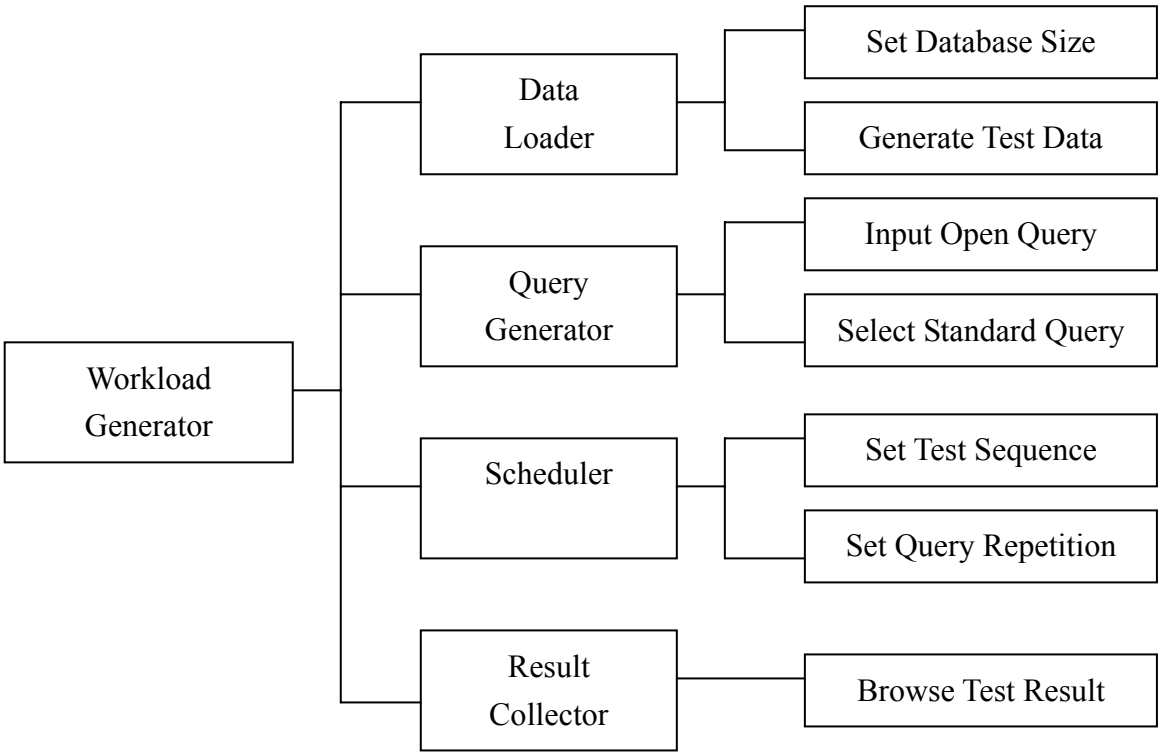



Figure 4.9: Prototype Function

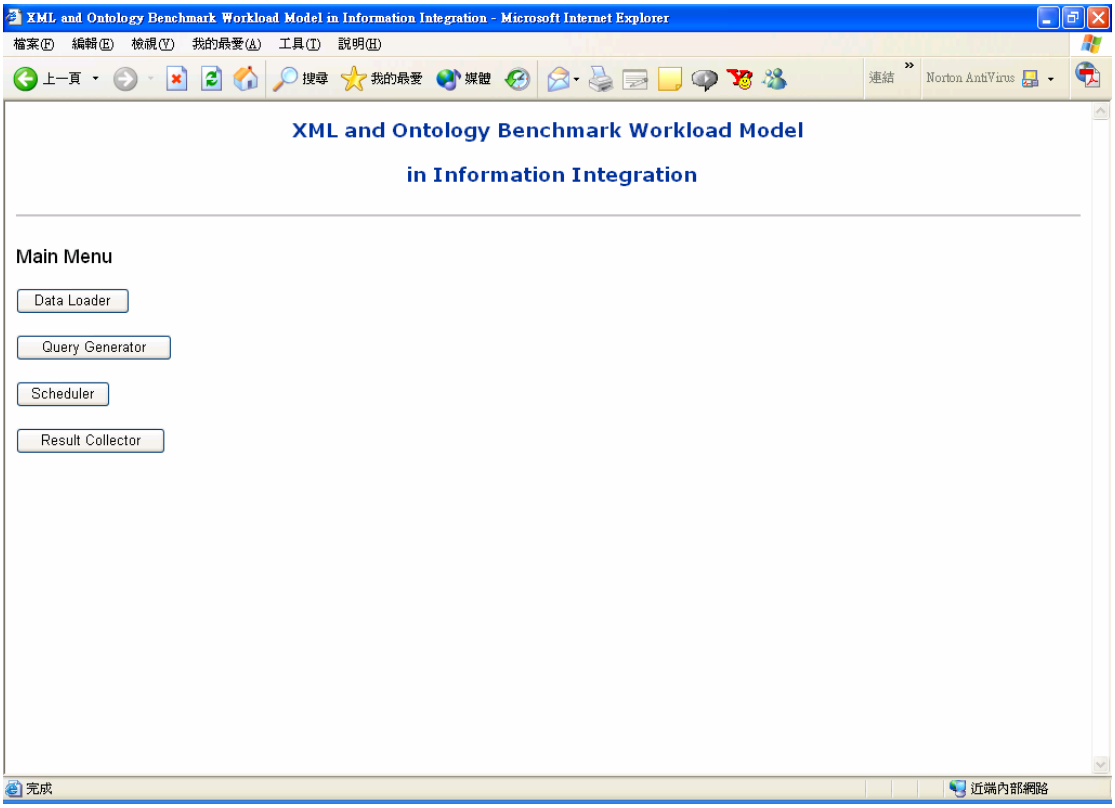



Figure 4.10: The Main Menu of the Prototype

### 4.3.1. Data Loader

In the data loader, users can define the logical database size (Figure 4.11). The default logical database sizes for each data source are uniform and users can modify them individually (Figure 4.12). The data sources and data schema in the prototype are described in the previous section. Furthermore, in the SQL server users can adjust the number of tuples loaded into each relation (Figure 4.13). Similarly, users can alter the number of instances generated for each schema in Tamino XML server.

