Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master's Thesis No. 3305

Adaptation of the Data Access Layer to Enable Cloud Data Access

S.M. Mohsin Reza

Course of Study:	Information Technology
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Steve Strauch
Commenced:	March 01, 2012
Completed:	August 31, 2012
CR-Classification:	C.2.4, D.2.11, D.2.12, H.2.1, H.3.4

Abstract

In the current era of technology, Cloud computing has become significantly popular within enterprise IT community, as it brings a large number of opportunities and provides solutions for user's data, software, and computations. As part of the Cloud computing the service model Database-as-a-Service (DBaaS) has been recognized, where application can access highly available, scaled, and elastic data store services on demand with the possibility of paying only for the resources are actually consumed. While enterprise IT becoming larger these days, the current challenges are to manage the traditional database with entire enterprise data. One possible solution is to move the application data to the Cloud and then accessing Cloud data from the traditional application on local server. Thus, ensuring the use of economies of scale and reducing the capital expenditure of enterprise IT.

Moving data layer to the Cloud introduces an issue how an application can access data from the Cloud data store services with full functionality of accessing like traditional database service. To ensure this possibility, the application needs to be implemented a Data Access Layer (DAL) separately in order to enable access to Cloud data, where DAL is responsible for encapsulating the data access functionalities and interacts with business logic within the application system. Thus reduces the application complexity and brings the solutions for managing entire enterprise, data. However, accessing heterogeneous data store services the DAL requires implementing necessary adaptations.

This master's thesis focuses on investigating the adaptations of SQL statements required for accessing Relational Database Management Systems (RDMS) in the Cloud. In this scope, we perform testing on several RDMS (i.e. MySQL, Oracle, PostgreSQL) in different Cloud services in order to determine the required adaptations. However, the adaptations are to be implemented in DAL for enable accessing Cloud data. Evaluating the adaptations of SQL statements, a software application called SQL Evaluation tool has been developed in this master's thesis, where the application has implemented a DAL explicitly and is capable to execute the SQL statements simultaneously in different Cloud data store services. The purpose of developing this application is verifying the concept of adaptation of DAL.

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

Cloud computing is one of the most significant discovery over the last decade within enterprise Information Technology (IT). It is an Internet-based computing paradigm for offering computing resources such as software, platform or even networking infrastructure through Web interfaces as well as standardized protocols. The benefit of using Cloud computing is that customers uses various computing resources including business applications, services, storages, and virtual servers on-demand pay-as-consume basis [MG11]. Moreover, it brings opportunity to an application to perform necessary computations through a required number of computers that are running on online. In recent year, Cloud computing has considerably become popular within the world of enterprise IT, as it brings an increasing number of opportunities and provides solutions for the enterprises. The most important progress is that an application can use highly available resources. Besides, an industry can be benefited by reducing capital expenditure and minimizing maintenance cost, as the resources are available on-demand. While enterprise IT becoming larger, a traditional infrastructure may not be adequate to perform required services. A simple solution could be to enlarge the IT infrastructure, but it would be cost effective. To take advantage of Cloud computing, the concept is that instead of increasing an infrastructure, it can be transformed into operational cost. Thus, it will leverage the economies of scale in overall estimation, as the operational cost may always lower than adding a new infrastructure [CAP], [Bak10].

With the concept of layering based application architecture, a typical application is built using three layers consisting of a presentation layer, a business logic layer, and a data layer [Eck95]. According to the application architecture, the presentation layer is the top most layer, which is responsible for displaying the resulting information and illustrating the application-user interactions. Besides, the business layer is responsible for realizing business logic and performing comprehensive processing for controlling the functionality of an application. The data layer is consisting of database server and responsible for managing application data storage. Nevertheless, this thesis focuses on a four layers Cloud application architecture model consisting of Presentation Layer (PL), Business Logic Layer (BLL), Data Access Layer (DAL), and Database Layer (DBL). While comparing with typical application architecture, the data layer is subdivided into a data access layer and a database layer, where DAL is placed between business layer and database layer and responsible for encapsulating the data access functionality. The DBL is the lower layer of an application system and responsible for data persistence and data manipulation.

Within Cloud computing there are four deployment models introduced such as Private Cloud, Public Cloud, Community Cloud, and Hybrid Cloud [MG11]. Using diverse Cloud deployment models individual application layer can be hosted in a distributed manner. While enterprise IT growing larger, it might appear a question how entire enterprise data could be managed with a traditional database server. Until today, one considerable solution would be to move the whole application with entire data to the Cloud. However, this is not an adequate solution, as with this approach the existing infrastructure may become unused. Therefore, this thesis focuses on moving the application data completely or partially to the Cloud according to their need and then accessing Cloud data from the traditional application on local server. Additionally, to perform Cloud bursting, data analysis or backup and archiving, usually the application data is moved to the Cloud. Now it is an important issue that how application can access Cloud data store services with standard functionality of accessing a traditional database system. To en ensure this opportunity,; the data access layer has to be adapted accordingly in order to enable appropriate data access to the Cloud. The main goal of this master's thesis is to investigate the adaptations of SQL statements required to enable access to the database layer hosted in the Cloud using storage and database services of established Cloud providers (i.e. Amazon Web Services (AWS)).

The required adaptations of DAL are determined by performing a set of Structured Query Language (SQL)-Standard based tests. The initial consideration of investigating the required adaptations of DAL was based on tests of an example application (e.g. Customer Relationship Management (CRM) application openCRX) hosting its data layer in the Cloud. Therefore, initially we have decided adapting the DAL of openCRX application, then it is realized that the data access functionality of openCRX has implemented with very complex structure and accessing database via openMDX, an open source Model Driven Architecture (MDA) application framework. To reduce the work effort and focusing on the core contributions of the thesis then we decided using the database and the data only from the openCRX application and migrate to different Cloud data store services. To evaluate the SQL support of Cloud data services, we implement a tool called *SQL Evaluation tool*. The tool offers evaluating a set of selected Cloud data store services by performing SQL-Standard based tests.

1.1. Problem Statement and Scope

Cloud computing allows provides serving DBaaS on demand via Internet, where Cloud database may be a relational or non-relational database services. The example relational database could be a traditional MySQL [Qui12] [MYS], Oracle [ORAa] or PostgreSQL [PSQ] server and the non-relational database could be a NoSQL [SDB] data store service. This enables the applications accessing highly available and scalable data store service with standard functionalities over the Web. In the current trend of technology there are great changes on world of enterprise IT where the technologies are still in progress. The industries are fundamentally transforming, the enterprise applications and enterprise data are getting larger and larger these days.

While the enterprise data are still growing, the current challenge is to manage the database with entire enterprise data. With the large amount of data, using present infrastructure and the traditional database is not sufficient to guarantee the data scalability as well as availability and stability. However, for the betterment a possible solution may be increase the IT infrastructure which is very cost effective and not an ideal solution, as the infrastructure may require further enlargement due to handling newly upcoming a large amount of data. Even though if the infrastructure is increased radically, now the question is how the traditional database server keeps their services alive. While the current infrastructure is not sufficient to manage the data, the applications or platforms or even collaborations may migrate to the Cloud.

To reduce the capital expenditure, profit from economies of scale and ensuring solutions for above problems the current approach is to move the database layer to the Cloud for an existing application, where the application can benefit accessing database from the Cloud services on demand via Internet. This means that the application including a presentation layer, a business logic layer, and a data access layer are remained at previous place in the local server, only database layer will move to the Cloud. Then application will access data and computations from the Cloud services only whenever they need. Moreover, the concept is that an application also can access various Cloud services and data storages simultaneously for the different computations purposes. This enables vast advantages for the traditional application systems, as different Cloud services established with unique and particular goals. And different computations are appropriate in specific Cloud service.

As Cloud providers offers implementing a traditional database such as MySQL or Oracle in the Cloud environment, the application can use Cloud computing to achieve elastic data storage, optimized scaling, high availability, multi-tenancy and effective resource allocation. So that the developers and enterprises are no longer needed to be worried about the database scaling, tuning, upgrades and backups, as the several Cloud services providing solutions regarding these issues. However, managing all the data within the enterprise, it is realized that the database is the foundation for all enterprise software and moving to the Cloud the core architecture of database is changed fundamentally [Fis10].

While moving database layer to the Cloud now it is an open issue how the application can access data from the Cloud data service with full functionality of accessing as a traditional database. As a possible solution we consider adapting DAL for accessing proper data from Cloud database service where data access layer encapsulates the data access functionality. In order to implement the data access functionality, it is necessary to investigate the existing approaches for adaptation of the database layer to enable Cloud data access as well as the adaptation of the DAL to enable Cloud data access, where the existing approaches can be reused for the Catalog for DAL adaptation guidance for Cloud data access.

The main goal of this master's thesis is to investigate the adaptations of SQL statements required to enable access Cloud data store services. Therefore, we focus on SQL query language and how SQL query should be adapted to support cloud data access. To evaluate the SQL support of Cloud data services, it is considered to perform SQL-Standard based tests in order to investigate the required adaptations. Out of scope of this master's thesis is considered implementing a secure data access while application accessing data from the Cloud data store service. However, adapting the data access layer for the Cloud data access must be facilitated. Also, data encryptions as well as data replication are not focused in this work. Furthermore, this work does not present concepts for ensuring the performance of Cloud data access from traditional application.

1.2. Research Design and Questions

While enterprise data is increasing rapidly, the traditional database is becoming unable to keep the performance of data handling. It requires a big amount of data storage to perform the necessary computations; otherwise business model is failing to ensure the application requirements. Initial solution might be to increase the IT infrastructure, which is however raising the capital expenditure of a company. Therefore, to reduce the capital expenditure and transforming it into operational costs, an existing application may be moved to the Cloud or designed from the beginning to use Cloud technologies, as Cloud service providers offering resources on-demand and can achieve optimized database scaling. Figure 1.1 describes in detail steps of the master's thesis research, where it shows a clear overview of how we proof our concept of implementing and evaluating the Cloud data access.

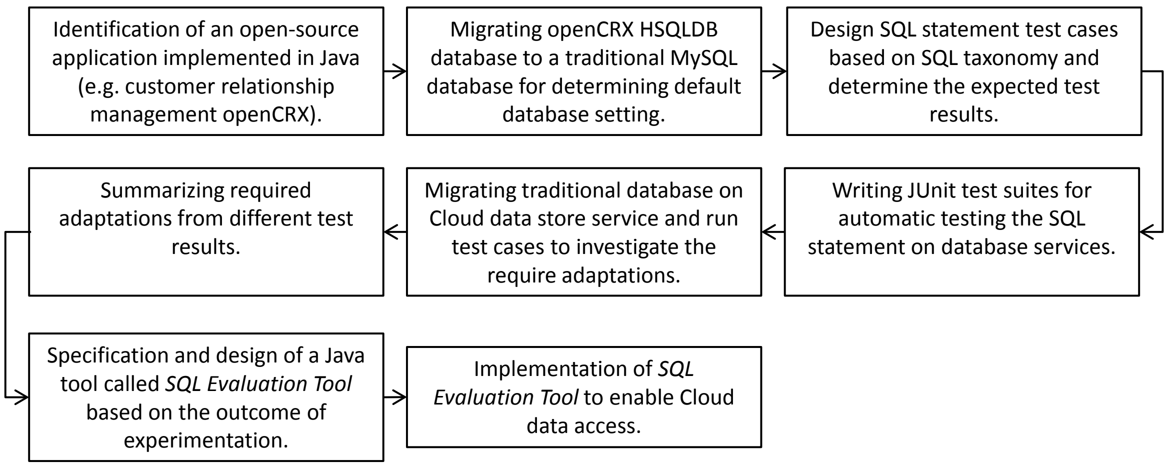

Identification of an open-source application implemented in Java (e.g. customer relationship management openCRX).
Migrating openCRX HSQLDB database to a traditional MySQL database for determining default database setting.
Design SQL statement test cases based on SQL taxonomy and determine the expected test results.
Writing JUnit test suites for automatic testing the SQL statement on database services.
Migrating traditional database on Cloud data store service and run test cases to investigate the require adaptations.
Summarizing required adaptations from different test results.
Specification and design of a Java tool called *SQL Evaluation Tool* based on the outcome of experimentation.
Implementation of *SQL Evaluation Tool* to enable Cloud data access.

Figure 1.1.: Research Design.

This master's thesis purposes its goal of investigating the required adaptations of the data access layer in order to enable Cloud data access through research on existing Cloud data access approaches with tooling support concerning migration of database layer and adapting a data access layer (i.e. a middleware). Therefore, an open-source Java application (e.g. a CRM application openCRX) has been considered, where openCRX is generically shipped with HSQLDB database. openCRX is a Java based open source Web application, which provides functionality for Sales Force Automation (SFA), customer service, project and activity management, products and services, etc. Also it is platform-independent and it runs on any platform with Apache Tomcat or any J2EE-compliant AppServer [CRM12].

Moreover, it consider defining a set of test cases based on SQL taxonomy (see Table 2.1). To find the maximum taxonomy, the test cases are designed mapping with corresponding taxonomy (see Table 4.2). The default test cases are written based on traditional MySQL database, however, to investigate the adaptation of SQL statements the same test cases have to be executed on every selected Cloud database services (see Table 4.1) to perform testing the SQL test cases. To achieve better results, it is consider selecting both relational and non-relational Cloud database services, as application might be benefited to access different data models for the different types of computations.

In order to determine the default database setting, it is considered to migrate in a traditional MySQL database, as a Cloud database can be a traditional database such as a MySQL [Qui12], which is currently supported by several Cloud database service providers. Then, there are number of SQL test cases are designed based on SQL-Standard Taxonomy. By running a SQLqueryAnalyzer we determine the expected test results. These test results are to be compared while executing the SQL test cases on different Cloud database systems for the investigation, which SQL queries and statements have to be adapted. For comparing the test result we presently using the JUnit test suites for automatic testing the SQL statement on database services.

To run the SQL test cases on Cloud data store service and achieve acceptable results, it is necessary to migrate the traditional database to Cloud database service correctly considering the database compatibility and then summarize the required adaptations from different test results after running on all possible Cloud services. The summarizing results shows the distinguish adaptations required by the different Cloud services. After determining all essential adaptations, the adaptations are to be considered for implementing in *SQL Evaluation Tool*, which will be used as proof of concept implementation of a data access layer.

This master's thesis analysis the state of art of existing approaches for adaptation of the database layer in order to enable Cloud data access and also analysis of existing categorizations of SQL statements, which might be reused for the Catalog for DAL adaptation guidance for Cloud data access. There are number of general specifications on how to realize the enabling data access layer within a Java application system and using these detailed specifications it implements the a data access layer in order to enable access to the database layer hosted in the Cloud. The concept and implementation of a Java tool (i.e. *SQL Evaluation Tool*) including a data access layer are evaluated by accessing various Cloud database services via data access layer with the knowledge of SQL statements adaptations.

All above concepts are considered to adapt a data access layer to enable Cloud data access and support for the migration of the database layer to the Cloud. The following research questions are defined the major goals, which are focused on and should be achieved in this thesis.

- What are the differences in SQL support of selected Cloud Storage Services?
- Which adaptations of SQL queries are required to support the access to various Cloud Storage Services?
- How the adaptations of SQL statements could be implemented in a data access layer which allows the accessing of heterogeneous Cloud data store services concurrently for a traditional application?
- How to implement the business logic in order to realize an adaptation of the SQL queries in a prototype?

The research questions define the goal of this thesis and they are addressed by designing, implementing and evaluating of an example Java tool (i.e. *SQL Evaluation Tool*) in order to proof the concept of implementation of data access layer.

1.3. Motivating Scenario

This section describes a concrete scenario of adapting data access layer within the application architecture model and also describes why we need a data access layer that enables access to the data has migrated to the different Cloud storage solutions. While enterprise data becoming larger and requires a high amount of computations for handling the data, a traditional application can take advantages of accessing Cloud data storage. Thus, reduces the application complexity. In Cloud computing approach, an important activity is that the data can be exchanged between different Cloud data services. This allows applications accessing multiple Cloud storage infrastructures for storing and retrieving data objects according to their need [RW12].

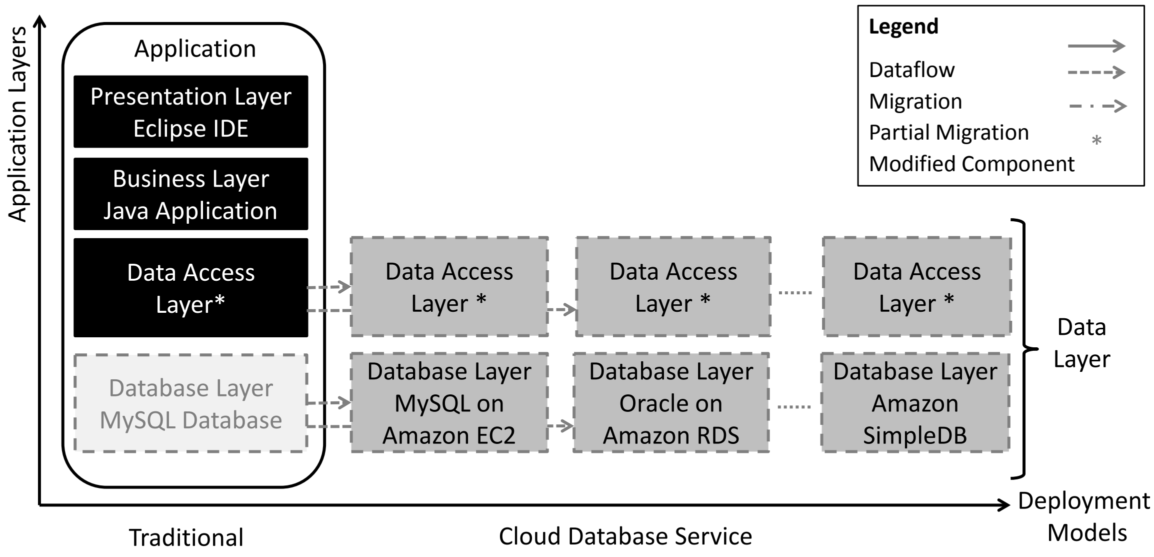


Figure 1.2.: Scenario of adapting a data access layer to enable Cloud data access.

Nowadays most of the applications are built considering a multiple layers architecture model environment. This usually includes a presentation layer, a business rules, and database layer, where database is an important part in an application architecture system. It is examined that every application (i.e. Web, Windows, or even Web service) systems are normally incorporating the database systems in order to store the application data. The database may directly access from an application resource such as a Web page. If so, this would not be an adequate solution for the application development. Because, if it is required to change the database access it will dramatically increase the application maintenance work. Therefore, to simplify the application architecture, it is necessary implementing a data access layer. The concept of introducing a data access layer is that, it makes interaction to the business logic layer with encapsulating the data access functionality to the database layer. This means that a data access layer is becoming a middleware between business logic and the database layer in the application architecture model. Additionally, the data access layer can be adapted implementing strategy to make the database independent. Then application can easily access any kind of the database systems using store procedure created in the database [Pat06].