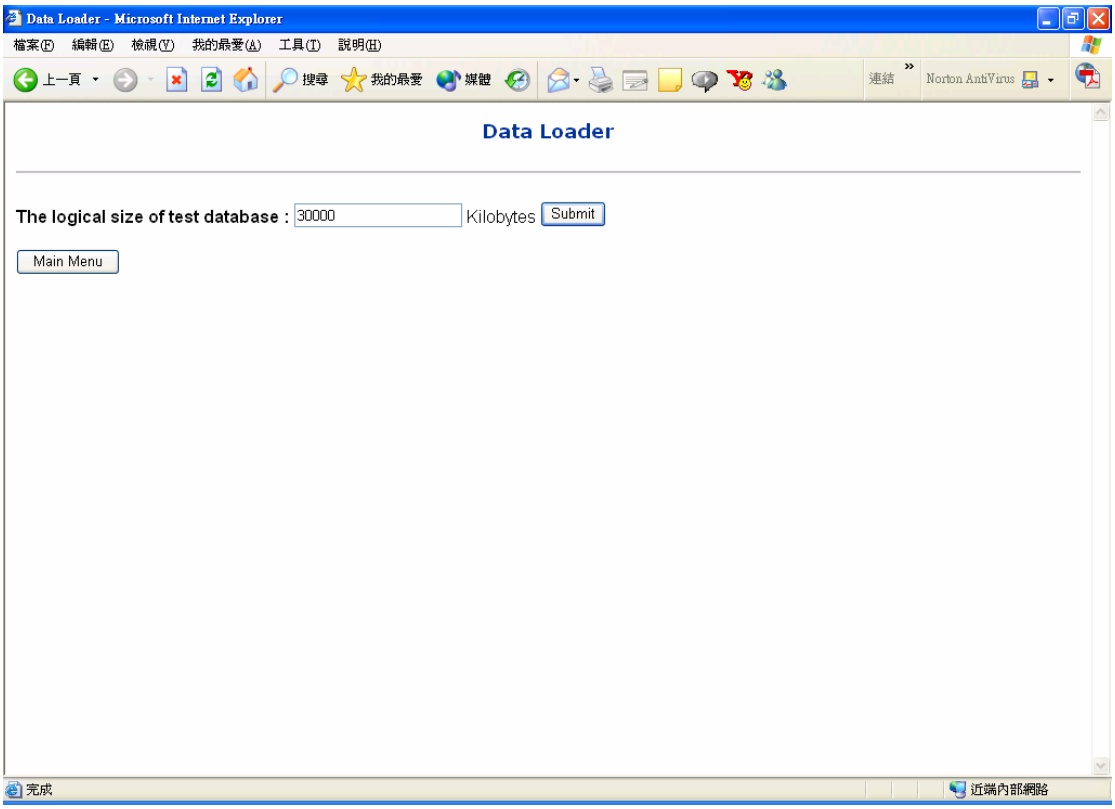



Figure 4.11: Data Loader – Set Logical Database Size

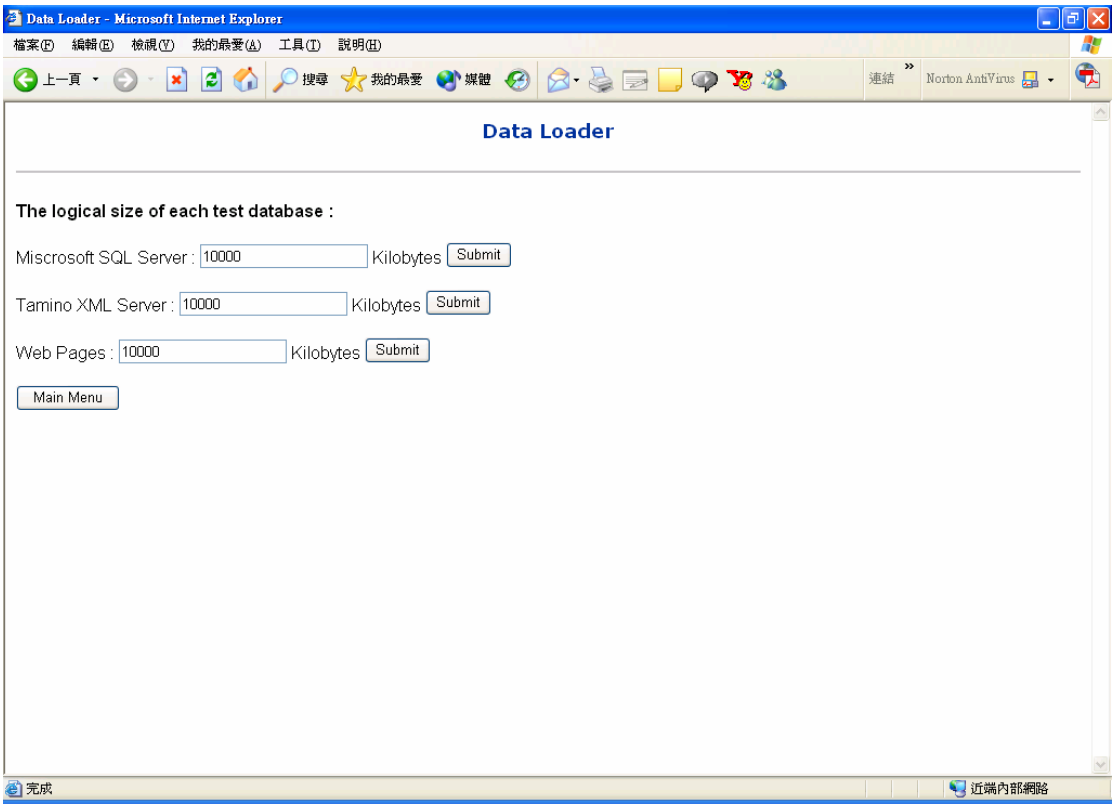



Figure 4.12: Data Loader – Modify Logical Database Size of Each Data Source

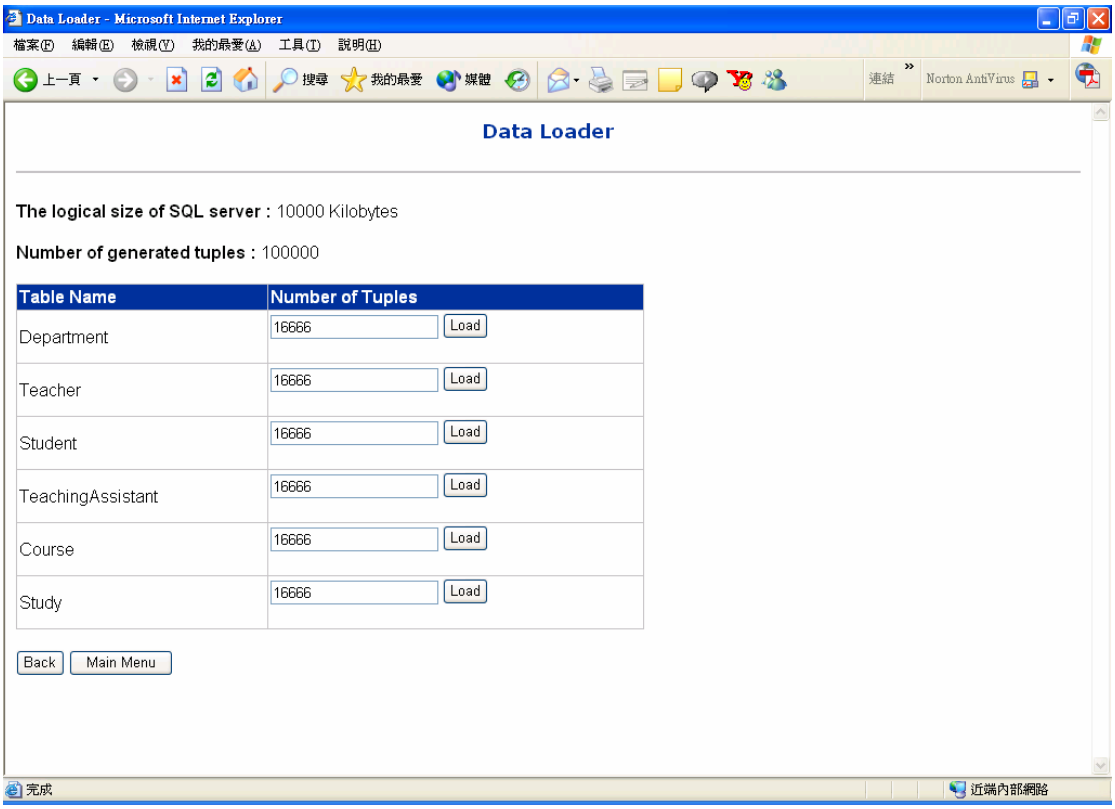



Figure 4.13: Data Loader – Modify Number of Tuples of Each Relation

### 4.3.2. Query Generator

The query generator can divided into two parts: pure XML query and XML query combined ontology reasoning (Figure 4.14). Users can evaluate pure XML processing performance of heterogeneous information integration systems or combined ontology reasoning services. In each of them, users can input the queries they want to test (Figure 4.15) or simply select the standard query types predefined in the query selector (Figure 4.17). The queries are generated according to the XML query model we defined in chapter 3. In open query input, users can specify several queries according to global schema and their requirements. All specified queries will be executed in one test, and users can delete any one in the open query list (Figure 4.16). In the standard query selector, after choosing the standard test queries, users

can specify the complexity of each query further (Figure 4.18). It would help users to evaluate the system performance at different complexity levels. The complexity of query is determined by the complexity factor defined in previous chapter, too.

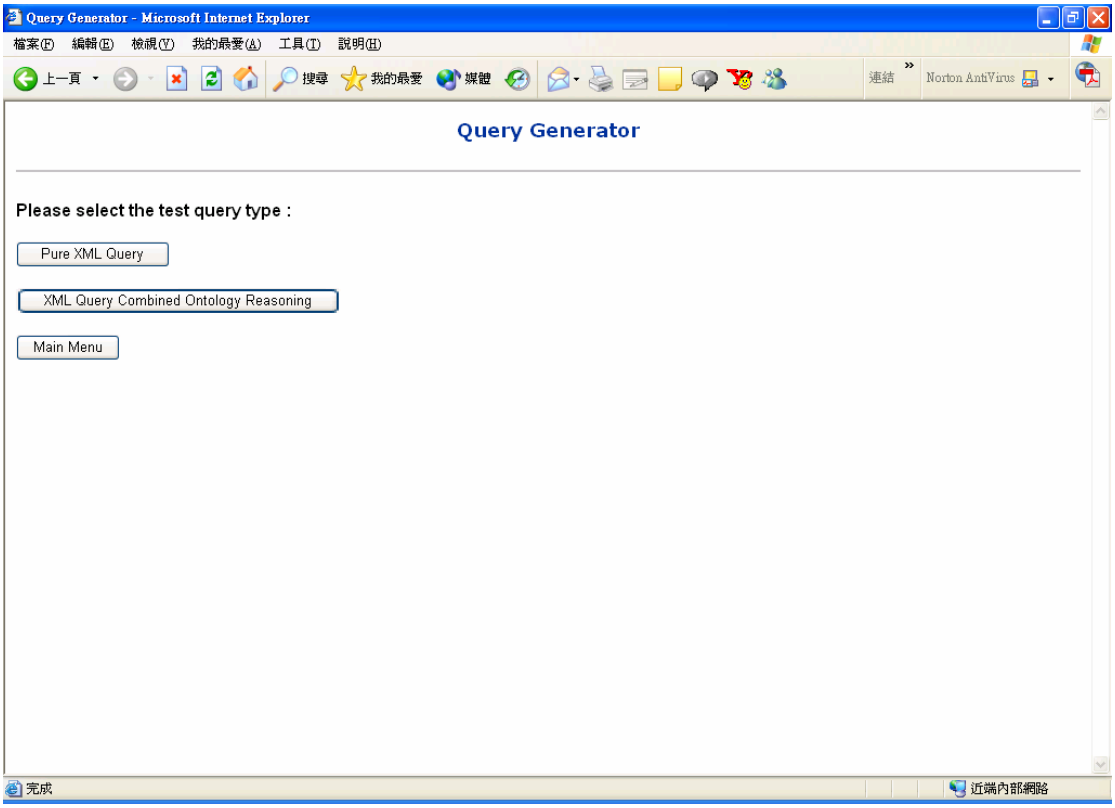



Figure 4.14: Query Generator – Select Test Query Type

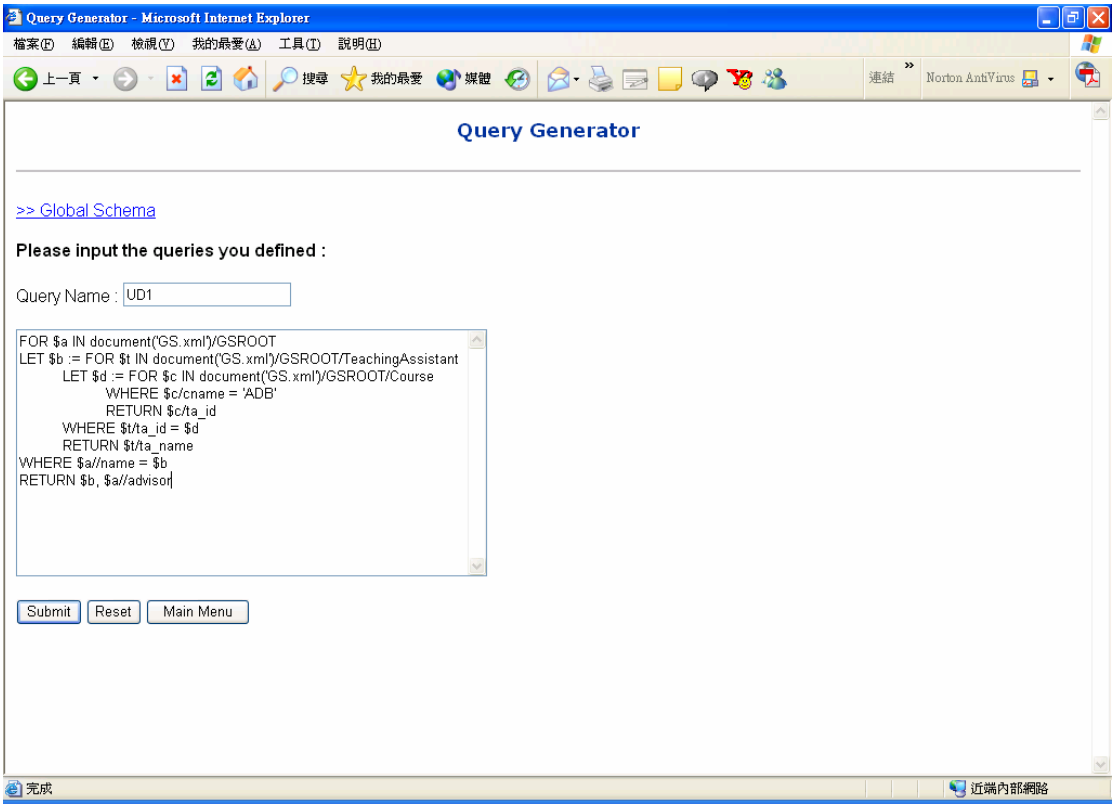



Figure 4.15: Query Generator – Input Open Query

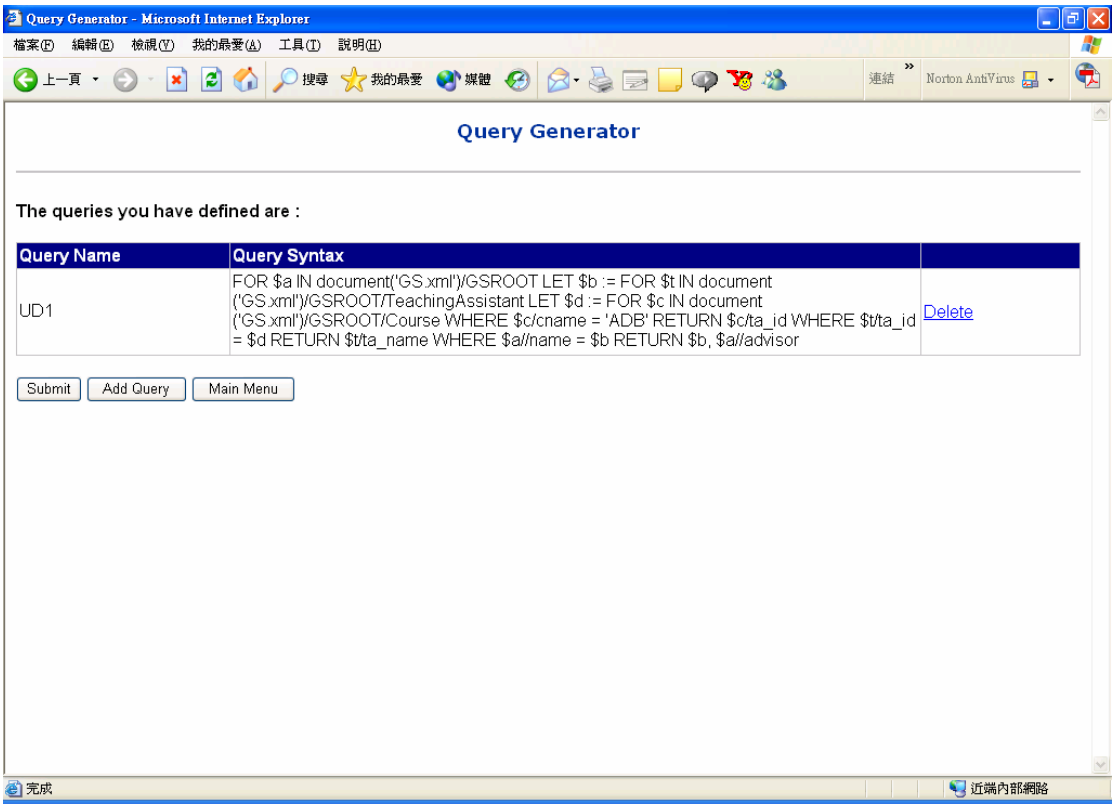



Figure 4.16: Query Generator – Delete Open Query

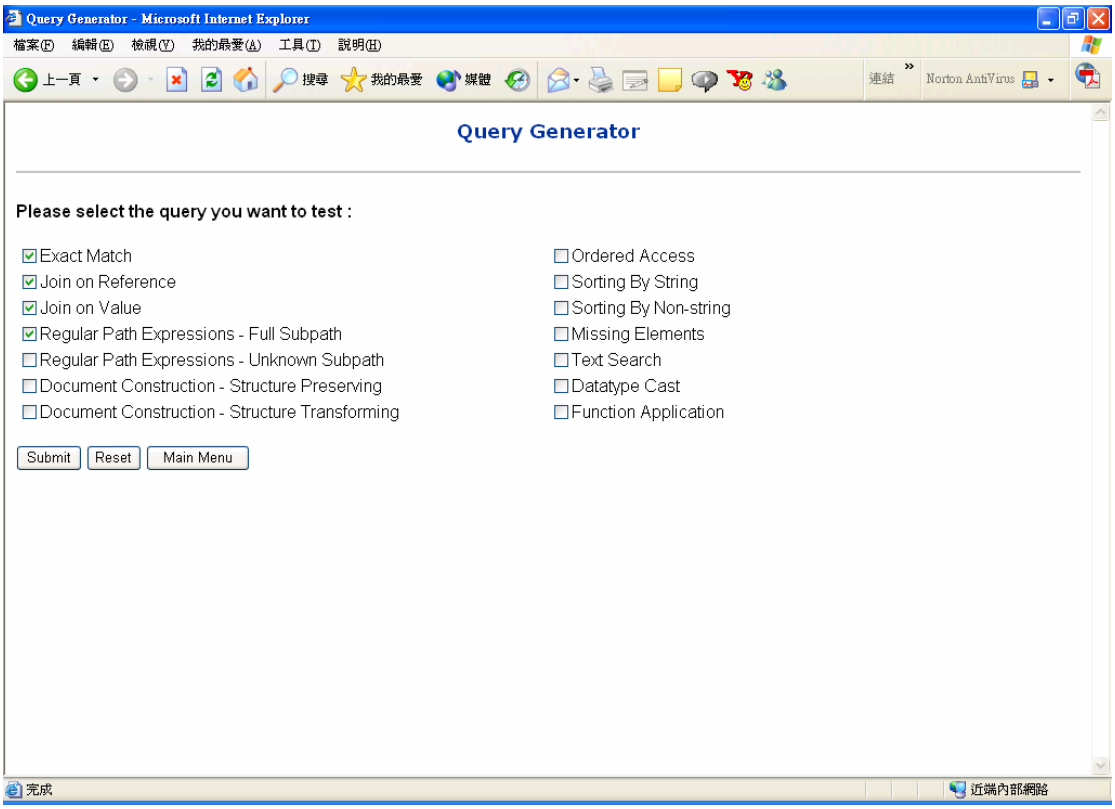