Figure 1.2 represents a conceptual scenario of adapting data access layer within a typical application architecture model, where application accessing database systems using a set of appropriate data access functionalities. The concept of implementing a data access to enable Cloud data access is based on *SQL Azure Migration Wizard* (SQLAzureMW) tool [Mic12] [Ber10] documentation, where they describe step by step migration process of Cloud database layer migration. However, it is studied that there are various migration tools or scripts available online can be used to migrate the traditional database to the Cloud data store services. The tool SQLAzureMW allows user migrating SQL server version 2005 or greater version of SQL server to SQL Azure. As a general view, migrating a database layer to the Cloud environment, the very first step is to take a detail plan about data layer migration and ensure the development environment by choosing an appropriate tool for data migration.

According to the definition of data migration with SQLAzureMW, it is necessary analyzing the database compatibility issue between source database and the target database systems (i.e. Cloud data storage). To resolve the compatibility issues, it may require adapt the source database by modifying the database script. Upon launching the migration tool the data migration scenario shall be realized. Besides selecting the source database over the migration tool there are several options has to be selected for executing the source database script against the target one. The one important option is to select the provider, to where database should be migrated.

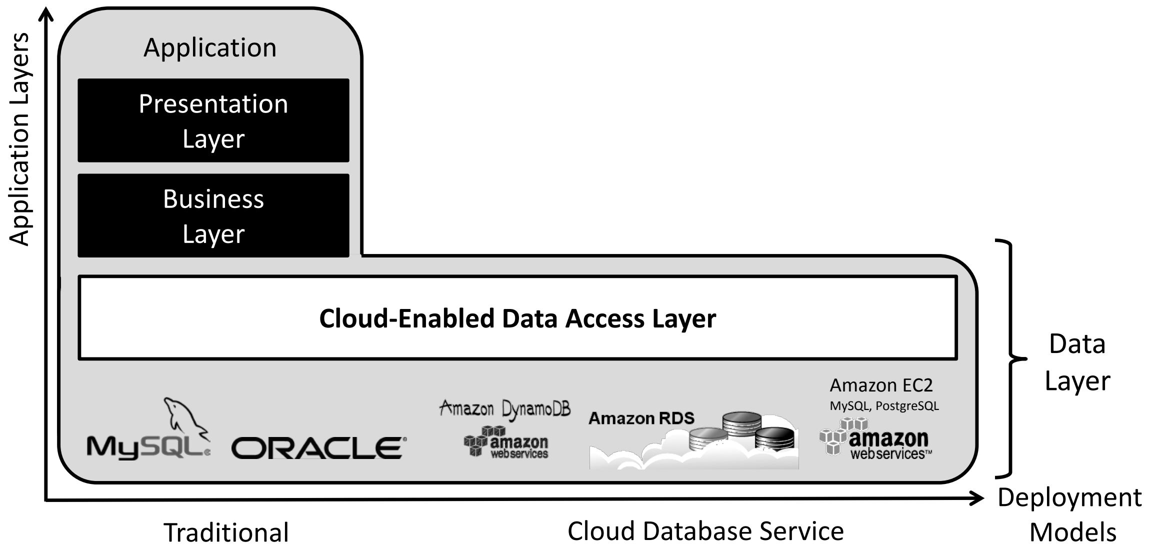


Figure 1.3.: Cloud-Enabled Data Access Layer Architecture [Str12].

Based on the selected provider, the tool will adapt the data access functionalities and these functionalities are to be considered a data access layer within the application architecture of SQLAzureMW tool. Therefore, it can be determined that the SQLAzureMW migration tool is using a data access layer for migrating database layer to the Cloud. This approach triggers our interest and proofs the concept of implementing a data access layer within a typical application architecture model to enable Cloud data access. Furthermore, there are some more database and application migration examples investigated, such as Migrating from MySQL to Amazon SimpleDB [MyS09], Netflix's Transition to High-Availability Storage Systems [Ana10] and Migration Scenario: Migrating Web Applications to the AWS Cloud [MWA10]. From all these examples we determine that it explicitly makes clear the need for a data access layer enabling cloud data access.

Figure 1.3 shows a motivational evaluation of Cloud-enabled data access Layer Architecture, where application can access Cloud enabled data store services including both traditional and Cloud based storage via an adapted data access layer. Here the data access layer should be adapted based on required data access functionalities for individual database services. So that application can access any of the database systems according to their wish. In this scenario, in order to access data from particular Cloud data store service, the application will send the SQL query including target database information to the data access layer. Then the data access layer will route the SQL query execution to the target database based on destination information. Thus, an application can access several data store services simultaneously depending on the computation needed. Moreover, based on architecture model of Cloud-enable data access layer, it may be determined that the data access layer is a middleware between business logic and the database system.

1.4. Outline

After introducing the main goals and the scope of the work, the remainder of this thesis is structured as follows.

Chapter 2 - Fundamentals: In the beginning, the important literature was studied, which covers the fundamentals of this master's thesis. This chapter describes the basic understanding in enabling a data access layer in a traditional application in order to access Cloud data and gives an explanation of Cloud computing, data hosting solutions, and different Cloud services which is focusing on RDMS data store services. Moreover, the basic rules of SQL statements and a CRM product openCRX are also explained.

Chapter 3 - Related Works: This chapter describes the common approaches and technologies that are designed for accessing multiple Cloud services including data store service within one application system. Several existing concepts and state-of-the-art systems were examined for data access layer ideas. But there was no direct related work found in the existing research. VISION Cloud introduces an abstraction of data access layer, whereas jclouds, Apache Deltacloud, and Libcloud provides different libraries for accessing multiple Cloud services with their unique purpose.

Chapter 4 - Evaluation of SQL Support of Cloud Data Service: In order to determine the required adaptations of SQL statements, this chapter describes a detailed procedure of test setup and a default scenario for SQL statement testing. Also, it illustrates the overview scenarios of running SQL test cases in different Cloud data store services and represents the test results counting adaptations which are required for every selected Cloud services. Finally, this chapter concludes a summary result with required adaptations of SQL statements.

Chapter 5 - Concept and Specification: This chapter emphasizes more precise concepts of the thesis by considering the lessons learned from first chapter which was considered when formalizing the functional requirements and non-functional requirements for adapting a data access layer (DAL) within an application architecture model to enable Cloud data access. This includes a conceptual overview, heterogeneous data access, system extension, secure data access and a use-use case analysis.

Chapter 6 - Design: This chapter illustrates an architectural overview and class diagram including technological explanations for fulfilling the described system requirements, where the architectural design organizes necessary components and their relations. A Java GUI application (i.e. SQL Evaluation tool) is described, which represents the general activities of the adaptation of data access layer and the evolutionary result of SQL statement adaptations.

Chapter 7 - Implementation: A detailed implementation of a Java software application (i.e. SQL Evaluation tool) including code listing and most significant classes are described in this chapter, which clarifies how the system meet the necessary requirements. Also, it explains about important technologies that are required to implement a complete software system. Finally, a software tool manual is described in this chapter.

Chapter 8 - Outcome and Future Work: The last chapter concludes this thesis by describing the main contributions of this thesis and suggests future work proposals.

1.5. Conventions

This section illustrates a list of abbreviations which are used throughout the thesis.

List of Abbreviations

The following list describes the significance of various abbreviations and acronyms used in this document. Full names by convention not valid or not used any more are marked as deprecated.

ACID	Atomicity, Consistency, Isolation and Durability
ACL	Access Control List
AMI	Amazon Machine Image
API	Application Programming Interface

ASP	Application Service Provider
AWS	Amazon Web Services
BLL	Business Logic Layer
CCI	Content Centric Interface *(deprecated)*
CFI	Cloud Federation and Interoperability *(deprecated)*
CRM	Customer Relationship Management
DAL	Data Access Layer
DB	Database
DBL	Database Layer
DBMS	Database Management System
DBaaS	Database-as-a-Service
DDL	Data Definition Language
DML	Data Manipulation Language
DNS	Domain Name System
DNSaaS	DNS-as-a-Service
DOL	Data Operation Layer
DOM	Document Object Model
EBS	Elastic Block Store
EC2	Elastic Compute Cloud
ESB	Enterprise Service Bus
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IaaS	Infrastructure-as-a-Service
IP	Internet Protocol
ISO	International Organization for Standardization *(deprecated)*
IT	Information Technology *(deprecated)*
JAR	Java Archive

JDBC	Java Database Connectivity
JDK	Java Development Kit
JDT	Java Development Tools
JRE	Java Runtime Environment
JSch	Java Secure Channel
LBaaS	Load Balancers-as-a-Service
MDA	Model Driven Architecture
MIL	Management Interface Layer
MVC	Model View Controller
NIST	National Institute of Standards and Technology
NoSQL	Not only SQL *(deprecated)*
OMG's	Object Management Group
OVF	Open Virtualization Format
PL	Presentation Layer
PaaS	Platform-as-a-Service
RDMS	Relational Database Management Systems
RDP	Remote Desktop Protocol
RDS	Relational Database Service
REST	Representational State Transfer
S3	Simple Storage Service
SAI	Secure Access Interface *(deprecated)*
SAX	Simple API for XML
SBC	Smart Business Desktop Cloud
SDB	SimpleDB
SFA	Sales Force Automation
SOA	Service-Oriented Architectures
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SaaS	Software-as-a-Service

TCP	Transmission Control Protocol
UI	User Interface
URL	Uniform Resource Locator
VFS	Virtual File System
VPC	Virtual Private Cloud
VPN	Virtual Private Network
XML	Extensible Markup Language

2. Fundamentals

This chapter explains the common approaches in enabling data access layer in a traditional application model and specifically focuses on technologies that are used for accessing data store services in the Cloud. It defines and describes the hands-on knowledge for understanding the concepts and principals of this work. Moreover, it represents an overview of basic terms of technologies that are used in this thesis and all together they appearance a background for the results of this thesis.

The first section defines the basic terms of Cloud computing. The goal of this work is to adapt data access layer to access enabling heterogeneous Cloud storage services, which will be a part of Cloud computing in the term of Platform-as-a-Service (PaaS). It also explains how the data access layer facilitates implementing SQL taxonomy based SQL adaptations. The second section gives the reader a basic impression about example Cloud services where database layer can be implemented to evaluate the adaptation data access layer. The detail evaluation procedures of SQL supports of Cloud services are demonstrated in Chapter 4. Moreover, the next section is dedicated to present the SQL-standard rules (SQL taxonomy). This thesis is a continuation of research outcomes of the authors Strauch et al. [SKLU11] where they introduce a taxonomy for Cloud data hosting solutions, the summery results of their research work are given in Section 2.1.2. This chapter introduces the available approaches and the purposes of adapting data access layer by comparing against the previous works for the further realization. Finally, this chapter describes the technologies are widely used in this work.

2.1. Cloud Computing

While Internet technology has become highly available with significant speed of computation capacity, the World Wide Web has introduced a widely spread innovative distribution channel for application software. Over the last decade an increasing number of software retailers have started marketing the computing services on a rental basis through online, which can be recognized as Application Service Provider (ASP). An ASP brings a significant development in networking infrastructure and computing technologies, and thus made it possible reducing capital expenditure of an enterprise IT, distribution costs and a consistent user base. It also reduces the installation and maintenance efforts of large scale application software to the customers [Tao01].

Cloud computing is an innovative paradigm, which provides the facility of using various application software services and configurable common computing resources. The computing resources are for example storage, networks, servers, services and so on. These computing resources can be used on-demand with minimum management effort and low cost pay-per-consume basis. According to the definition of Cloud computing of National Institute of Standards and Technology (NIST), there are five essential characteristics of the Cloud model [MG11]. The essential characteristics are On-demand self-service, Broad network access, Resource pooling, Rapid elasticity and Measured service. On-demand self-service allows that a consumer can provision computing resources automatically as needed without additional involvement of the service provider. Broad network access denotes that the computing resources are reachable over the network infrastructure and accessible by different types of client platforms through standard protocols. Moreover, the resources are pooled serving multiple tenants according to the customer demand, where resources are automatically scaling elastic on growing demand. Measured service allows that the service provider can automatically control and optimize resource usage to monitor computing capabilities and billing customers. In Cloud computing, there are three service models with respect to the computing capabilities provided to the Cloud service consumer. The definition of service models are as follows [MG11]:

- **Software-as-a-Service (SaaS)** is a service model provided by Cloud service provider where customer can employ and run applications on the Cloud infrastructure. The customer can have access the applications from a verity of client devices, such as Web browser. The underlying Cloud infrastructures are automatically managed or control by the service system and enables Cloud characteristics such as elasticity and accessibility. In this model the customer has only the possibility to control individual configuration settings for a user-specific application.
- **Platform-as-a-Service PaaS** is s service model where customers are allowed to deploy and configure their own created or acquired applications onto the Cloud infrastructure, for example software applications, services, or libraries and so on. The acquired applications are defined using programming language or tools supported by the service provider. In this model the customer has control over the deployed applications and the configuration settings, but not the underlying Cloud infrastructure.
- **Infrastructure-as-a-Service (IaaS)** is a service model where the customers are allowed to deploy and run random software. In this model the customer has control on operating systems, storage, deployed applications, virtual machines and select networking components. Moreover, this model allows customer deploying applications that runs on the existing operating systems.

To deploy an application in the Cloud four deployment models can be distinguished: Private Cloud, Public Cloud, Community Cloud and Hybrid Cloud. A private Cloud infrastructure is exclusively used by a single organization and can be included multiple business units. A community Cloud infrastructure is shared by a specific community, this mean that a multiple organizations can be used the Cloud infrastructure together. Both deployment models may exist on-premises or off-premises. A public Cloud infrastructure is openly used by the general public, which means it is not restricted to a single or group of organizations. This Cloud infrastructure is off-premises of the Cloud consumer. A hybrid Cloud infrastructure is a combination of two or more different Cloud infrastructures, which can be combined with private, public and community Cloud infrastructure. In this model, the Cloud infrastructures are staying unique entities and allowing data and application portability [MG11].

As part of the Cloud computing, Cloud Database-as-a-Service (DBaaS) has been introduced where customer can access database on-demand via the Internet from service provider. Currently, there are both relational and non-relational database services supported by database service providers. In Cloud computing, a database can be a traditional database such as MySQL, Oracle, PostgreSQL, Microsoft SQL Server, NoSQL, etc. (see Sect. 2.2) and can guarantee scalability, high availability and stability [Qui12]. Therefore, application can benefit using Cloud database service in order to maintain a big amount of IT enterprise data as well as big data.

2.1.1. Cloud Deployment Models and Application Layers

Usually traditional application is designed using a three layer architecture model and the layer are recognized as a presentation layer, business layer and data layer [DEF+08]. However, this thesis focusing on Data Access Layer (DAL), which is responsible for encapsulating the data access functionality and places between business layer and data layer [SKLU11]. This means that the application accesses database through DAL.

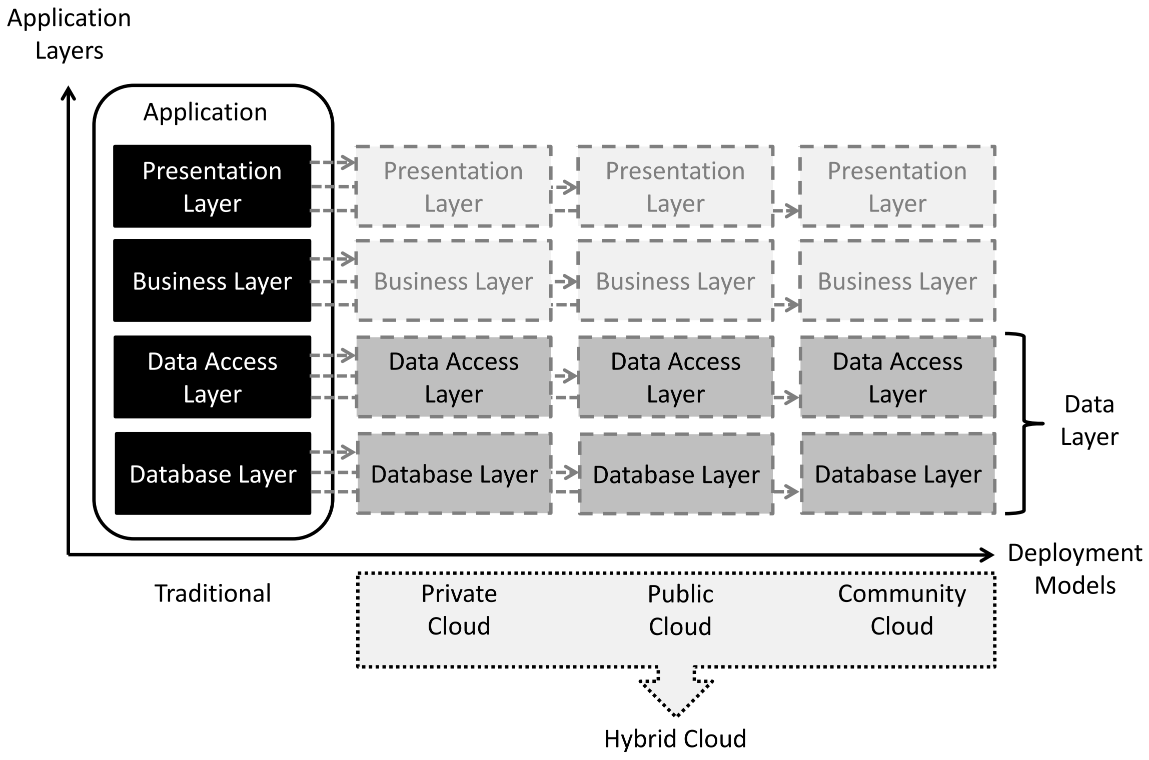


Figure 2.1.: Overview of Cloud Deployment Models and Application Layers [SBK+12].

Designing an application with an additional layer do not have change effect on the responsibilities of other layers, they are performing the same as traditional application layers. As

Cloud infrastructures allow accessing data store service within the Cloud like a traditional database server, the application has the possibility to move a traditional database to the Cloud. By accessing data store service a traditional application can take advantages of using highly available and scalable database service. However, adapting DAL in application architecture model, now an application can access various data store services which brings a great compensation in order to achieve the data computation, as different types of data computation can be performed in a suitable, high performance data store service.

Figure 2.1 shows an overview of distribution of four layer application architectures where DAL is a new layer and placed between business layer and database layer. The traditional application accessing a traditional database is shown on the left side of this figure. Depending on required data computations for an application system the database layer moved to the any of one Cloud deployment models. Moving database layer to the Cloud current concern is how traditional application can access data and perform data computation in the Cloud. Strauch et al. have presented a taxonomy supporting decision making for data hosting, migrating or building a database layer in the Cloud [SKLU11].

2.1.2. Cloud Data Hosting Solution

While Cloud computing becoming more and more popular these days, it is necessary ensuring Cloud data accessibility and availability for and enterprise application, because in Cloud computing different deployment models have unique approaches for Cloud data hosting. Strauch et al. [SKLU11] investigates and categorized possible solutions in a taxonomy for Cloud data hosting considering the scope of pure and hybrid Cloud. In order to organized the solutions in taxonomy there are six properties consisting of application layer, deployment model, location, service model, data store type and compatibility has been considered. This taxonomy can help the developer deciding a suitable approach, how to build or migrate a database layer in the Cloud. This reduces a lot of efforts for an application engineer.