Figure 4.17: Query Generator – Select Standard Query Type

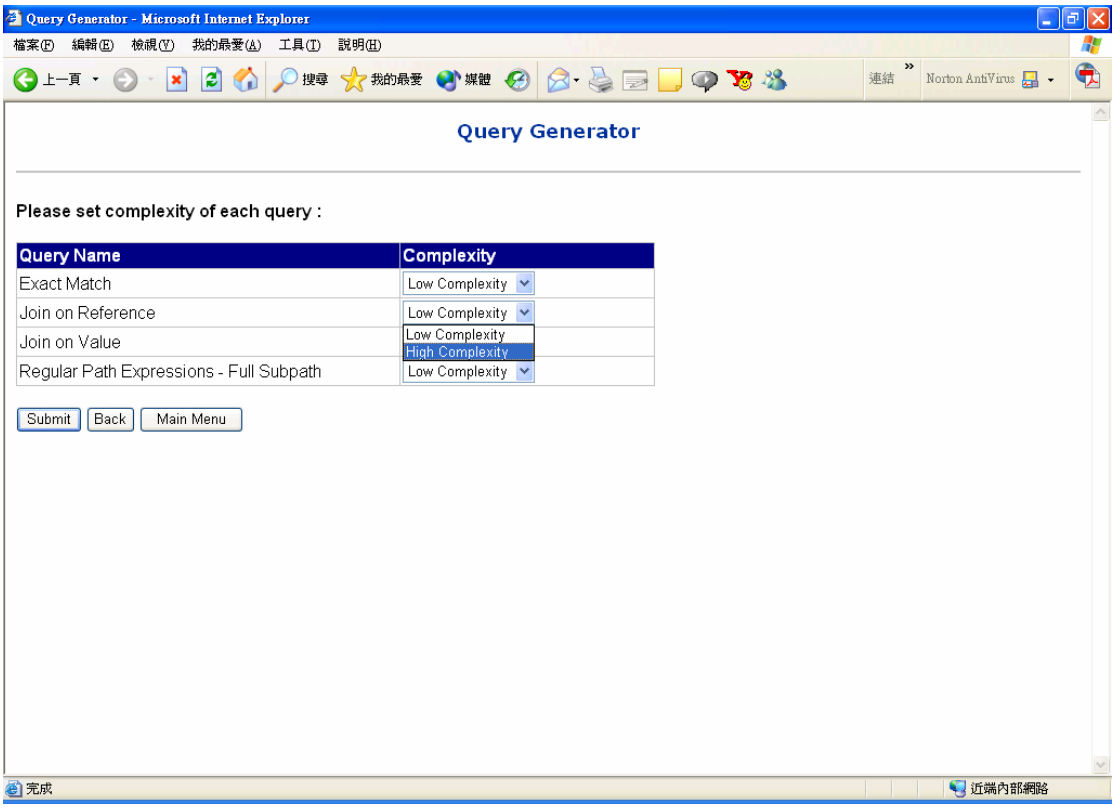



Figure 4.18: Query Generator – Set Standard Query Complexity

### 4.3.3. Scheduler

According to the control model, several parameters should be set to execute the benchmark. The parameters we implement in the prototype are test sequence and number of repetitions. Both in the open query input and in the standard query selector, once the test query set has been determined, users can set up the executed sequence (Figure 4.18) and the repetitions of each query (Figure 4.19) in the scheduler.