Depending on used deployment models the pure (private, public and community) Cloud have in total 60 Cloud data hosting solutions. However, with a straightforward calculation there are 72 possibilities can be found including off-premise hosted for public Cloud. These pre defined off-premise hosted properties in public Cloud are invalid and have to be eliminated in the final calculation of Cloud data hosting solutions. According to the principal of hybrid Cloud the Cloud data hosting solutions are infinite, as hybrid Cloud infrastructure allows random combinations or two or more different pure Cloud. In this taxonomy the classifications for Cloud data hosting are distinguished using simple regular expression and the expressions are preparing suing abbreviation of the properties. To present the explicit possibilities in taxonomy the six properties have been discovered more importantly for classifying the solutions because hybrid Cloud present an infinite solutions [SKLU11].

2.2. Cloud Services

This section introduces a number of Cloud services where data store services have to be tested for the SQL evaluation. It is studied that there are several databases supported on a single Cloud infrastructure. For example, Amazon RDS supports MySQL, Oracle, Microsoft SQL server. Therefore, this section will give an overview about access enabling DBaaS for individual Cloud services.

2.2.1. Amazon EC2

Amazon Elastic Compute Cloud (EC2) is a Web service, which is designed to build Web-scale computing simpler and provides elastic compute facility in the Cloud. It is a true virtual computing environment. It allows developer to launch instances with different types of operating systems and customize the application environment. It supports several database services, likely MySQL, Oracle and PostgreSQL database services. With Amazon EC2 instances application can benefit mounting on-demand volume of storage, i.e. block level storage volume Amazon Elastic Block Store (EBS), Amazon Simple Storage Service (S3), etc [EC2]. Now, application can take advantages of maintain a big amount of IT enterprise data on Amazon EBS and easily archive the database on Amazon S3. With the concept of Cloud computing, the data store services as well as mounted volumes on Cloud are scaling automatically. In addition, it allows mounting multiple storage volumes in an instance, if require [EBS]. Therefore, using these facilities IT industries are no more worries about maintain a high amount of enterprise data, while they are moving to the Cloud.

2.2.2. Amazon RDS

Amazon Relational Database Service (RDS) is a Web service, which is especially designed to manage the relational database in the Cloud and provides facility of accessing on-demand storage capacity. It allows user to launch several types of relational database instances, likely MySQL, Oracle, Microsoft SQL Server database services. According to the definition of Amazon RDS, a traditional application can access a relational database with full functionalities. In addition, Amazon RDS provides the facility of automatic scaling the compute resources, easy storing backups and replication to enhance availability and reliability for production databases [RDS]. Therefore, application can take advantages of maintain a big amount of IT enterprise data on Amazon RDS Database (DB) instances and easily archive the database. Using these facilities IT industries are no more worrying about maintain a high amount of enterprise data, while they are moving to the Cloud. Purpose of this SQL statements testing on Amazon RDS is to ensure the facility of enable accessing MySQL database service.

2.2.3. Amazon VPC

Amazon Virtual Private Cloud (VPC) is a scalable private, isolated virtual networking environment like a traditional network infrastructure. It allows user a full control over virtual networking environment. For example, user can assign a range of Internet Protocol (IP) address, create subnets, and configure routing tables and network gateways, which ensure a complete control of networking infrastructure. The infrastructure can be divided into one or more public or private subnets with Amazon VPC private IP address range. Each subnet can be attached with one or more Amazon EC2 or Amazon RDS DB instances. On Amazon VPC, it is possible to build a public-facing subnet, which can have Internet access, and have access to the backend systems in a private-facing subnet where private-facing subnet has no Internet access (e.g. application servers or database systems). However, to access an Amazon VPC instance directly from the Internet, it is necessary to attach an Elastic IP address to the instance [VPC].

2.2.4. Amazon SimpleDB

Amazon SimpleDB is a Web service providing highly available, simple, scalable and flexible non-relational data store service in the Cloud. Using SimpleDB developers can reduce their work load of database administration, as they will only store and query data items via Web services requests and SimpleDB automatically index data, update schemas in case added a new data, scale-out of domains, and so on. This mean that SimpleDB automatically manage the database services, a structured data storage service in the Cloud to make Web-scale computing easier. Moreover, SimpleDB is designed for running queries on structured data in real time and store multiple data sets in an attributes [SDB].

Amazon SimpleDB is a NoSQL based data storage service, which introduces indexed (key) base structured data storage, the keys must be unique. It is similar to spreadsheet and structured with domains, items, attributes and values. Comparing with a relational database model, Amazon SimpleDB Domain is similar to a Table where Attributes represented by Columns and Items represented Rows and values are represented by cells. According to the concept of SimpleDB data structure, a domain can have zero or more attributes and an attribute can have multiple values [Ama09]. In Amazon SimpleDB there is no relation between the domains, as it is freely structured. It is possible to access only a domain while query to a database. Also it has no data types because query returns only Text (String) type of data and the data are originally stored in Unicode character UTF-8 format.

2.3. SQL Taxonomy

SQL is a programming language designed for defining and maintaining relational database systems, which is originally based upon relational algebra and tuple relational calculus. Its scope includes data insert, query, update and delete, schema creation and modification, and data access control [Dar05]. This section describes the basic SQL rules for managing data in RDMS and illustrates the SQL test cases based on basic SQL queries and statements.

2.3.1. General SQL Rules

The SQL language is divided into several elements, including queries, statements, clauses, expressions, predicates, insignificant whitespace, etc. [Ame03]. It is investigated that queries and statements are the most important elements in database management systems. The clauses, expressions and predicates are often using for the completeness of a query and statement operation and they bring different working roles. For example, the clauses are considering constituent components of queries and statements, where the expressions can produce tabular data or scalar values which comprises of columns & rows and the predicates are specifying the conditions.

SQL query is the most important element and common operation in SQL language. It is used to retrieve data from the database based on specific conditions. The query operation is performed with the declarative SELECT statement, which fetches data from one or more tables, or expressions in a database scheme. In a normal use of SELECT statements it does not have persistent effects on the database, but exceptions could be found in some special implementations. For example, in some databases the SELECT INTO syntax is existed and which can have persistent effects [INT]. The queries permit the user to describe expected data and leaving the Database Management System (DBMS) accountable for planning, optimizing, and performing the physical operation needed to produce the desired result.

In SQL language, the SQL statements are designed for controlling transactions, program flow, connections, sessions, or diagnostics and it may has a persistent effect on schemata and data manipulation. The Data Manipulation Language (DML) [RG02] is used to retrieve and manipulate the data with SQL statements in a relational database. It is the subset of SQL used to add (INSERT), update (UPDATE) and delete (DELETE) data. Table and index structure in a relational database can be managed by Data Definition Language (DDL) [RG02]. The DDL introduces with most frequently used SQL items, for example, CREATE, ALTER, RENAME, DROP and TRUNCATE, etc. statements.

Table 2.1.: Overview of SQL statements based on SQL:2003 standard.

SQL SELECT STATEMENTS	SQL JOIN STATEMENTS
SELECT * FROM tbl Select all rows and columns from table tbl	**SELECT * FROM tbl1** **INNER JOIN tbl2 ON** ***join-conditions*** Inner join table tbl1 with tbl2 based on join-conditions

SELECT c1,c2 FROM tbl
Select column c1, c2 and all rows from table tbl
	SELECT * FROM tbl1
	INNER JOIN tbl2 ON ***join-conditions***
	WHERE ***conditions***
	Inner join table tbl1 with tbl2 based on join-conditions with specific criterion

SELECT c1,c2 FROM tbl
WHERE ***conditions***
Select columns c1, c2 from table tbl with specific criterion
	SELECT * FROM tbl1, tbl2
	WHERE ***join-conditions***
	Select all values from table tbl1 and tbl2 based on join-conditions

SELECT c1,c2 FROM tbl
WHERE ***conditions***
ORDER BY c1 ASC, c2 DESC
Select columns c1, c2 with specific criterion and from table tbl order result by column c1 in ascending order and c2 in descending order
	SELECT * FROM tbl1
	LEFT JOIN tbl2 ON ***join-conditions***
	Left join table tbl1 with tbl2 based on join-conditions

SELECT DISTINCT c1,c2
FROM tbl
Select distinct (different) rows by columns c1 and c2 from table tbl.
	SELECT * FROM tbl1
	RIGHT JOIN tbl2 ON ***join-conditions***
	Right join table tbl1 with tbl2 based on join-conditions

SELECT c1, aggregate (expr)
FROM tbl
GROUP BY c1
Select column c1 and use aggregate function on expression expr, group columns by column c1
	SELECT * FROM tbl1
	FULL OUTER JOIN tbl2 ON ***join-conditions***
	Full outer join table tbl1 with tbl2 based on join-conditions

SELECT c1, aggregate(expr) AS c2
FROM tbl
GROUP BY c1
HAVING c2 > v1
Select column c1 and c2 as column alias of the result of aggregate function on expr. Filter group of records with c2 greater than value v1
	Example:
	SELECT * FROM tbl1 AS at1
	INNER JOIN tbl2 AS at1 ON at1.c1 = at2.c2

SQL OPERATORS	SQL UPDATE DATABASE

SELECT * FROM tbl
WHERE c1 [NOT] BETWEEN v1 AND v2
Select all rows from table tbl where the value of column c1 is not in the inclusive range between v1 and v2
	INSERT INTO tbl (c1, c2, . . .)
	VALUES (v1, v2, . . .)
	Insert data into table tbl

SELECT * FROM tbl
WHERE c1 [NOT] IN (v1,v2,. . .)
IN: to specify multiple possible values for a column
	INSERT INTO tbl1 (c1, c2)
	SELECT c1, c2 FROM tbl2
	WHERE ***conditions***
	Insert data from tbl2 into tbl1

SELECT * FROM tbl
WHERE c1 > v1 AND c1 < v2
Select all rows from table tbl where the value of column c1 is greater than v1 and less than v2
	UPDATE tbl
	SET c1 = v1, C2 = v2, . . .
	WHERE ***conditions***
	Update data in table tbl

SELECT * FROM tbl
WHERE c1 < v1 OR c1 > v2
Select all rows from table tbl where the value of column c1 is either less than v1 or greater than v2
	DELETE FROM tbl
	WHERE ***conditions***
	Delete records from table tbl based on specific criterion

SELECT * FROM tbl
WHERE c1 = v1
Select all rows from table tbl where the value of column c1 is equal to v1
	TRUNCATE TABLE tbl
	Drop table tbl and re-create it, all data is lost

Continued on next page

Table 2.1:Overview of SQL statements based on SQL:2003 standard – (continued from previous page)

SELECT * FROM tbl
WHERE c1 <> v1
Select all rows from table tbl where the value of column c1 is not equal v1	

SQL TABLE STATEMENTS	SQL VIEW STATEMENTS

CREATE TABLE tbl (
c1 datatype(length),
c2 datatype(length),
. . .
PRIMARY KEY (c1)
)
Create table tbl with primary key is c1
	CREATE VIEW vw AS
	SELECT c1, c2
	FROM tbl
	The view vw lists all active columns (c1 and c2) from the tbl table.

DROP TABLE tbl
Remove table tbl from database
	ALTER VIEW vw AS
	SELECT c1, c2
	FROM tbl
	Update the view vw lists all active columns (c1 and c2) in the tbl table

ALTER TABLE tbl **ADD COLUMN c1 datatype(length)** Add column c1 to table tbl	**DROP VIEW vw** Remove view vw from the database
ALTER TABLE tbl **DROP COLUMN c1** Drop column c1 from table tbl	

SQL INDEX STATEMENTS	Abbreviations
CREATE UNIQUE **INDEX idx ON tbl (c1, c2, . . .)** Creates a unique index on tbl table. Duplicate values are not allowed.	*tbl: table name; c: column name; vw: view name; idx: index name; v: value; at: table alias; aggregate: aggregate function*
DROP INDEX tbl.idx Drop an index idx from table tbl	

The Table 2.1 represents an overview of SQL statements based on SQL:2003 standard [Dar05] which covers the existing SQL rules in a relational or a nor-relational database system. The initial design of the taxonomy is based on MySQL relational database, as most of the Cloud services supports MySQL database these days. However, the taxonomy also has to be tested on non-relational database, for example, Amazon SimpleDB, Amazon DynamoDB, these are fully managed NoSQL database services. In the taxonomy, it is considered to cover the basic and frequently used SQL statements, which include SQL select, operators, update database, table, join, index and view statements. It is investigated that there are several SQL cheat sheets implemented basic SQL taxonomy can be found online, but no one can be found covering 100 percent taxonomy. Therefore, in order to achieve maximum SQL taxonomy covering we have followed the SQL tutorial from w3schools [W3S] as well as a book "Database Management Systems" [RG02], those are recognized as SQL-standard.

2.4. openCRX

In order to evaluate the adaptation of DAL for enable accessing Cloud data store service, we consider using openCRX application. The openCRX is a popular enterprise-class open source CRM solution, which is based on openMDX, an open source MDA framework software development based on the Object Management Group (OMG's) Model Driven Architecture MDA modeling standards [CRI12].

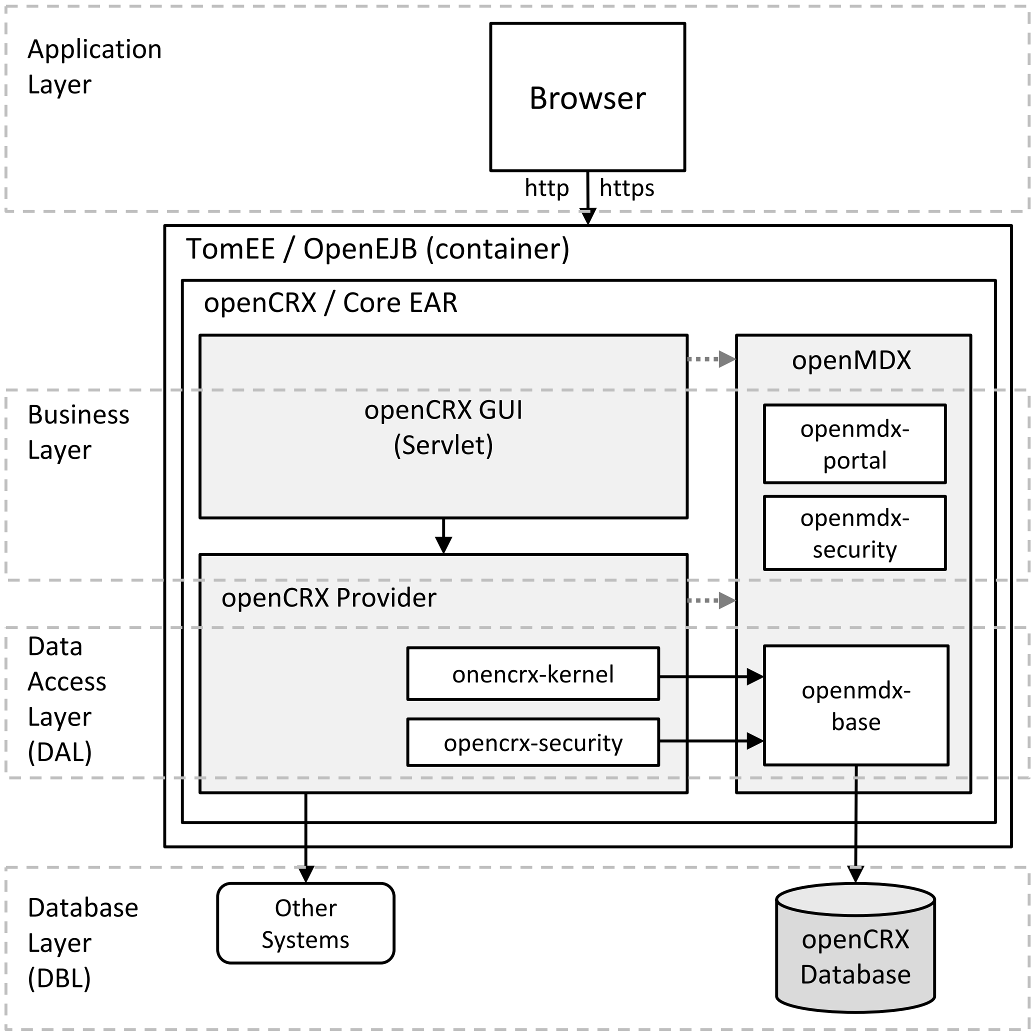


Figure 2.2.: openCRX application architecture model based on application layers.

Figure 2.2 shows an application architecture model of openCRX application where four layers, such as application layer, business layer, data access layer and database layer are clearly defined. openCRX is a Platform-independent, as it runs on any platform with Apache Tomcat or any J2EE-compliant AppServer and also it is a platform-neutral, as it can support with one of the major database management systems (e.g. HSQLDB, MySQL, PostgreSQL, MS SQL, Oracle, or IBM DB2). Nevertheless, it is important to consider the compatible versions of each technology it is designed for. The initial plan was to adapt the DAL of openCRX, but then we decided to use the database and the data only to reduce the effort in order to focus on the core contributions of the thesis. As a result the openCRX application is not any more part of a core contribution of this thesis. However, we decide using the database and data of openCRX instead of adapting its DAL for each scenario.

The reason is to use database and data of openCRX instead of adapting its DAL, there are various dependencies and lot of effort has to be done to deploy openCRX application. For example, it is important to set-up the system environment before running the application. To set-up the environment, it is required to install ant 1.8.2 and Java 6. However, it is not sufficient to have a Java Runtime Environment (JRE) only, the full-blown Java Development Kit (JDK) is essential to run openCRX. Moreover, it is depending on the version of openMDX, as openMDX is an underlying platform of openCRX application. It is a component based software system, the different functionalities and features are designed with individual projects and packages. Therefore, it meant recognizing the necessary packages for DAL implemented in openCRX can be easier. But, it is studied that the most of the DAL packages are implemented in openMDX, which means the openCRX is accessing database through openMDX. Then it becomes a high effort for adapting the DAL of openCRX. As we have planned to use openCRX database and data for evaluation of SQL support of Cloud data service, it is necessary investigating openCRX is running on MySQL database without any problem because our default test setting is designed based on MySQL database.

3. Related Works

This chapter represents the work of other authors who have designed data access layer in a multi-layered traditional application setting in order to enable accessing data store service in the Cloud. The VISION Cloud has developed a concept and describes how next generation software approaches can benefit implementing Data Access Layer (DAL) where access layer enables transparent and unified access to data in the Cloud storage. jclouds offers an open source library Application Programming Interface (API), a portable API which can be used to access Cloud resources with cloud-specific features. Apache Deltacloud offers a unified RESTful-based API which can be used to manage a rising number of Cloud provides and controlling the individual Cloud infrastructure. Apache Libcloud provides a client library, which can be used for interacting with multiple Cloud service providers. Currently, there are about 26 different Cloud providers are supported by Libcloud library. This analysis of existing approaches towards data access layer shows, how existing concepts can be reused for adapting data access layer to enable accessing Cloud data storage.