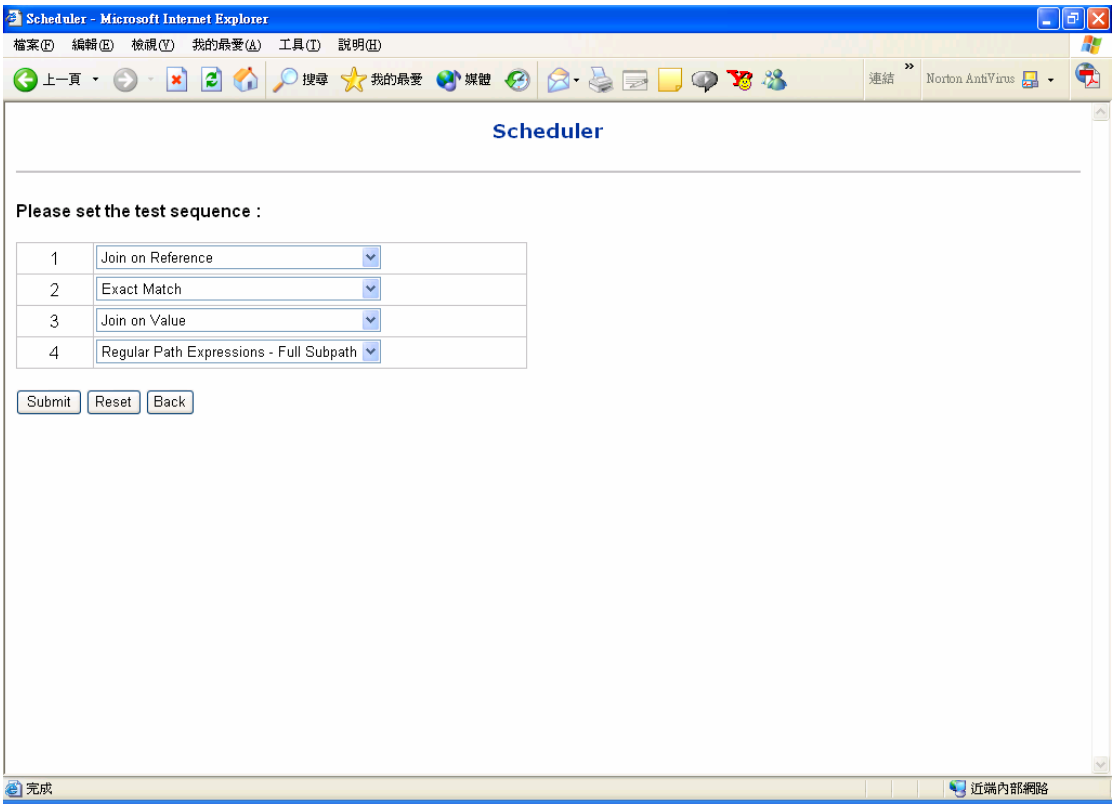



Figure 4.19: Scheduler – Test Sequence

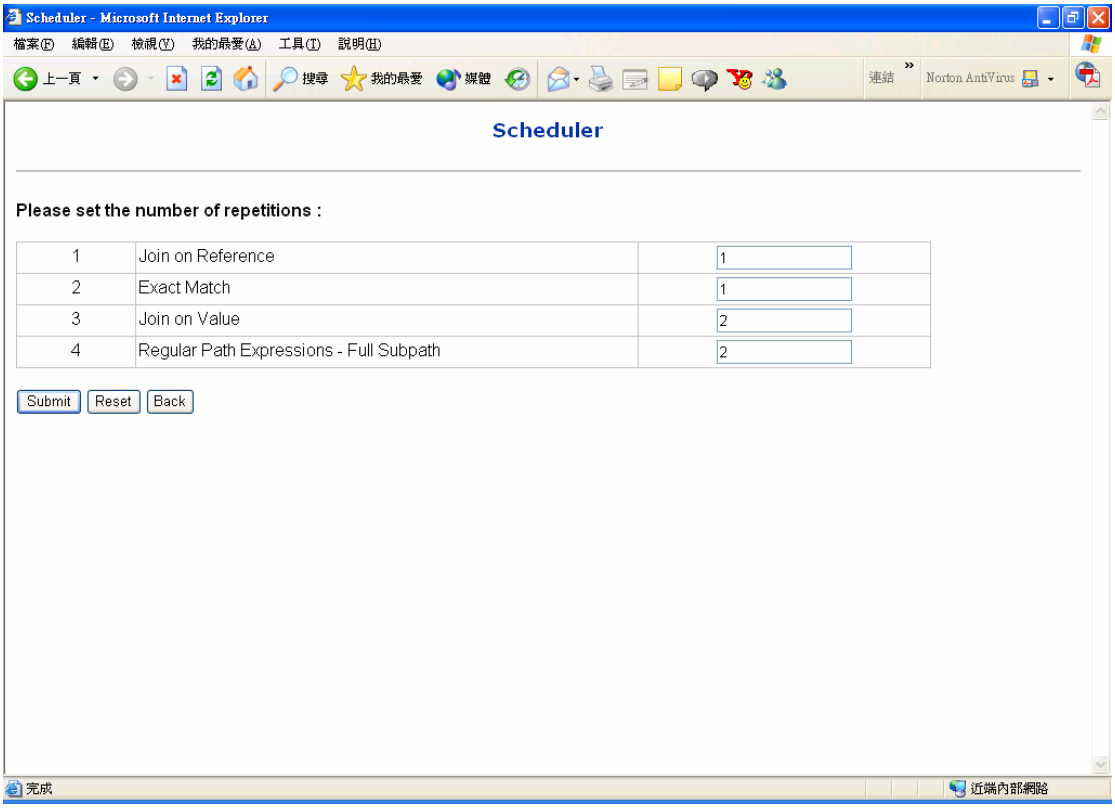
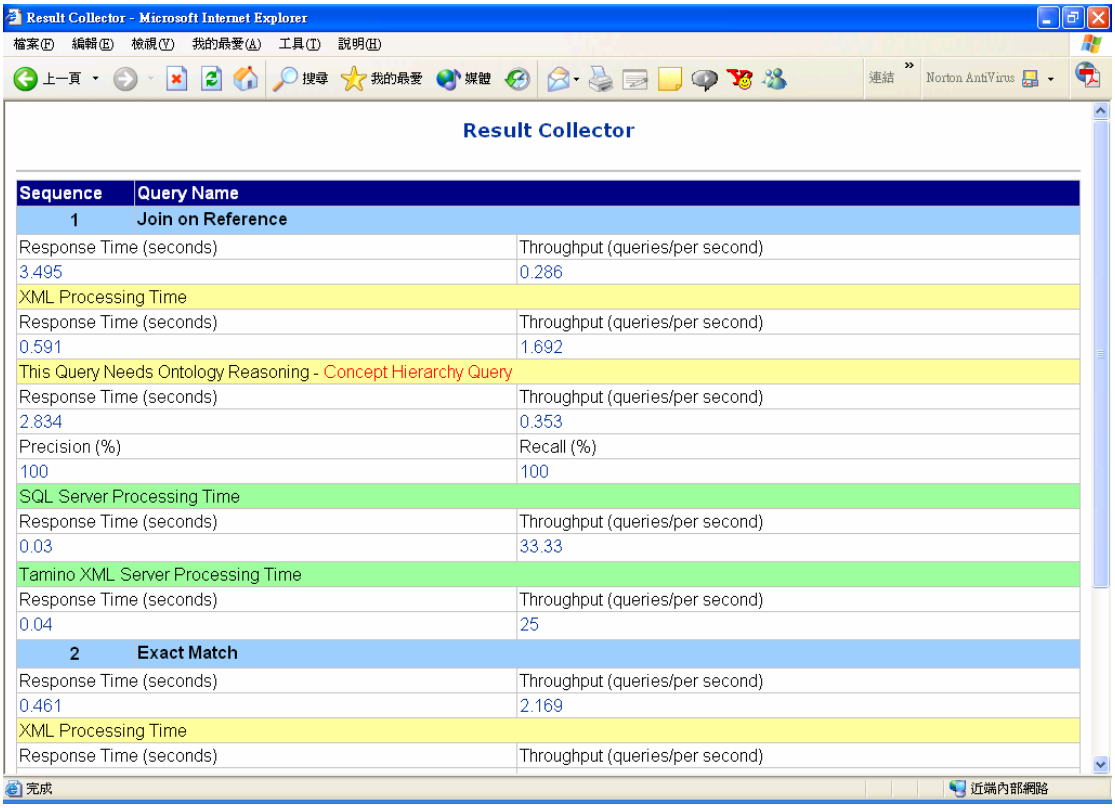



Figure 4.20: Scheduler – Test Repetition

### 4.3.4. Result Collector

The result collector shows the test results of the queries we specified. The test results can be divided into three parts: the total execution result, the XML query test result, and test result of each data source (Figure 4.21). If the test query needs to combine ontology reasoning service, the ontology query test result will be shown in the test result. In the total execution result, the total query response time and throughput are illustrated. In the XML query test result, it shows the query response time and throughput of the XML query processing. In the ontology query test result, it adds two extra performance metrics: recall and precision. This can help users evaluate the quality of answer entailed by the ontology. Finally, the result collector lists the query response time and throughput of each data sources.




## Result Collector

| Sequence | Query Name | |
|---|---|---|
| 1 | Join on Reference | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 3.495 | | 0.286 |
| XML Processing Time | | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 0.591 | | 1.692 |
| This Query Needs Ontology Reasoning - Concept Hierarchy Query | | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 2.834 | | 0.353 |
| Precision (%) | | Recall (%) |
| 100 | | 100 |
| SQL Server Processing Time | | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 0.03 | | 33.33 |
| Tamino XML Server Processing Time | | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 0.04 | | 25 |
| 2 | Exact Match | |
| Response Time (seconds) | | Throughput (queries/per second) |
| 0.461 | | 2.169 |
| XML Processing Time | | |
| Response Time (seconds) | | Throughput (queries/per second) |

Figure 4.21: Result Collector