3.1. VISION Cloud

VISION Cloud is an EU-funded project, which has introduced a potential ICT infrastructure for consistent delivery of data-intensive storage services. Using this infrastructure the developers can take advantages establishing secure and quality data storage services on demand with an optimum and tolerable investment basis, instead of creating a new physical datacenter which is cost effective [Lor10]. In this project various aspects of Cloud computing have been considered, such as virtualization techniques, Cloud storage, secure data access, Cloud federation and interoperability, and so on. However, the main focus of this project is Cloud data storage services. To ensure secure data access in the Cloud, this project introduces a new layered based architecture model for next generation software application approaches. It includes two new layer concepts in the application architecture: Data Access Layer (DAL) and Data Operation Layer (DOL). In the application setting, DAL is responsible for establishing the communication between the application and the storage by providing necessary functionalities. Precisely, it can be determined that this layer is allocating as the transaction layer and placed between application layer and data storage layer. The DOL is an underlying layer in the application setting which mainly responsible for implementing the storage functionality [AAB+11].

With designing access layer, VISION Cloud introduces a significant potential activity that Cloud data can be exchanged among different Cloud data services. This means that access layer allows transparent and incorporate data access within Cloud data centers. This approach brings a lot of advantages to the Cloud application and can reduce application complication. Additionally, in the design of access layer, it also offers security mechanisms for single and community Cloud services. By using these mechanisms, it makes possible determining the implementation of access model and enabling secure data access in the Cloud. In this scope, DAL makes available the core interface and using this interface application can store and retrieve data object from the Cloud store services. Moreover, DAL also introduces content-based access to the Cloud storage, where DAL is representing an abstraction of Data storages.

In VISION Cloud, there are several important functionalities can be performed by DAL components, where the components are: Content centric interface, Cloud federation and interoperability, Secure access and Computational storage. For example, it makes feasible to achieve Cloud-grade distributed strategy enforcement to all process and access in the Cloud through the security and access control, directly execute exhaustive computational task in the Cloud storage by computational storage and also directly access to the content using content centric interface component through interfacing with application. By using this approach, the application are no more required storing and managing Uniform Resource Locator (URL)s or use metadata for finding contents [AAB$^+$11].

3.1.1. VISION Cloud Architecture

In vision Cloud application architecture, the DAL is considering a logical entry position to the system, where DAL is ensuring a set of functionalities to the applications including access to the data objects, secure data access, unified access to a federation of data store services and allows changing object of data storage, a storlet [Vil12] in the Cloud. A storlet is a computational mediator which has role to classify the computation and isolating metadata automatically by performing the triggering conditions. In this scope, data storage modification means create, delete or manage storlets using a user API. According to the software layer architecture in VISION Cloud, the DAL is representing a communication media or translator between application and the storage, where DAL is implementing data access functionalities based on the DOL. The DOL is the lower layer in VISION Cloud application architecture, which implements the core functionalities providing data objects in the application system in order to access Cloud data storage. As a result applications or users have the opportunity to use a common access layer for enabling Cloud data access.

Figure 3.1 shows an abstraction level conceptual architecture of Data Access Layer based on VISION Cloud software context, where applications are communicating to the DAL through a Hypertext Transfer Protocol (HTTP) server request processing and then the DAL is communicating to the DOL, the storage of data objects. In this design concept, it is considered that DAL has implemented with a set of components (i.e. Content Centric Interface (CCI) , Cloud Federation and Interoperability (CFI), Secure Access Interface (SAI) and Computational Storage) and the functionalities depending on DOL, then the basic functionalities including storage operations, metadata management, replication, etc. are performed by DOL. Within DAL, the CCI has the responsibility to keep track the content information, where the CFI has ensured the service quality, both approaches are using metadata to perform the allocated

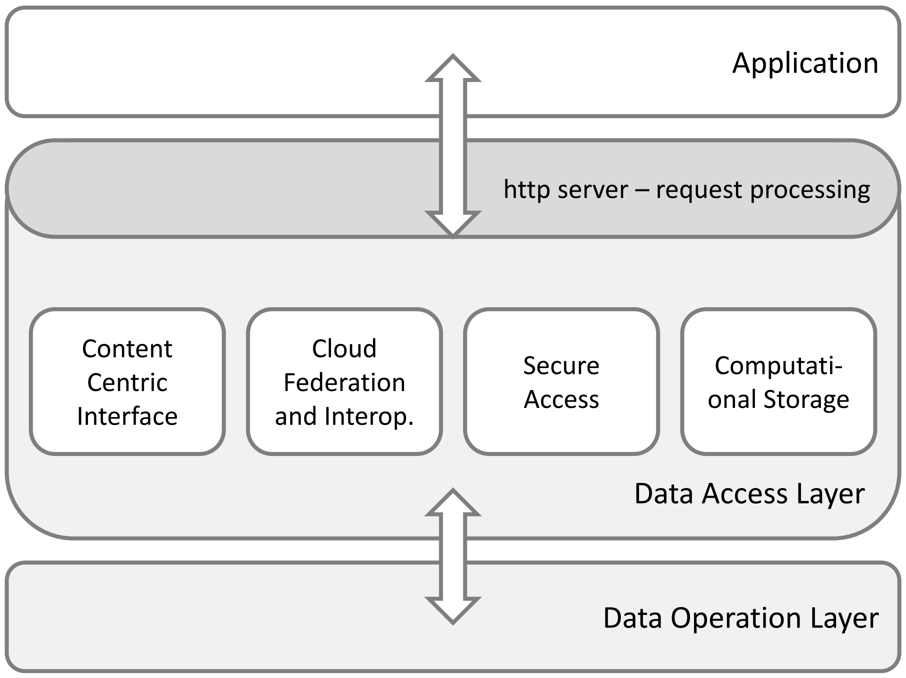

Application
http server – request processing
Content Centric Interface
Cloud Federation and Interop.
Secure Access
Computati-onal Storage
Data Access Layer
Data Operation Layer

Figure 3.1.: High Level Conceptual Architecture of the Data Access Layer [AAB+11].

functionalities. Moreover, the SAI is dealing with authentication procedures in the system where the applications are using Computational Storage component for interacting with DOL objects. However, when application communicates to the DAL through a process request, the Management Interface Layer (MIL) also establish a communication tunnel to the application in order to achieve the management functionalities of VISION Cloud. Nevertheless, VISION Cloud implements a layer based application architecture model, which is common and standard approach for a software systems since a few last decades. Designing layering based application is advantageous because in this approach the applications can be implemented with separating different segments and functionalities in a logical blocks or components in software system where components are often reusable [AAB+11].

Technical Architecture

As part of the project work VISION Cloud has presented a detail technical architecture for the entire system on respected to the access layer shown in Figure 3.2. This architecture describes how DAL implements interaction between application and the data storage service. Also, it summarizes the important necessary relations with different components in the system. Additionally, it show a detail functional implementation of DAL, means how DAL performs data access functionalities within the VISION Cloud system. The applications are communicating to the DAL using Representational State Transfer (REST) method request and the authentications are required when calling a functional component. In this master's thesis we are not currently considering secure data access or data replication, however our goal is to adapt the DAL in order to enable Cloud data access. This is relatively similar approach but we have focused on a concrete design how traditional application can benefit access a scaled and stabled data store service in the Cloud to reduce the application complexity instead of using traditional database.

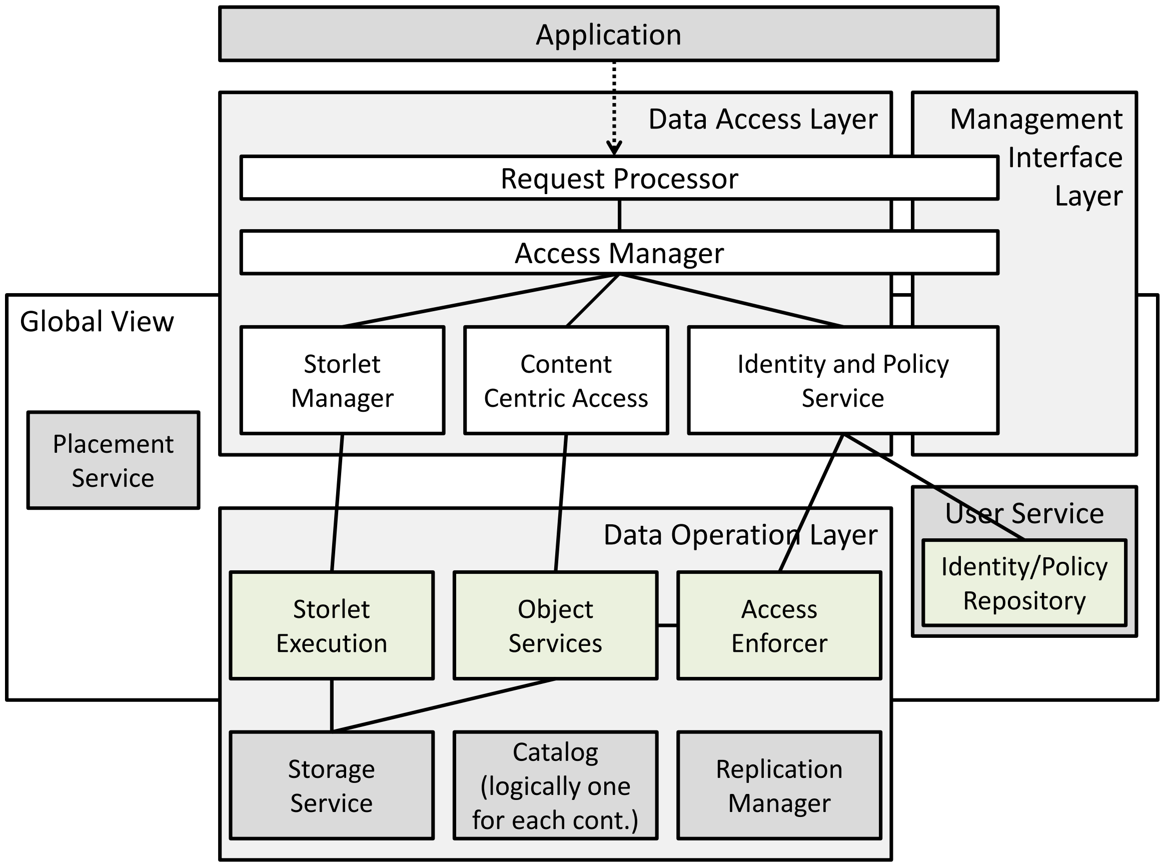


Figure 3.2.: High Level Technical Architecture of the Data Access Layer [AAB+11].

Request processor is the only entry position to the application systems communicating to the DAL, where it processes requests form the application and implements basic functionalities including systems performance and scalability features for managing proper distribution to the system. It forwards a request the Access Manager, a access control component to control the systems implementing authentication and authorization. It also handles request by request handler for the MIL to achieve the management functionalities, where MIL is a one of the part of VISION Cloud system besides DAL. However, during processing a request the request handler characterized the request separately for both DAL and MIL. Additionally, in case errors during processing request, it has the responsibility to notify to the application about occurring errors. In the technical point of view there are three components to be considered as main in DAL, such as Content Centric Service, Computational Storage and Secure Access.

The Content Centric Service is an important component in DAL which is responsible for accomplishing content-based access and interacting to the data objects service with necessary metadata request, where the metadata are maintaining by DOL. According to the functional point of view, it includes a number of modules where each modules are consisting of several self-contained functions and performs basic object functions. For example, individual module implements the basic functionality of rich metadata handling, putting and getting objects and a matrices service and so on.

The Computational Storage is responsible to establishing an entry point in order to perform the management of starlets by the application (or the user). Moreover, using functionality of computational storage it makes possible the creation of starlets. The fundamental idea of this component is that the user will define the necessary starlet functionalities and then the starlet will automatically adapted with required configuration and the execution environment of the DOL. The functioning principal of this component is significant a to our research work and can be reused some of the concepts in this master's thesis.

The Secure Access is responsible for ensuring feasible access control by implementing various authentication and authorization to the system in order to restrict the computation as well as Cloud data access. To achieve secure computation and data access this component has included there sub component such as Access Manager, Access Enforcer, and Identity Policy Services. The authentication and access control requests processing are done by the Access Manager, where Access Control List (ACL) based access control can be performed by Access Enforcer component. Furthermore, the policy-based authorization and various authentications (i.e. user provisioning, credential management, etc.) can be performed by the Identity and Access Policy Services component [AAB$^+$11].

3.2. JClouds

JClouds is an open source library written in Java and built upon based on Clojure programming language. This library permits Cloud application developers to use Cloud-specific features. Also, it allows the developers to use portable notions. With the Cloud models, this library has implemented as PaaS where several applications can be plugged in this scope. Using jclouds library the customers benefit accessing multiple Cloud services at the same time and currently this library has supported by more than 30 Cloud providers and software stacks, such as Amazon, Azure, GoGrid, Ninefold, vCloud, OpenStack, etc. The approach is somewhat related to this master's thesis where we are only focusing on access enabling data store service in the Cloud with traditional application while traditional data has move to the Cloud. This means that our focus is enabling Cloud DBaaS for traditional application. In jclouds structure there are two packages introduced with jclouds library: ComputeService and BlobStore. ComputeService is responsible for managing and controlling machines (i.e. Instances) in the Cloud. For example, using ComputeService package it is possible to start several machines at the same time and configure them by installing necessary software. On the other hand BlobStore makes easy dealing with key-value providers like Amazon S3. Using BlobStore package is it possible to determine a straightforward map view of a container [jcl11b].

For managing nodes in the clouds, the Compute API performs as one of the portable means. The functionality of jclouds Compute API belongs to generate a basic abstraction among common Compute APIs (e.g. Amazon EC2, VMware vCloud etc.). It has special features make greater visibility which include a single connection for multi homes regions (e.g. resources of all regions of EC2). Multiple node sets can be running under this cloud API. This is easily controllable by Secure Shell (SSH) keys to the node on start up. Executing scripts can be running on the machines in a set and to handle error it offers some special exception types. By providing in memory blobstore to test provisioning instructions, it offers an infrastructure as code implies testability. With it updated features, it offers a map to persist credentials of nodes which can be used to keep track of all cloud nodes credential from single place [jcl11c].

The idea of BlobStore API contains three concepts which include service, container and blob. Container contains namespace for data and inside container one can store data as blob which should be referenced by a particular name. Like Microsoft Azure Blob Service and Amazon S3, BlobStore API provides key-value storage management which includes both synchronous and asynchronous APIs. Asynchronous API provides most efficient access gain where the access could be via threads or native asynchronous clients. This portable API supports map based data access. It also has some popular tools integrated (e.g. Apache commons Virtual File System (VFS)). The location API of BlobStore helps to portably identify location oriented context (e.g. America or Europe). By using BlobRequestSinger, HTTP requests can be portably generated and passed to the external system for execution or processing. BlobStore file system allows for testing storage code without any proper credential. The same API can be used for persistent disk, memory or a remote BlobStore (e.g. Amazon S3) [jcl11a].

Some important features are introduced while using jclouds library for connecting to the Cloud services, jclouds focuses on following areas: Simple interface, Runtime portability, Deal with Web complexity, Unit test ability, Performance, Location and Quality. While jclouds provides Simple interface for managing Cloud envirionments, the developer can reuse their ideas using common programming model without creating a new object type. So that it can reduce the workload of troubleshooting and dealing with Web services or APIs. Moreover, jclouds provides drivers for managing in restricted environment in the Cloud, such as Google App Engine, which shows that jclouds is runtime portable. This library automatically handled the transient failures and redirects. It also provides the facility of implementing automatic unit testing for Cloud endpoints through Stub connections. In order to speed up the computations, the tasks can be performed in parallel wherever straight Simple API for XML (SAX) for Extensible Markup Language (XML) can be used. Furthermore, using this library it is also possible to determine the location where computations or resources are runs in [jcl11b].

3.3. Deltacloud

Apache Dealtacloud is a unified RESTful-based API server which can be used to manage a growing number of popular Cloud provides by connecting with any Cloud platform and controlling the individual Cloud services through essential adapter called *driver*. While using

Deltacloud API, it is possible to connect different Cloud service providers simultaneously using compatible drivers on different ports. Currently, there are several drivers are available for the subsequent Cloud platforms: Condor, VMware vSphere, OpenNebula, Eucalyptus, RHEV-M, IBM Smart Business Desktop Cloud (SBC), GoGrid, Rackspace, RimuHosting, Terremark, Amazon EC2, OpenStack. Also it has sifted with a number of storage drivers, for example Amazon S3, Eucalyptus Walrus, Rackspace CloudFiles, Microsoft Azure, Google Storage.

Deltacloud API has introduced a number of entities in the back-end provider Cloud, unfortunately all features are not supported by every Cloud service providers. Some are supporting all and some are supporting a few of entities. For example, the Microsoft Azure driver presently supports only the 'Buckets' collection. However, the drivers are dynamically performed according to the Cloud infrastructure. The Deltacloud API entities are as follows: Realms, Instances, Images, Instance States, Keys, Storage_Volume, Storage_Snapshot, Bucket, Blob, Address, Load Balancer, Firewalls. Where Realms represents a datacenter of an organizational unit, Instances represents the status of a back-end Cloud (i.e. server Images), Storage Volume represents a virtual storage device attaching with an Instance, Storage Snapshot represents a snapshots of a Storage Volume in a particular point in time, Bucket represents a container for data blobs.

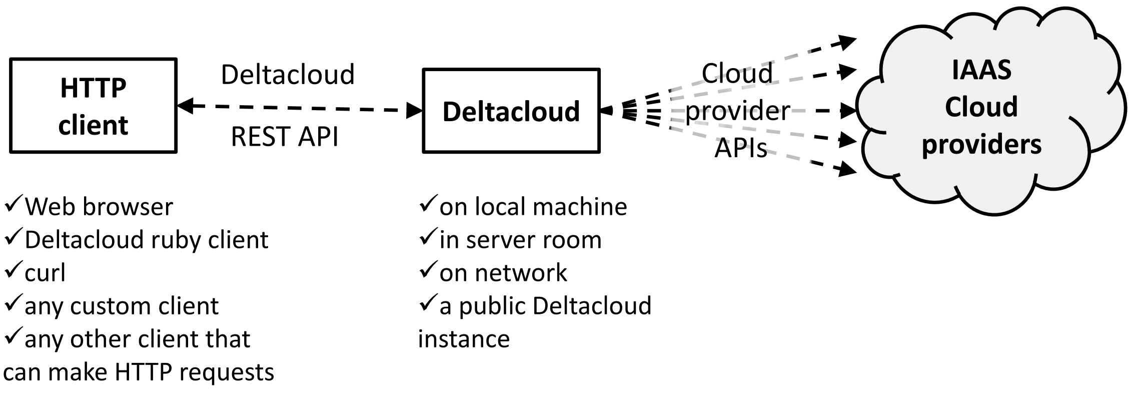


Figure 3.3.: Overview of conceptual working procedure within Deltacloud infrastructure where a HTTP client application accessing Cloud services through Deltacloud server using REST API request call [Apa11].

Figure 3.3 shows that HTTP client (i.e. Web browser) makes HTTP request using Deltacloud API to talk to the server and control the Cloud infrastructure, as HTTP client comes with a simple HyperText Markup Language (HTML) interface. The HTML interface is usually written in jQuery mobile framework. So that it make easier to connect using a thin client (e.g. mobile or tablet devices). Furthermore, with Deltacloud it maintains a ruby client which allowing application to interact with Deltacloud server programmatically controlling Cloud services. Using Deltacloud client (Ruby), the clients intends to insulate users in order to handle HTTP and REST directly. There are several ways to configure Deltacloud server, for example it can be configure on local machine, in server room, on a network or even on a public Deltacloud instance. The concept behind about working principal of Deltacloud, the

HTTP client requesting computation to the Deltacloud server through REST API call and Deltacloud server forwarding request to the specific Cloud providers APIs in order to perform the computation in the Cloud. The approach is somewhat related to our work. This means that in this scenario the Deltacloud server is a middleware between client (e.g. an application) and the Cloud providers. This also can be considered a data access layer comparing our application architecture models when applications are accessing Cloud data store service.

Within Deltacloud, while starting a number of Cloud service instances it provides a method to instruct the server dynamically route to the specific Cloud service by using specific drivers despite of the default driver that the server was invoked for the initial starting. The concept of provider sometimes can be supported by some of the drivers which make accessible the different instances of a Cloud using same driver, like accessing different regions of Amazon EC2 at the same time using a same driver. Additionally, installing the Deltacloud client automatically provides a deltacloud command line tool; this tool allows controlling Cloud infrastructure form the command line basis. Other command line tool distribution is cURL which is a Linux based HTTP client. It can also be used to speak to the Deltacloud server using Deltacloud REST API. This tool reveals the flexibility and power of the Deltacloud REST API. In order to access the Deltacloud API, it also provides a C/C++ library called Libdeltacloud. Manipulating Cloud objects through Deltacloud API this library exports a suitable structures and functions [Apa11].

3.4. Apache Libcloud

Apache Libcloud is a Python based library API for managing Cloud services. It provides a standardized API and single interface for connecting to multiple Cloud provides as well as various Cloud services like computing, storage and so on [Che12]. There are currently about 26 Cloud providers can be supported by using Libcloud [Apa]. Developers can benefit using this library in order to access multiple Cloud resources for different computations purpose. The goal of this library is to provide a functional cross-cloud environment. In the current version (i.e. Libcloud v0.10.1) of Libcloud, It has included multiple components consisting of Cloud Servers (Compute), Cloud Storage, Load Balancers-as-a-Service (LBaaS), DNS-as-a-Service (DNSaaS) for managing Cloud resources [AKM$^+$12].

Compute component make feasible to the user managing Cloud infrastructure and virtual instances accessible form available Cloud providers, for example Amazon EC2, Rackspace and others. Additionally, using this component developer can run the deployment scripts in order to manage and configure the newly created instances. Furthermore, this component represent the available configurations and status information of an Instance which includes hardware configuration, operating system, server location and the sever state. Storage component make possible to the developer managing Cloud storage and services, for example CloudFiles, Google Storage, Amazon S3, Amazon EBS, etc. This component has the inclusion of sub component s or functionalities as Object and Container, where object represent the BLOB and container has implemented with a concept that it can contain multiple objects.

Load Balancer component make reasonable to the developer managing the services and configuring the Load Balancer as a Service to several instances with the scope, such as GoGrid Load Balancers, Rackspace Cloud Load Balancers, etc. This component can represent the allocated members as well as load balancing algorithm with a load balancer instance. Domain Name System (DNS) component allows developer managing the services and representing the information about a Cloud domain. Using this component, it is also possible to manage and configure the DNSaaS. The example DNSaaS are Zerigo DNS, Rackspace Cloud DNS as so on. Each component is functioning moderately self-sustainable, as they are implemented based on component based architecture, and exposes a straightforward way to make use of base API with all supported Cloud providers. However, in some architectural views the components are depending on each others. For example, in order to add a member to the load balancer it reads the IP address from the Node object [Apa].

Apache Libcloud library has implemented drivers for specific cloud providers (i.e. Amazon EC2, IBM SmartCloud Enterprise, etc.). The drivers are designed like as extension methods besids base API where methods are implemented functionality connecting to a specific Cloud provider. Therefore, according the design concept of Libcloud it is possible to connect with multiple providers simultaneously using one library. The library is written using standard Python programming language, so that it is compatible with multiple versions of Python. Also, it brings advantages for the Cloud application to ease integration with various operating systems and virtualization tools because Python is a language which can be supported by various operating systems, such as Linux, Windows and others.

Apache Libcloud API is designed with a set of python classes and functions, which are to be used to enclose and control an Infrastructure as a Service of a Cloud. The classes are designed individually for specific Cloud providers so that it make feasible communicating a specific Cloud service consistently, while services are provided equivalent features. The functions are implemented based on core of the methods provided by different Cloud services that make capable the user to discover the information in order to prepare and connecting the virtual machines. Using Libcloud the developer can take advantages building applications that extending uses of multiple Clouds and can be migrated to different Clouds in order to balance the workload. However, Libcloud does not offer capabilities specifically managing storage volume or data store service in the Cloud, which we are defining in this master's thesis. In order to determine the compute size of an instance, initially it simplifies consideration that all available commute size is valid for every image. This means that the size is considered as dynamic, for example some instances are not supporting micro instance or some are not supporting large instance, the all conditions can be coped with this library. Furthermore, using Libcloud's SSH utilities or an open source tools, an Open Virtualization Format (OVF), it is possible connecting to nodes and transferring data to the Cloud instances [AKM$^+$12].

4. Evaluation of SQL Support of Cloud Data Services

This chapter describes a test setup, default scenario, and the scenarios of running SQL test cases on several selected Cloud data storages for the evaluation of SQL support of Cloud data services. The test setup section describes a detailed organization of the testing environment and the default test scenario section explains how the SQL taxonomy could be covered by possible SQL test cases. The default SQL test cases are designed based on MySQL database, as it is broadly accepted open source database among various database servers and there are several Cloud data storage providers supporting the MySQL database. Also, MySQL database is one of the world's most used RDMS and supports most of the standard SQL queries [DBC], [Tea12]. The default scenario of the test cases are built upon MySQL v5.5, this version is currently available on several top listed Cloud services (i.e. Amazon EC2, Amazon RDS, Eucalyptus, etc.).

However, each Cloud service has individual goal and principles. Therefore, to fulfill their target they are established with different data storage service. On Cloud services the database can be either a relational database or a non-relational database system. For example, Amazon RDS is an optimized scale relational database service, where Amazon SimpleDB is a highly available and flexible non-relational data store service and a fully managed NoSQL database. So that the application can use an appropriate data storage service according to their computation need. An application may use several Cloud data store services simultaneously for storing the data.

Service Provider	Data Store Service
Amazon Elastic Compute Cloud (Amazon EC2)	MySQL
Amazon Elastic Compute Cloud (Amazon EC2)	Oracle
Amazon Elastic Compute Cloud (Amazon EC2)	PostgreSQL
Amazon Relational Database Service (Amazon RDS)	MySQL
Amazon Relational Database Service (Amazon RDS)	Oracle
Amazon Virtual Private Cloud (Amazon VPC) with Amazon EC2	MySQL
Amazon SimpleDB	NoSQL

Table 4.1.: List of Cloud data storage to be tested for the evaluation of SQL support.

Now, it is a considerable issue that how an application can access Cloud data storage, as Cloud infrastructure and database service can be different than a traditional database. Therefore, in order to enable Cloud data access, it is require to investigate the existing approaches to adapt of the database layer. By running the SQL test cases, we are investigating which SQL statements have to be adapted in case the database layer is moved to the Cloud. That means that the required adaptations of SQL statements are to be determined based on testing a set of SQL test cases on the selected Cloud data store services. However, it also requires an evaluation of SQL support to observe the adaptations, which are working accordingly. Table 4.1 shows the list of Cloud data storages, which are to be tested for the evaluation of SQL support. The JUnit test results of SQL statements investigations are described in Appendix A.

After achieving the SQL adaptation results, the required adaptations are to be stored in data access layer within an application system to adapt the data access layer and then evaluate the adaptation of data access layer accessing different Cloud storage service solutions based on use case scenario. In order to test the system, a simple Java Application is to be used to send SQL requests of the different categories of the SQL Taxonomy showing that the extended Data Access Layer (DAL) is capable to perform the required transformations of the SQL statements to query the different Cloud data storage services. However, the adaptation of data access layer, we will specify, design, and realize in following chapters.

4.1. Test Setup

This section describes the detail procedure of test setup including preparation of testing environment as well as how we define the test cases. Our initial plan was to use an open-source application implemented in Java for the investigation, which adaptations of the data access layer are required in order to enable Cloud data access. Due to reduce the complexity of the work and focusing on actual research goal, we consider reusing the database of openCRX only to have an example database containing data. Moreover, it is considered to use a simple Java application instead of using a complex structured application (i.e. openCRX application) to achieve the SQL statements testing results. This also reduces considerably the thesis work load. Also, we have followed several tutorials available in online in order to prepare the Cloud service environment and to migrate the openCRX database to the Cloud data store service. Migrating traditional database to the Cloud is an important issue because the implementation of Cloud base data store services can be different than a traditional one. Therefore, it is necessary to investigate the proper database migration while database is moved to the Cloud.

In order to perform the SQL statements testing in Cloud data store service, a set of SQL statements test cases have defined. The test cases are designed based on SQL 2003 Standard [Ame03]. Moreover, to finalize the test cases, we have followed the example statements form w3schools [W3S], as currently w3schools is widely recognizable. The default test cases are written based on traditional MySQL database because it is observed that currently there are growing number of Cloud service providers offers MySQL database service and MySQL database service is widely accepted to the software application.

4.1.1. openCRX Database

As described in Section 2.4, to setup testing environment, it is considered to use the database version from openCRX version 2.9.1.

openCRX database is especially designed for the Web based application (i.e. Web service), which is usually accessible over the Internet. And we are planning to test the accessibility of Internet based database, Cloud data store services for a traditional application. Both services are available and accessible through Internet technology. A traditional application is typically accessing traditional database on local infrastructure, however this thesis focuses on accessing Cloud data store services from traditional application. According to above concepts, it is determined that openCRX database is certainly a solution for SQL statements testing. As openCRX database is already compatible for a CRM application, it makes easy to investigate the SQL statements adaptations and evaluates the database functionality with the application, while moving the database layer to the Cloud.

It is studied that openCRX is launched with HSQLBD database by default. However, according to the design specification of the application it is also run on MySQL database. Therefore, it is required to investigate whether openCRX running perfectly with MySQL database locally or not by considering default SQL test cases based on MySQL database. In order to prepare a MySQL database for openCRX, a database migration is needed. However, migrating database is become easier, as current version of openCRX provides tools to migrate from an existing database to another database. It is found that there are two functionalities called DbSchemaWizard and DbCopy are responsible for database migration, where DbSchemaWizard permits creating, validating or upgrading an openCRX database and DbCopy permits copying the openCRX data to an assigned database [Ocr12]. These tools can be used to migrate from an existing openCRX HSQLDB database to MySQL database.

4.1.2. Statements and Queries Covered by Test Cases

This section describes about how the SQL taxonomies are covered by designing the SQL test cases based on openCRX database. SQL taxonomy (see Table 2.1) provides the basis of SQL-standard, which is used to design and maintain a relational database system. The SQL is generically based upon relational algebra and tuple relational calculus [Dar05]. The basic SQL rules are fundamentally included in SQL taxonomy, for example, data insert, query, update and delete, schema creation and modification, and data access control. The test cases are however designed based on basic SQL standard rules.

Test Cases Covering

Table 4.2 shows all possible test cases mapping with SQL taxonomy where each test case is designed depending on corresponding SQL taxonomy. The SQL test cases are initially written based on a traditional MySQL database, however, these test cases are to be considered as default test cases in order to accomplish SQL statement testing on Cloud data storage service.

It does mean for investigating the SQL statement adaptations our concept is to execute the same test cases on Cloud database service while moving the database to the Cloud. However, it is examined that currently there are lot of Cloud services which does not support MySQL database, the different Cloud services established with unique approaches and different database services according to their need. Therefore, to enable accessing the same functionality of SQL-standard with an existing application on different Cloud store services, it is quite important to find the SQL adaptations and mapping with existing traditional SQL approaches.

Also, our study analysis says that in order to accomplish the explicit goal of individual Cloud services the Cloud service established with suitable data storage systems, some of them are supporting relational database systems and some of them are supporting non-relational database systems.

Table 4.2.: SQL Test cases default scenario based on SQL taxonomy.

SQL Taxonomy	Test Case
SELECT * FROM tbl *Select all rows and columns from table tbl*	SELECT * FROM oocke1_componentconfig
SELECT c1,c2 FROM tbl *Select column c1, c2 and all rows from table tbl*	SELECT object_id, access_level_browse FROM oocke1_account

SELECT c1,c2 FROM tbl
WHERE *conditions*
Select columns c1, c2 from table tbl with specific criterion
	SELECT object_id, access_level_browse FROM oocke1_account
	WHERE object_id='account/CRX/Standard/admin-Standard'

SELECT c1,c2 FROM tbl
WHERE *conditions*
ORDER BY c1 ASC, c2 DESC
Select columns c1, c2 with specific criterion and from table tbl order result by column c1 in ascending order and c2 in descending order
	SELECT object_id, name FROM oocke1_activitygroup
	WHERE pparent='activities/CRX/Mohsin'
	ORDER BY object_id ASC, name DESC

SELECT DISTINCT c1,c2 FROM tbl *Select distinct (different) rows by columns c1 and c2 from table tbl*	SELECT DISTINCT object_id, name FROM oocke1_calendar

SELECT c1, aggregate (expr) FROM tbl
GROUP BY c1
Select column c1 and use aggregate function on expression expr, group columns by column c1
	SELECT object_id, AVG (access_level_browse * access_-level_delete) FROM oocke1_calendar
	GROUP BY object_id

SELECT c1, aggregate(expr) AS c2 FROM tbl
GROUP BY c1
HAVING c2 > v1
Select column c1 and c2 as column alias of the result of aggregate function on expr. Filter group of records with c2 greater than value v1
	SELECT object_id, AVG (access_level_browse * access_-level_delete) AS access_level_browse FROM oocke1_calendar
	GROUP BY object_id
	HAVING (access_level_browse > 4)

SELECT * FROM tbl
WHERE c1 [NOT] BETWEEN v1 AND v2
Select all rows from table tbl where the value of column c1 is not in the inclusive range between v1 and v2
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE IDX NOT BETWEEN 0 AND 1

SELECT * FROM tbl
WHERE c1 [NOT] IN (v1,v2,...)
IN: to specify multiple possible values for a column
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE IDX NOT IN(0, 2)

SELECT * FROM tbl
WHERE c1 > v1 AND c1 < v2
Select all rows from table tbl where the value of column c1 is greater than v1 and less than v2
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE ((IDX > 0) AND (IDX < 2))

Continued on next page

Table 4.2: SQL Test cases default scenario based on SQL taxonomy – (continued from previous page)

SQL Taxonomy	Test Case

SELECT * FROM tbl
WHERE c1 < v1 OR c1 > v2
Select all rows from table tbl where the value of column c1 is either less than v1 or greater than v2
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE ((IDX < 1) OR (IDX > 1))

SELECT * FROM tbl WHERE c1 = v1
Select all rows from table tbl where the value of column c1 is equal to v1
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE (IDX = 1)

SELECT * FROM tbl WHERE c1 <> v1
Select all rows from table tbl where the value of column c1 is not equal v1
	SELECT object_id, owner FROM oocke1_activityprocess_
	WHERE (IDX <> 1)

CREATE VIEW vw AS
SELECT c1, c2 FROM tbl
The view vw lists all active columns (c1 and c2) from the tbl table
	CREATE VIEW oocke1_join_accthasassaddr (assigned_-address, account) AS
	SELECT addr.object_id AS assigned_address, addr.authority AS account FROM oocke1_address addr

ALTER VIEW vw AS **SELECT c1, c2 FROM tbl** *Update the view vw lists all active columns (c1 and c2) in the tbl table*	ALTER VIEW oocke1_join_accthasassaddr (assigned_ad-dress,account_) AS SELECT addr.object_id AS assigned_address, addr.authority AS account_ FROM oocke1_address addr
DROP VIEW vw *Remove view vw from the database*	DROP VIEW oocke1_join_accthasassaddr

INSERT INTO tbl (c1, c2, . . .)
VALUES (v1, v2, . . .)
Insert data into table tbl
	INSERT INTO oocke1_segment (access_level_browse, dtype, owner_, access_level_update, access_level_delete, object_id)
	VALUES (4, 'org:opencrx:kernel:account1:Segment', 2, 3, 1, 'accounts/CRX/Masud')

UPDATE tbl
SET c1 = v1, C2 = v2, . . .
WHERE ***conditions***
Update data in table tbl based on specific criterion
	UPDATE oocke1_segment
	SET access_level_browse=3, owner_=1
	WHERE (object_id='accounts/CRX/Masud')

DELETE FROM tbl
WHERE ***conditions***
Delete records from table tbl based on specific criterion
	DELETE FROM oocke1_segment
	WHERE (object_id='accounts/CRX/Masud')

CREATE TABLE tbl (**c1 datatype(length),** **c2 datatype(length),** **. . .** **PRIMARY KEY (c1)** **)** *Create table tbl with primary key is c1*	CREATE TABLE test_table1 (id INTEGER UNSIGNED NOT NULL AUTO_INCRE-MENT, name VARCHAR(250) NOT NULL, age INTEGER UNSIGNED, marks BIGINT UNSIGNED, PRIMARY KEY (id))

INSERT INTO tbl1 (c1, c2)
SELECT c1, c2 FROM tbl2
WHERE ***conditions***
Insert data from tbl2 into tbl1 based on specific criterion
	INSERT INTO test_table3 (id, name)
	SELECT id, name FROM test_table2
	WHERE (id=1)

ALTER TABLE tbl **ADD COLUMN c1 datatype(length)** *Add column c1 to table tbl*	ALTER TABLE test_table3 ADD COLUMN birthday date
ALTER TABLE tbl **DROP COLUMN c1** *Drop column c1 from table tbl*	ALTER TABLE test_table3 DROP COLUMN birthday
CREATE UNIQUE **INDEX idx ON tbl (c1, c2, . . .)** *Create a unique index on tbl table. Duplicate values are not allowed*	CREATE UNIQUE INDEX idx ON test_table3 (id, name)
DROP INDEX tbl.idx Drop an index idx from table tbl	DROP INDEX idx ON test_table3

Continued on next page

Table 4.2: SQL Test cases default scenario based on SQL taxonomy – (continued from previous page)

SQL Taxonomy	Test Case
TRUNCATE TABLE tbl *Drop table tbl and re-create it, all data is lost*	TRUNCATE TABLE test_table3
DROP TABLE tbl *Remove table tbl from database*	DROP TABLE test_table3
SELECT * FROM tbl1 **INNER JOIN tbl2 ON** ***join-conditions*** *Inner join table tbl1 with tbl2 based on join-conditions*	SELECT access_level_browse, full_name FROM oocke1_-account INNER JOIN oocke1_account_ ON (oocke1_ac-count.object_id=oocke1_account_.object_id)

SELECT * FROM tbl1
INNER JOIN tbl2 ON ***join-conditions***
WHERE ***conditions***
Inner join table tbl1 with tbl2 based on join-conditions with specific criterion
	SELECT access_level_browse, full_name FROM oocke1_-account
	INNER JOIN oocke1_account_ ON (oocke1_ac-count.object_id=oocke1_account_.object_id)
	WHERE (oocke1_account_.idx=1) ORDER BY oocke1_ac-count.full_name

SELECT * FROM tbl1
LEFT JOIN tbl2 ON ***join-conditions***
Left join table tbl1 with tbl2 based on join-conditions
	SELECT access_level_browse, full_name FROM oocke1_-account
	LEFT JOIN oocke1_account_ ON (oocke1_account.object_-id=oocke1_account_.object_id)
	WHERE (oocke1_account_.idx=2)

SELECT * FROM tbl1
RIGHT JOIN tbl2 ON ***join-conditions***
Right join table tbl1 with tbl2 based on join-conditions
	SELECT access_level_browse, full_name FROM oocke1_-account
	RIGHT JOIN oocke1_account_ ON (oocke1_ac-count.object_id=oocke1_account_.object_id)
	WHERE (oocke1_account_.idx=0)

SELECT * FROM tbl1, tbl2
WHERE ***join-conditions***
Select all values from table tbl1 and tbl2 based on join-conditions
	SELECT persons.LastName, persons.FirstName, or-ders.OrderNo FROM persons, orders
	WHERE ((persons.P_Id > 1) AND (orders.P_Id < 3))

SELECT * FROM tbl1
FULL OUTER JOIN tbl2 ON ***join-conditions***
Full outer join table tbl1 with tbl2 based on join-conditions
	SELECT persons.LastName, persons.FirstName, or-ders.OrderNo FROM persons FULL JOIN orders ON persons.P_Id=orders.P_Id ORDER BY persons.LastName
	
	The SQL rule cannot be covered by the SQL test cases, as MySQL lacks support for FULL OUTER JOIN.
	
	Emulation of MySQL FULL JOIN:
	SELECT Persons.LastName, Persons.FirstName, Or-ders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id
	UNION
	SELECT Persons.LastName, Persons.FirstName, Or-ders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id

In default SQL test cases design, it is intended to cover complete taxonomy; therefore, in this scope we consider the basic SQL rules, such that Select statements, Update database, Operators, Table statements, Join statements, Index statements, View statements. Unfortunately, there is taxonomy in SQL join statements called FULL OUTER JOIN cannot be covered in this scope, as MySQL lacks support for this taxonomy. It can be possible to emulate this taxonomy by doing an UNION of a LEFT JOIN and a RIGHT JOIN of selected tables, but having the question of original taxonomy use this SQL rule is included in the test cases set, as FULL JOIN functionality is implemented perfectly in other database services, i.e. Oracle, PostgreSQL. Furthermore, to achieve the maximum SQL taxonomy the openCRX database is extended with some tables.

4.2. JUnit Test for Automatic Testing

JUnit is an extensible unit testing framework for implementing testing easy by simplifying the writing of test cases and by providing automated running of tests and test suites in Java. It provides a straightforward manner to specifically test specific areas of a Java program. By using JUnit framework, it is possible to employ testing a single or even multiple units of a hierarchy of program code.

In order of testing full functionality of a Java program, JUnit testing framework is very useful and advantageous to use. Because of according to the concept of JUnit framework and achieving positive results, it is necessary to define explicit expected outcomes of a specific program execution routes. However, a test can be written with expected outcomes during debugging a program and the debugging can be preformed until the test comes out positive. Moreover, a project can have several core-components and the components can be dependent each other. In this situation, if we modify a specific areas of the project, it is possible to monitor the immediate effect of the modification and also possible to observe the effect of dependent components immediately. Another advantage of using JUnit testing is that before coding it is possible to test all necessary units of a program [Mil05].

Therefore, for testing the SQL test cases, it is considered to use the JUnit testing because it provides the facility of automatic running tests and also possible to compare the expected results of specific test case in a simple way. To realize the explicit test result, the test cases are defined with individual test methods. Using individual methods the test becomes easier to observe, which SQL statements are passed and which are failed. In addition, a simple Java application "SQLQueryAnalyzer" is developed to prepare the expected results. The query analyzer is capable to execute a set of SQL queries on local database system and then return the execution results, the achieving results might be considered as expected outputs for SQL test cases.

4.3. Test Scenario

In order to determine the necessary adaptations of the SQL statements for accessing Cloud data storage services, a set of SQL test cases are to be executed on different selected Cloud data storage services. The SQL test cases are defined in previous section. The test cases are to be executed on traditional SQL server as well as on Cloud database systems for the investigation, which SQL queries and statements have to be adapted in case the database layer is moved to the Cloud. This section describes the different scenarios and outcomes of SQL test cases for both traditional and Cloud database services. In addition, the usage diagram for corresponding test scenarios are originated from Strauch et al. publication on Cloud Data Hosting Solutions [SKLU11].

4.3.1. Default Scenario

This section describes default test scenario of SQL statements testing where application accessing traditional MySQL database on local server. In order to achieve accurate results, it is considered to use the same version of database setting for all similar testing. For example, MySQL database can be implemented in several Cloud services and for all cases it should considered implementing with MySQL version v5.5. Therefore, to setup default testing scenario, it is considered by using MySQL version v5.5. In general, a traditional application is built with three layer architecture, i.e. presentation layer, business logic layer and database layer. However, according to the purpose of data accessing from Cloud data store services a Data Access layer (DAL) has to be implemented in current application architecture model. This introduces a four layer application architecture model. The DAL is placed between Business Logic Layer and Database Layer where it is responsible for encapsulating the data access functionality.

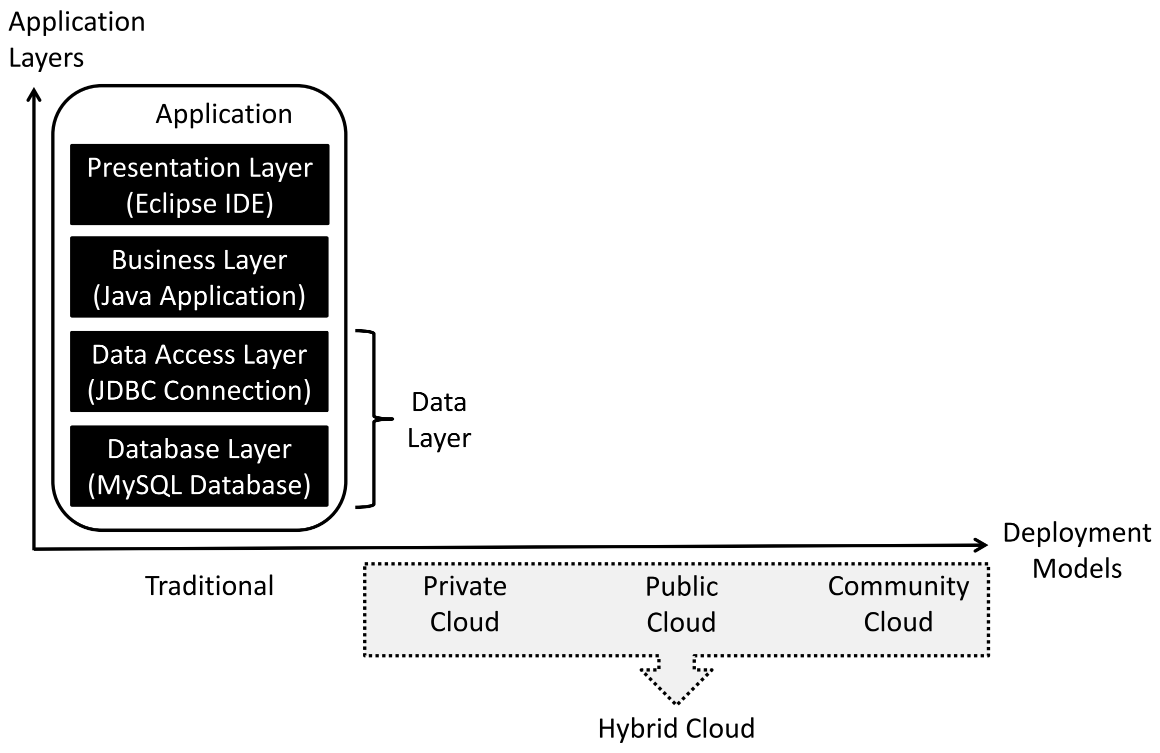

Application Layers
Application
Presentation Layer (Eclipse IDE)
Business Layer (Java Application)
Data Access Layer (JDBC Connection)
Database Layer (MySQL Database)
Data Layer
Deployment Models
Traditional
Private Cloud
Public Cloud
Community Cloud
Hybrid Cloud

Figure 4.1.: Scenario overview of traditional MySQL database version 5.5 running on local server and submit the SQL queries.

Figure 4.1 shows a layer based architecture model of a traditional application where application runs on local traditional database server. However, the application does not run on any distributed computing or Cloud storage services. In default scenario setting, a Java application is running on local server and accessing traditional MySQL database where the application is implemented using JUnit framework in Eclipse platform. Having comparison with traditional application layers, the Java application (SQL statements testing) is running on business layer where Java Database Connectivity (JDBC) connection and MySQL database represents DAL and DBL respectively. The Eclipse Integrated Development Environment (IDE) is considered a presentation layer because the application is running in Eclipse platform to investigate the results of JUnit testing. In addition, to setup the default testing scenario, it is necessary to migrate the openCRX database to traditional MySQL database, as openCRX originally comes with HSQLDB database. Furthermore, a middleware driver, JDBC connector is required to connect the database from application. In order to establish the connection to the traditional database server, a recent JDBC driver and standard connection string has used.

Interpretation of Test Results

The default test settings are defined with expected outputs for every SQL statements designing by individual JUnit test methods. The expected outputs are achieved by running a Java application called SQLQueryAnalyzer. After successfully setting up the JUnit testing methods, the JUnit test suite has run to compare the actual outputs with expected outputs automatically. While running JUnit tests, it shows the outcomes which methods (e.g. SQL statements) are passed and which are failed. Using JUnit testing framework it is also possible to visualize of comparison result between expected and actual output in case a method fails. From this comparison result one can easily realize why method fails and then adapt the method immediately.

Table A.1 shows JUnit tests results for default test scenario where all SQL test cases are passed on traditional MySQL database except MySQL FULL JOIN functionality, as MySQL lacks supporting FULL JOIN. This feature can be emulated using UNION operation between LEFT JOIN and RIGHT JOIN of two tables for achieving expected result. This is just a possible solution, but the results may vary comparing with actual FULL JOIN feature depending on table descriptions. Furthermore, while running SQL test cases it is realized that SQL statements (i.e. create table, update, delete) are not returning a result set, as it has no way to represent the result sets of a database changes. It can be proved the stored procedure runs without error by using query analyzer.

4.3.2. MySQL on Amazon EC2

This section describes the test scenario of access enabling MySQL data store service on Amazon EC2 public Cloud from traditional application. However, in default scenario the application is accessing database on local server. By comparing with default scenario application model, the presentation layer, business layer and data access layer are remaining in the same place on local server, only database layer is now move to the Cloud.

In order to setup MySQL data store service testing environment on Amazon EC2 Cloud, there are several steps have to be done. Using Web based *AWS Management Console*, we can control the Cloud environment, for example, launch a new instance, start, reboot, stop and terminate an instance. Amazon EC2 provides the opportunity to launch a few of pre-configured Machine

Image [EC2]. To setup MySQL testing, a pre-configured Amazon Machine Image (AMI) is chosen called *amzn-ami-pv-2012.03.1.i386-ebs* which is a Linux platform. During launching an instance, it automatically checks monitor the AWS systems as well as the software and network configuration for the instance. However, it is important to setup security groups with declaring necessary ports, as application is accessing database service on Amazon EC2 Cloud using SSH tunneling. For example, the secure communication establishes on port 22, the default MySQL accessing port is 3306 and so on.

For SQL statements testing on Amazon EC2, it is considered using MySQL version v5.5, because default test cases are designed based on traditional MySQL v5.5. Different version of database can produce different results; therefore, it is important to keep the same version of database setting for achieving accurate results. To upload and migrate the openCRX database to MySQL data store service on Amazon EC2, the *MySQL Workbench tool* can be used with establishing a secure communication by SSH tunneling.

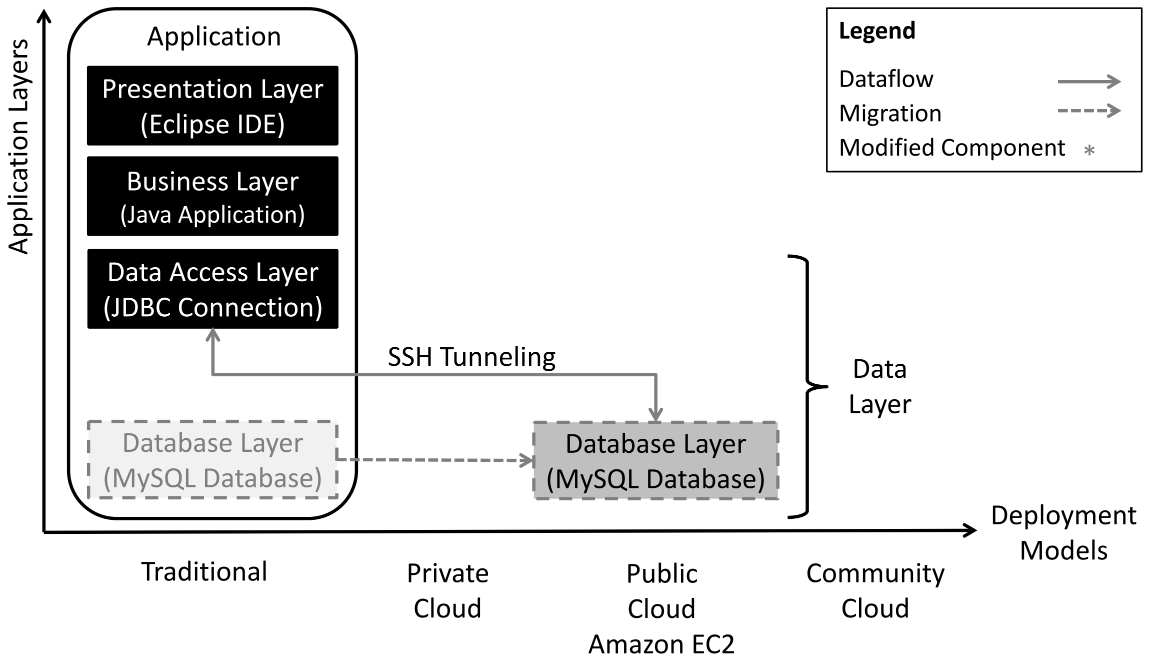


Figure 4.2.: Scenario overview of SQL test cases running on MySQL data store service in Amazon EC2.

While Amazon EC2 allow accessing MySQL data store service, the traditional application can take advantage of accessing data with elastic volume of storage capacity and facility of using high computations which required by the application. To ensure these facilities, it is necessary to investigate that MySQL server supports full functionality of SQL-standard as traditional MySQL server does. Therefore, it is considered to run SQL taxonomy based a set of selected SQL statement test cases in MySQL database on Amazon EC2, to check whether all test cases are passed. In order to test MySQL database on Amazon EC2, the same architecture model is used. Figure 4.2 shows a four layer application architecture, where Eclipse IDE is considering a presentation layer, the Java application is running on business layer and openCRX MySQL database (database layer) has moved to Cloud (Amazon EC2), and then the application accessing MySQL database on Amazon EC2 through JDBC connection via SSH tunneling.

Interpretation of Test Results

To setup JUnit testing for MySQL statements testing on Amazon EC2, we have followed the same principles of default MySQL test setting. Even it is not necessary to adapt the MySQL connection string, as the application is connecting with MySQL database on Amazon EC2 instance using SSH tunneling. Nevertheless, it is important to create SSH connection with a specific port (e.g. 3306) in order to connect the MySQL data store service. After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the actual results with expected results automatically. While running JUnit testing, it is observed that we have achieve same outcomes as default SQL testing (see Section 4.3.1). This means that all test cases (SQL statements) are passed except MySQL FULL JOIN, JUnit results are described in Table A.1. However, it is also realized that emulating FULL JOIN properties with UNION operation between LEFT JOIN and RIGHT JOIN of two tables produces the same results.

4.3.3. Oracle on Amazon EC2

This section describes the test scenario of accessing Oracle data store service on Amazon EC2 public Cloud from traditional application. However, in default scenario the application is accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure where only database layer is moved to the Cloud however.

In order to setup Oracle data store service testing environment on Amazon EC2 Cloud, there are several steps have to be done. The first step is to setup and manage Oracle database environment on Amazon EC2. Using a Web based *AWS Management Console*, we can control the Amazon EC2 Cloud environment. Amazon EC2 provides the opportunity to launch a few of pre-configured Machine Image [EC2] especially for Oracle database service. To setup Oracle data store service testing, we have chosen a pre-configured Amazon Machine Image (AMI) called *Oracle Linux 5.6 x86_64 - Amazon Xen* which is a Linux platform. After successfully launching an instance on Amazon EC2 for Oracle database, it is important to setup security groups with declaring necessary ports, as application is accessing database service on Amazon EC2 Cloud using SSH tunneling. For example, the secure communication establishes on port 22, the default Oracle accessing port is 1521 and so on.

To setup Oracle database service environment for SQL statements testing on Amazon EC2, it is considered implementing Oracle version v11g enterprise edition. To upload and migrate the openCRX database to Oracle data store service on Amazon EC2, the *Oracle SQL Developer tool* can be used with establishing a secure communication by SSH tunneling. Migrating openCRX MySQL database in Oracle is straightforward, however, there are adaptations needed, for example tables, indexes and views are to be migrated individually. Also, it is required to rename the column as *object_id* because the column name automatically acquired to *object_id_* while copying data table from MySQL to Oracle. The column *object_id* is a primary key for all openCRX data tables.

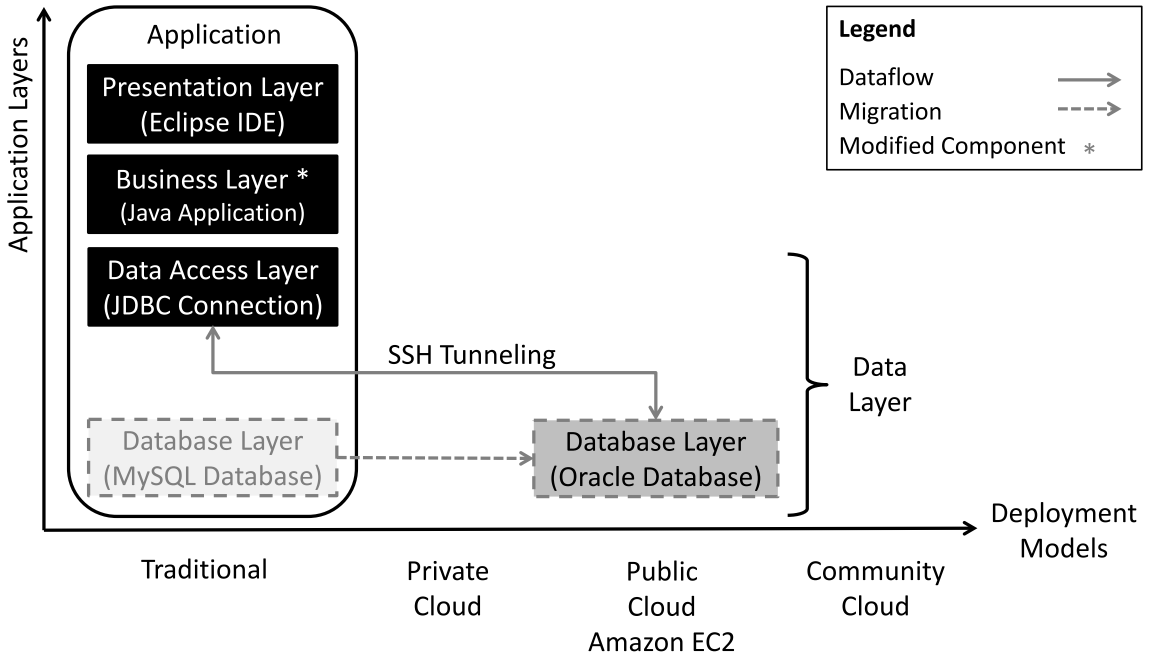


Figure 4.3.: Scenario overview of SQL test cases running on Oracle data store service in Amazon EC2.

While Amazon EC2 allow accessing Oracle data store service, the traditional application can take advantage of accessing data with elastic volume of storage capacity and facility of using high amount of computations required by the application. To ensure these facilities, it is necessary to investigate that Oracle server supports full functionality of SQL-standard on Amazon EC2 as the traditional Oracle server does. Therefore, it is considered to run SQL taxonomy based a set of selected SQL statement test cases in Oracle database on Amazon EC2, to check whether all test cases are passed or not. In order to test Oracle database on Amazon EC2, the same architecture model is used. Figure 4.3 shows a four layered application architecture, where the Eclipse IDE is considering a presentation layer, the Java application is running on business layer and openCRX Oracle database (database layer) has moved to the Cloud (Amazon EC2.Then the application accessing Oracle database on Amazon EC2 through a JDBC connection via SSH tunneling.

Interpretation of Test Results

To setup JUnit testing for Oracle statements testing on Amazon EC2, we use the same principles as default test setting (see Section 4.3.1). However, the Oracle database connection string has to be adapted according to the Oracle default connection setting. This means that traditional connection setting is enough to connect the Oracle data store service on Amazon

EC2, as application is connecting Oracle database using SSH tunneling. In addition, it is important to create SSH connection with a specific port (e.g. 1521) forwarding in order to connect the Oracle data store service, as Oracle database specified with port 1521. After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the actual results with expected results automatically. While running JUnit testing, it is observed that several SQL statements tests are failed which has to be adapted in order to enable accessing Oracle data store service on Amazon EC2.

Table A.2 shows the JUnit test results for SQL statements testing on Oracle data store service in Amazon EC2 Cloud where JUnit methods executes SQL statements on Oracle and compare with expected SQL result sets. It is realized that Oracle supports FULL JOIN functionality. According to the outcomes of JUnit SQL statements testing, we found that the test case number 1, 6, 7, 15, 20, 22, 25, 29 and 33 are initially failed. However, the test case number 29 and 33 are returning the expected results, but in different order. They can be adapted by simply adding SQL ORDER BY expression in the SQL statement. The result analysis in test case 33, it can be noticed that the emulating result of FULL JOIN in MySQL produces different ordering results than Oracle and PostgreSQL. By comparing the test case results, it is observed that the Oracle and PostgreSQL returns the same results for test case 33. Some test cases (i.e. test case number 1, 6, 7) failed because of accessing number format, for example, Oracle returns only integer number value where MySQL returns floating point value with point zero. More precisely, it can be determined that Oracle does not return a zero value after point of a number, i.e. if the value is 3.0000 then it will return only 3. Moreover, the test case number 7 is initially failed, as implementing GROUP BY expression in Oracle is different than MySQL in this scope.

Furthermore, the test case numbers 15, 20, 22 and 25 are failed because of syntax the structure implementing in Oracle is different than MySQL SQL syntax. For example, the MySQL ALTER VIEW can be adapted by CREATE OR REPLACE VIEW and adding a column in Oracle table it is not required to specify with keyword COLUMN. In order to create a table in Oracle, it follows the same standard as MySQL does, but some of the data types are different. Therefore, creating a table can be adapted by declaring correct data sets. To DROP an index in Oracle, it is not necessary to specify the table name, the statement can be adapted by eliminating the key word *ON tableName*. While running SQL test cases on Oracle database in Amazon EC2 Cloud, it is also realized that SQL statements (i.e. create table, update, delete) are not returning a result set of database changes like MySQL database. However, this can be proved the stored procedure runs without error by using *Oracle SQL Developer tool*.

4.3.4. PostgreSQL on Amazon EC2

This section describes the test scenario of accessing PostgreSQL data store service on Amazon EC2 public Cloud from traditional application. However, in default scenario the application is accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure where only database layer, openCRX PostgreSQL database is moved to the Cloud however.

In order to setup PostgreSQL testing environment on Amazon EC2, there are several steps have to be done. The first step is to setup and manage PostgreSQL database environment on Amazon EC2. Using a Web based *AWS Management Console*, we can control the Amazon EC2 Cloud environment. Amazon EC2 provides the opportunity to launch a few of pre-configured Machine Image [EC2]. To setup PostgreSQL data store service testing, it is chosen pre-configured Amazon Machine Image (AMI), a Linux platform and mount a new EBS volume to store or archive the data. After successfully launching an instance on Amazon EC2 for PostgreSQL database, it is significant to setup security groups with declaring necessary ports, as application is accessing database service on Amazon EC2 Cloud using SSH tunneling. For example, the secure communication establishes on port 22, the default Oracle accessing port is 5432 and so on.

After launching an instance on Amazon EC2, the next step is to migrate the openCRX MySQL database to PostgreSQL database and we consider migrating in PostgreSQL version 9.0 using *PostgreSQL Maestro*. Migrating openCRX MySQL database into PostgreSQL is straightforward, however, adaptations are needed, for example tables, indexes and views are to be migrated manually. After successfully preparing the database setup in Amazon EC2, the openCRX data has to be inserted separately, which can be done using *PostgreSQL Maestro* with establishing a SSH tunneling. However, it becomes a complex process in order of migrating a database. There are several third party tools can be found in the current market which provides the facility of migrating a MySQL database to PostgreSQL database. Unfortunately, none of them are offering free. Therefore, to reduce the work load of database migration, initially we migrate the database on local server and then dump the database to upload on Amazon EC2 Cloud.

To setup PostgreSQL database service environment for SQL statements testing on Amazon EC2, it is considered implementing PostgreSQL version 9.0 because traditional PostgreSQL migration is based on PostgreSQL v9.0. Different version of database can produce different results. Therefore, it is important to keep the same version of database setting. To upload and migrate the openCRX database to PostgreSQL data store service on Amazon EC2, the *PostgreSQL Maestro* can be used with establishing a secure communication by SSH tunneling.

While Amazon EC2 supports PostgreSQL database, the traditional application can take advantages to access data with elastic volume of storage capacity and the facility of using high amount of computations if required by the application. To ensure these facilities, it is necessary to investigate that whether PostgreSQL server supports full functionality of SQL-standard as traditional PostgreSQL server does. Therefore, it is considered to run SQL taxonomy based selected SQL statement test cases in PostgreSQL database on Amazon EC2, to check whether all test cases are passed or not. In order to test the PostgreSQL database on Amazon EC2, the same architecture model is used as default application setting. Figure 4.4 shows a four layered application architecture, where Eclipse IDE is considering a presentation layer, the Java application is running on business layer and PostgreSQL database (database layer) has moved to the Cloud Amazon EC2. Then the application accessing PostgreSQL database on Amazon EC2 through a JDBC connection via SSH tunneling.

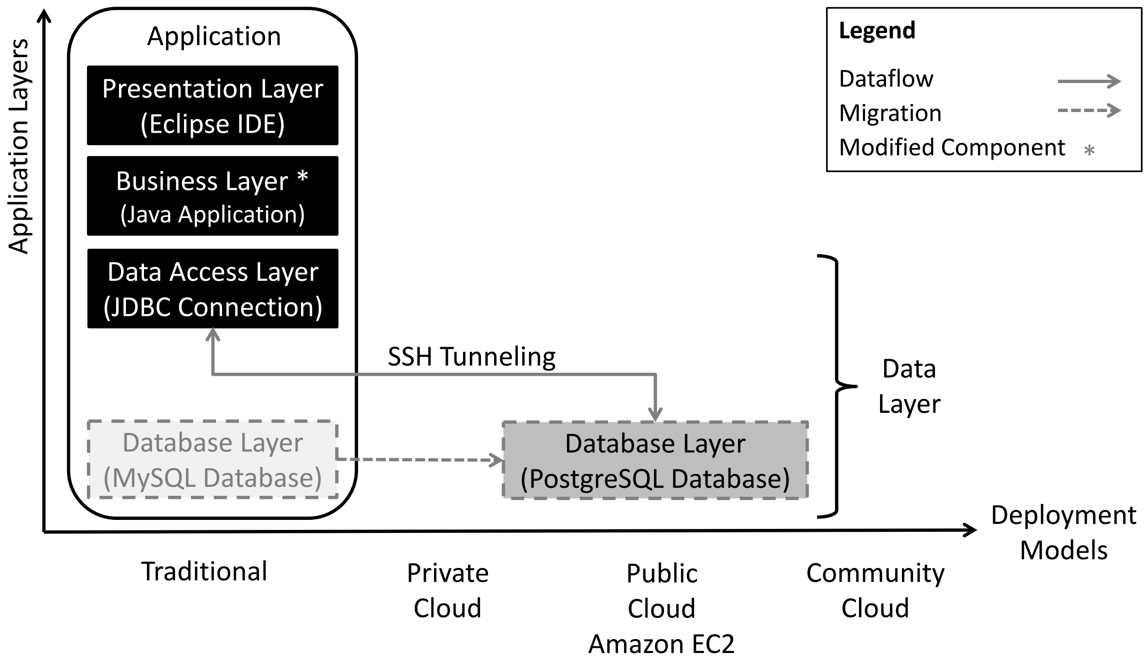


Figure 4.4.: Scenario overview of SQL test cases running on PostgreSQL data store service in Amazon EC2.

Interpretation of Test Results

To setup JUnit testing for PostgreSQL statements testing on Amazon EC2, we have followed the same principles as default test setting (see Section 4.3.1). However, the PostgreSQL database JDBC connection string has to be adapted according to the PostgreSQL default connection setting. This means that traditional connection setting is enough to connect the PostgreSQL data store service on Amazon EC2, as application is connecting PostgreSQL database using SSH tunneling. In addition, it is required to create SSH connection with a specific port (e.g. 5432) forwarding in order to connect the PostgreSQL data store service, as PostgreSQL database is specified with port 5432. After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the actual results with expected results automatically. While running JUnit testing, it is observed that there are several SQL statements tests are failed, which has to be adapted in order to enable accessing PostgreSQL data store service on Amazon EC2.

Table A.3 shows the JUnit tests results for SQL statements testing on PostgreSQL data store service in Amazon EC2 Cloud where JUnit methods executes SQL statements on PostgreSQL and compare with expected SQL result sets. It is investigated that PostgreSQL supports FULL JOIN functionality. According to the outcomes of JUnit SQL statements testing, we found that test case number 1, 2, 5, 7, 15, 20, 25, 31, 32 and 33 are initially failed. However, the test case number 2, 28, 31, 32 and 33 are returning the expected results, but in different order. They can be adapted by simply adding SQL ORDER BY expression in the SQL statement. The result analysis in test case 33, it can be noticed that the emulating result of FULL JOIN in MySQL produces different ordering results than Oracle and PostgreSQL. By comparing the test case results, it is observed that the Oracle and PostgreSQL returns the same results for test case 33. Moreover, some test cases (i.e. test case number 1, 6, 7) are failed because of accessing number format, for example the time format in MySQL can be 18:06:38.0 where PostgreSQL eliminates last point zero value in test case number 1. Also, in the case of fraction number the MySQL can have four digit numbers after pointing where PostgreSQL returns sixteen digits after point value, i.e. if the value is 6.0000 in MySQL then the PostgreSQL will return 6.0000000000000000 correspondingly. This difference can be determined by analyzing test case number 6 and 7. It is observed that returning these extra digit causes of different data structure supporting between MySQL and PostgreSQL database systems. However, the business layer in an application system has to deal with this issue. The structure of test case number 7 is to be adapted, as implementing GROUP BY expression in PostgreSQL is different than MySQL in this scope.

Furthermore, the test case number 15, 20 and 25 are failed because of syntax structure implementing in PostgreSQL is different than MySQL SQL syntax. The MySQL ALTER VIEW can be adapted by CREATE OR REPLACE VIEW. In order to create a table in PostgreSQL, it follows the same standard as MySQL does, but some of the data types are different. Therefore, creating a table can be adapted by declaring correct data sets. Also, to DROP an index in PostgreSQL it is not necessary to specify the table name, the statement can be adapted by eliminating the key word *ON tableName*. While running SQL test cases on PostgreSQL database in Amazon EC2 Cloud, it is realized that SQL statements (i.e. create table, update, delete) are not returning a result set of a database changes like MySQL (a default database). It can be proved the stored procedure runs without error by using *PostgreSQL Maestro* tool.

4.3.5. MySQL on Amazon RDS

This section describes the test scenario of accessing MySQL data store service on Amazon RDS public Cloud from traditional application. However, in default scenario the application is accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure, only database layer is moved to the Cloud however.

In order to seup the MySQL data store service testing environment on Amazon RDS, there are several steps have to be done. Using a Web based AWS Management Consol or Amazon RDS APIs, it is easy to launch a DB instance and manage the Cloud environment. Amazon RDS provides the opportunity to launch a pre-configured database instance for MySQL community server where database port (i.e. 3306) is a default setup while launching a MySQL DB instance. After successfully launching a DB instance, it is essential to add an appropriate local IP address to the DB security group option where database has to be accessed. To setup the SQL statements testing, we select MySQL version v5.5 because the default test cases are designed based upon traditional MySQL v5.5. Using different version of database can produce different result; therefore, it is significant to keep the same version of database setting. To upload and migrating the openCRX MySQL database on Amazon RDS MySQL DB instance, the MySQL Workbench tool can be used with Amazon RDS endpoint of database (host name).

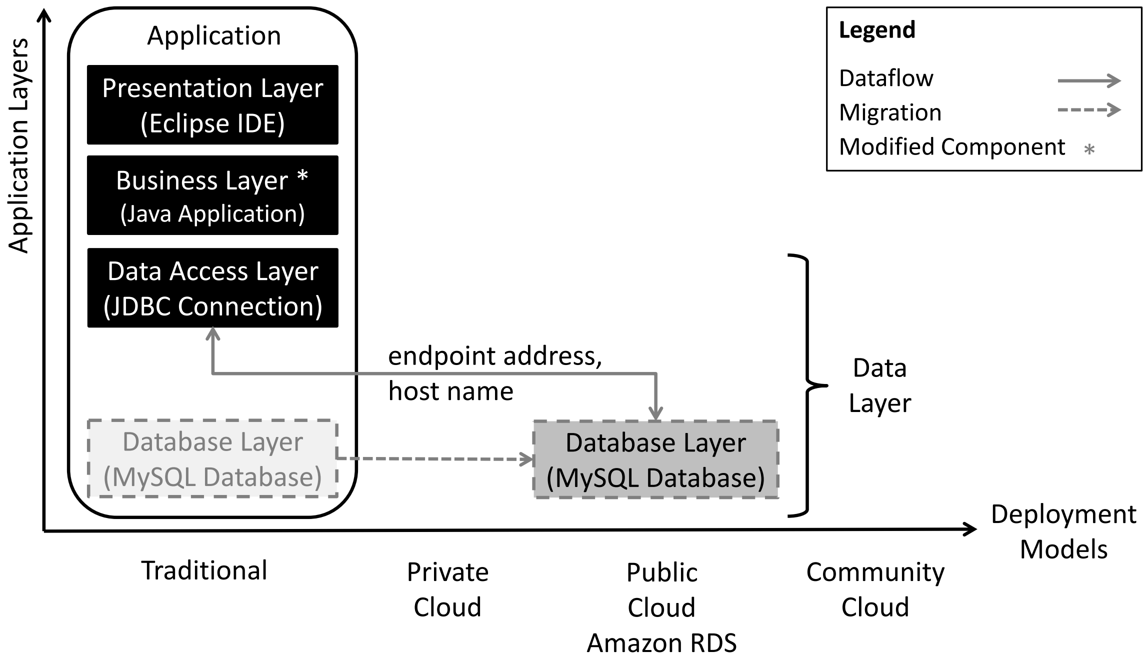

Application Layers
Application
Presentation Layer (Eclipse IDE)
Business Layer * (Java Application)
Data Access Layer (JDBC Connection)
Database Layer (MySQL Database)
endpoint address, host name
Database Layer (MySQL Database)
Legend
Dataflow
Migration
Modified Component *
Data Layer
Deployment Models
Traditional
Private Cloud
Public Cloud Amazon RDS
Community Cloud

Figure 4.5.: Scenario overview of SQL test cases running on MySQL data store service in Amazon RDS.

While Amazon RDS allow accessing MySQL native database, the application can take advantage to access data with on-demand storage capacity and the facility of using high amount of computations which is required by the application. To ensure this facility, it is necessary to investigate that MySQL server supports full functionality of SQL-standard on Amazon RDS as traditional MySQL server does. Therefore, it is considered to run SQL taxonomy based a set of selected SQL statement test cases on MySQL database in Amazon RDS, to check whether all test cases are passed. In order to test the MySQL database on Amazon RDS, the same architecture model is used as default application setting. Figure 4.5 shows a four layered application architecture, where Eclipse IDE represents presentation layer, the Java application running on business layer and MySQL database (database layer) has moved to the Cloud (Amazon RDS). Then application accessing MySQL database on Amazon RDS through JDBC connection with Amazon RDS endpoint of database, a host name i.e. mysqldbinstance.ctdc07yqmwpy.us-east-1.rds.amazonaws.com.

Interpretation of Test Results

To setup JUnit testing for MySQL statements testing on Amazon RDS, we use the same principles as default test setting. However, the MySQL database connection string has to be adapted with endpoint of database (host name). After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the expected results with actual results automatically. While running JUnit testing, it is observed that we achieve same outcomes as default SQL testing (see Section 4.3.1). This means that all test cases (SQL statements) are passed on MySQL data store service in Amazon RDS Cloud except MySQL FULL JOIN (see Table A.1). However, it is also realized that emulating FULL JOIN properties with UNION operation between LEFT JOIN and RIGHT JOIN of two tables produces the same results. Furthermore, it is also realized that SQL statements (i.e. create table, update, delete) are not returning a result set on MySQL database in Amazon RDS Cloud as we have the same behavior in default SQL testing.

4.3.6. Oracle on Amazon RDS

This section describes the test scenario of accessing Oracle data store service on Amazon RDS public Cloud from traditional application. However, in default scenario the application was accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure, only database layer is moved to the Cloud however.

In order to setup the Oracle data store service testing environment on Amazon RDS, there are several steps have to be done. The first step is to setup and manage Oracle database environment on Amazon RDS. Using a Web based AWS Management Consol or Amazon Oracle APIs, it is easy to launch a DB instance and manage the Cloud environment. Amazon RDS provides the opportunity to launch a pre-configured database instance for Oracle where database port (i.e. 1521) is automatically setup while launching an Oracle DB instance. After successfully launching a DB instance, it is essential to add an appropriate local IP address to the DB security group option where database has to be accessed like the way we have done for MySQL test setting in Amazon RDS (see Section 4.3.5). To setup Oracle database service environment for SQL statements testing on Amazon EC2 it is considered implementing Oracle version v11g enterprise edition. Using the same version of database setting in similar types of testing is beneficial for achieving a better comparison results. After successfully setting up Oracle database environment on Amazon RDS, the next step is to migrate the openCRX MySQL database to Oracle database. To migrate the database in Oracle, we have followed the same procedure as is done for migrating Oracle database in Amazon EC2. The migration procedure of openCRX database in Oracle database is described in Section 4.3.3. However, to upload and migrating the openCRX Oracle database to Amazon RDS Oracle DB instance, the Oracle SQL Development tool can be used with Amazon RDS endpoint of database (host name).

While Amazon RDS allow accessing Oracle native database, the application can take advantages to access data with on-demand storage capacity and facility of using high amount of computations required by the application. To ensure this facility, it is necessary to investigate that Oracle server supports full functionality of SQL-standard on Amazon RDS as traditional Oracle server does. Therefore, it is considered to run a set of SQL taxonomy based SQL statement test cases on Oracle database in Amazon RDS, to check whether all test cases are passed. In order to test the Oracle database on Amazon RDS, the same architecture model is used, as defined in default SQL test setting. Figure 4.6 shows a four layered application

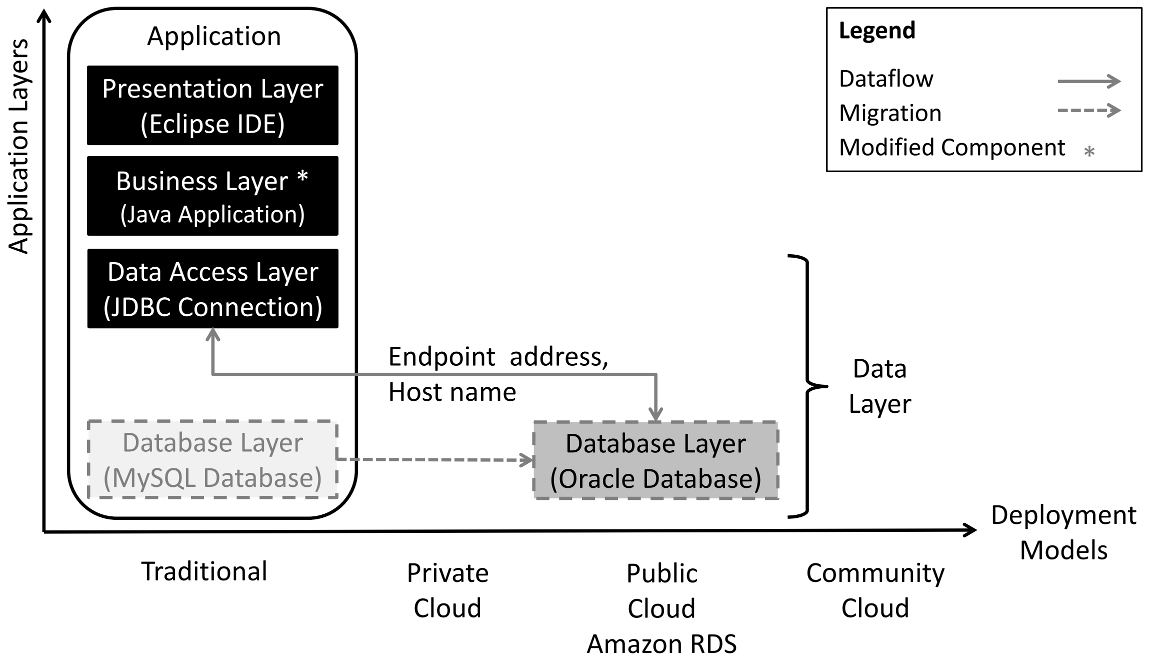

Application Layers
Application
Presentation Layer (Eclipse IDE)
Business Layer * (Java Application)
Data Access Layer (JDBC Connection)
Database Layer (MySQL Database)
Endpoint address, Host name
Database Layer (Oracle Database)
Data Layer
Deployment Models
Traditional
Private Cloud
Public Cloud Amazon RDS
Community Cloud
Legend
Dataflow
Migration
Modified Component *

Figure 4.6.: Scenario overview of SQL test cases running on Oracle data store service in Amazon RDS.

architecture, where Eclipse IDE represents presentation layer, the Java application running on business layer and Oracle database (database layer) has moved to the Cloud (Amazon RDS). Then application accessing Oracle database on Amazon RDS through JDBC connection with Amazon RDS endpoint of database, a hostname i.e. oracledbinstance.ctdc07yqmwpy.us-east-1.rds.amazonaws.com.

Interpretation of Test Results

To setup JUnit testing for Oracle statements testing on Amazon RDS, it is consider using same principles as in Amazon EC2 Oracle test setting (see section 4.3.3). However, the Oracle database connection string has to be adapted with endpoint of database (host name). After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the expected results with actual results automatically and we achieve same outcomes as in Amazon EC2 Oracle testing (see Section 4.3.3). This means that JUnit tests results for SQL statements testing on Oracle data store service on Amazon RDS Cloud where JUnit produces same results as Amazon EC2 Oracle test results. The test case number 1, 2, 5, 6, 7, 15, 20, 25, 31 and 33 are failed in Oracle Amazon RDS which are the same results we achieved in Oracle Amazon EC2 (see Table A.2).

4.3.7. MySQL on Amazon VPC

This section describes the test scenario of accessing MySQL data store service from traditional application on Amazon VPC private Cloud. However, in default scenario the application is accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure, only database layer is moved to the Cloud however.

To setup MySQL data store service testing environment on Amazon VPC, there are several steps has to be done. Amazon VPC offers the opportunities to launch four different types of network infrastructures, i.e. VPC with a Single Public Subnet Only, VPC with Public and Private Subnets, VPC with Public and Private Subnets and Hardware Virtual Private Network (VPN) Access, and VPC with a Private Subnet Only and Hardware VPN Access. For MySQL testing purpose we select a basic and simple structure, VPC with a Single Public Subnet Only where instances run in a private, isolated section of the Amazon Cloud and provide severe control over inbound and outbound network traffic to the instances. However, there is a "Getting started guide" [VPC11] provided by Amazon Cloud service can be used to set up the Amazon VPC environment. While launching an instance on Amazon VPC, it is necessary to prepare and configure Amazon EC2 instance for attaching with VPC. The configuration procedure of MySQL data store service in Amazon EC2 is described in Section 4.3.2. However, to setup the MySQL database on Amaton EC2 it is considered installing MySQL version v5.5 because default test cases are designed based upon traditional MySQL v5.5.

After launching Amazon EC2 instance now we have to setup VPC and Internet gateway. In order to setup Amazon VPC, it is necessary to attach an Internet gateway, create subnet and setup routing to the VPC. Then, the security groups have to be setup with declaring necessary ports in inbound and outbound option, e.g. port 80 for allowing HTTP access, port 443 for Hypertext Transfer Protocol Secure (HTTPS) access, port 22 for SSH access and 3389 for Remote Desktop Protocol (RDP) access control. Also, it is important to assign an Elastic IP address to the running instance. This Elastic IP address treating as public IP address to the private instance, so that a VPC instance can be reached directly from the Internet. This means that an application can access MySQL database on Amazon VPC using public IP address

While MySQL database can be access in a private Cloud infrastructure in Amazon VPC, the traditional application can take advantage to access data with elastic volume of storage capacity and facility of using high amount of computations required by the application attaching with Amazon EC2 instance. To ensure this facility, it is essential to investigate that MySQL server supports full functionality of SQL-standard on Amazon VPC as traditional MySQL server does. Therefore, it is considered to run a set of SQL taxonomy based SQL statement test cases in MySQL database on Amazon VPC, to check whether all test cases are passed. In order to test the MySQL database on Amazon VPC, the same architecture model is used as define in default test setting. Figure 4.7 represents a four layer application architecture, where Eclipse IDE can be represented a presentation layer, the Java application running on business layer and MySQL database (database layer) has moved to the Amazon

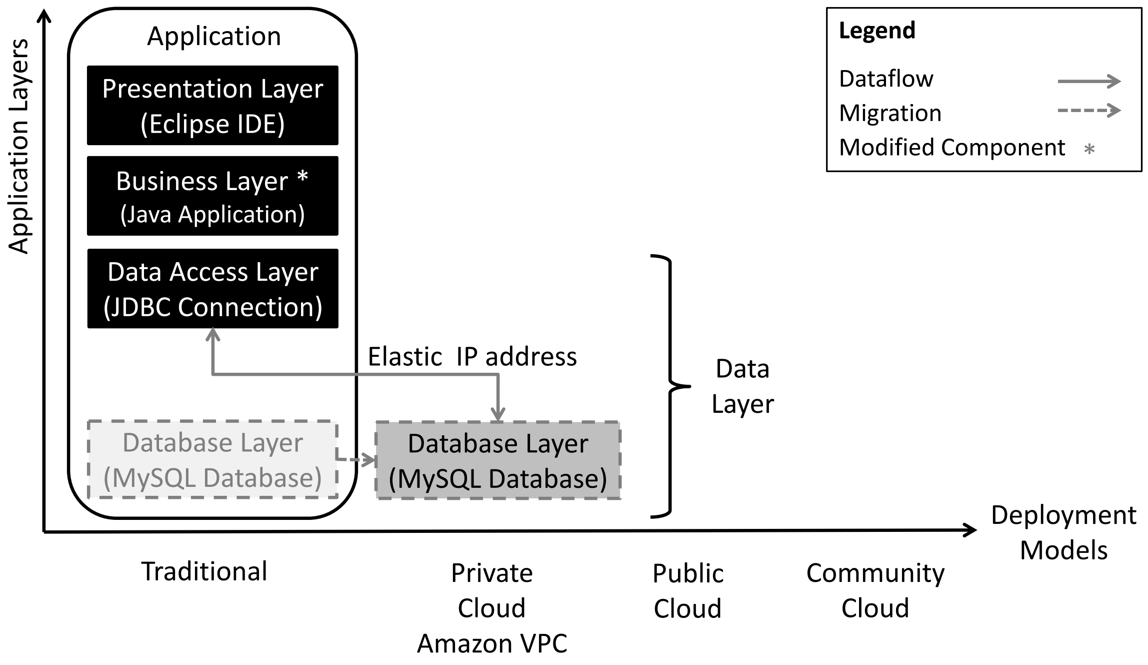


Figure 4.7.: Scenario overview of SQL test cases running on MySQL data store service in Amazon VPC.

VPC Cloud. Then application accesses MySQL database on Amazon VPC through JDBC connection with Amazon VPC public IP address.

Interpretation of Test Results

To setup JUnit testing for MySQL statements testing on Amazon VPC, it is considered using same principles as default MySQL test setting. However, the MySQL database connection string has to be adapted with elastic IP address (host name). After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the expected results with actual results automatically. While running JUnit testing, it is observed that JUnit brings the same outcomes as is in default MySQL testing (see Section 4.3.1). This means that all test cases (SQL statements) are passed on MySQL data store service in Amazon VPC Cloud except MySQL FULL JOIN (see Table A.1). It is also appreciated emulating FULL JOIN properties with UNION operation between LEFT JOIN and RIGHT JOIN of two tables returns the same results. In this scenario an application accessing MySQL data store service on Amazon VPC Cloud, where MySQL database is originally configured on Amazon EC2 and VPC attach Amazon EC2 instance. Furthermore, it is realized SQL statements (i.e. create table, update, delete) are not returning a result set on MySQL database in Amazon VPC Cloud as it has the same behavior in default SQL testing.

4.3.8. Amazon SimpleDB Data Storage

This section describes the test scenario of accessing Amazon SimpleDB data store service from traditional application. In default scenario an application is accessing database on traditional server. By comparing with default application model, the presentation layer, business layer and data access layer are remaining in the same place on local infrastructure; in this scope only database layer is moved to the Cloud however.

While creating Amazon Web Service (AWS) account, it automatically signed up for Amazon SimpleDB data storage in default setting. This mean that it is not required launching an instance like Amazon EC2 or Amazon RDS. Using AWS Access Key and AWS Secret Key it is easily feasible enable accessing data storage service in the Cloud. The database can be implemented in several zones simultaneously if the database size is very large or with a high amount of data tables. With Amazon SimpleDB initially it is possible to allocate a maximum of 250 domains in each account and the size of an individual domain can grow up to 10 GB. However, additional domains can be allocated on request to AWS. If the size of a domain becomes more than the allocating storage, Amazon Simple automatically scale-out architecture and store the data over several domains [SDB].

To setup Amazon SimpleDB data storage service for SQL statements testing, there is a tool called SimpleDB (SDB) Explorer can be used. Using SDB Explorer with AWS Access Key and AWS Secret Key, we can easily migrate and upload openCRX MySQL database on Amazon SimpleDB. However, each data table has to be imported individually. Before importing data into SimpleDB, it is necessary to create a domain manually in a SimpleDB zone. While importing data into a domain the attributes are automatically allocated by corresponding column name from the MySQL data table. The creation of an attribute is missing in case the column contains empty data. However, it is possible to create an attribute manually using SDB Explorer in SimpleDB data storage. During importing a MySQL table in SimpleDB domain, it will offer selecting primary key for the data table. The primary key column converted to index (itemName()) of a domain and the keys must be unique. In openCRX MySQL database, some of the data tables contain same key multiple times, which has to be adapted to import data in SimpleDB data storage. Furthermore, using SDB Explorer tool we can query to the SimpleDB data store service, execute select statement with various operations to retrieve expected results. SimpleDB supported operations are: =, !=, <, > <=, >=, like, not like, between, is null, is not null, and every (keyword)[SDB].

While Amazon SimpleDB allow accessing structured data storage service, the application can take advantage to access data with on-demand storage capacity and facility of using high amount of computations required by the application. To ensure this facility, it is necessary to investigate whether SimpleDB data storage service supports full functionality of SQL-standard as traditional MySQL server does. Therefore, it is considered to run a set of SQL taxonomy based SQL statement test cases on Amazon SimpleDB data store service, to check whether all test cases are passed. In order to test the SimpleDB data store service on Amazon Cloud, the same architecture model is used as default application setting. Figure 4.8 shows a four layer application architecture, where Eclipse IDE represents presentation layer, the Java application running on business layer and MySQL database (database layer) has

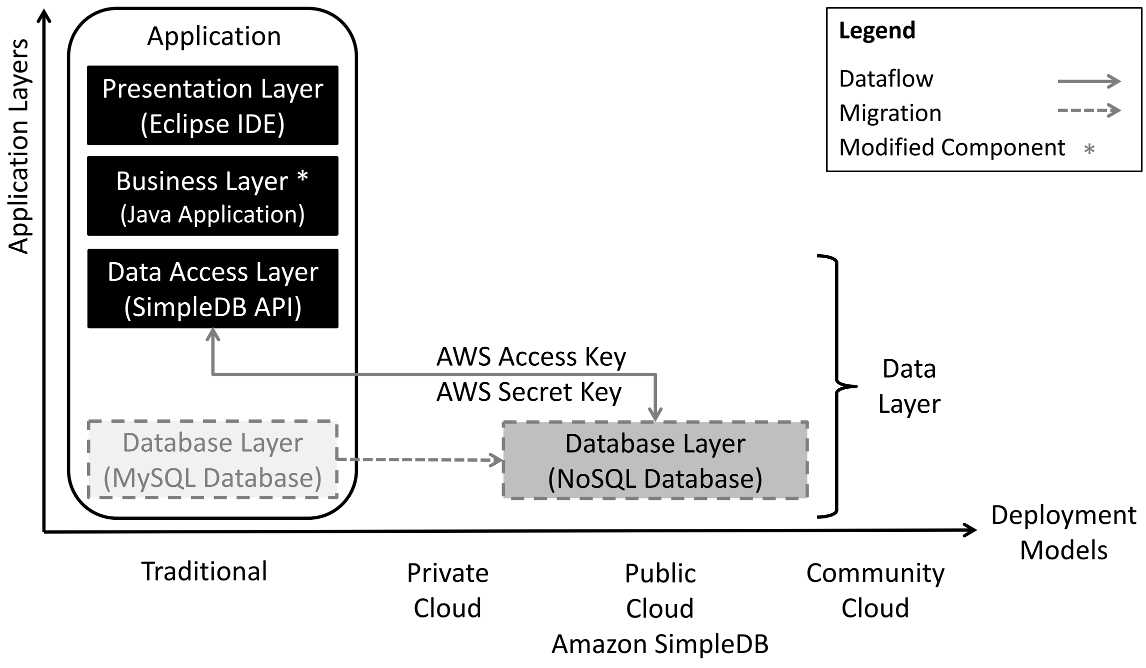

Application Layers
Application
Presentation Layer (Eclipse IDE)
Business Layer * (Java Application)
Data Access Layer (SimpleDB API)
Database Layer (MySQL Database)
AWS Access Key
AWS Secret Key
Database Layer (NoSQL Database)
Data Layer
Legend
Dataflow
Migration
Modified Component *
Deployment Models
Traditional
Private Cloud
Public Cloud Amazon SimpleDB
Community Cloud

Figure 4.8.: Scenario overview of SQL test cases running on Amazon SimpleDB data storage service.

moved to the Cloud (Amazon SimpleDB). Then application enables accessing SimpleDB data storage through either the Simple Object Access Protocol (SOAP) interface or the Query/ReST interface using SimpleDB API call attaching with appropriate AWS Access key and AWS Secret Key.

Interpretation of Test Results

In order to test SQL statements on Amazon SimpleDB, it is consider using JUnit testing framework. Accessing SimpleDB data store service we have to reorganize JUnit testing methods because we have to follow different procedure than default testing settings. In this scenario the Java application connecting SimpleDB through SDB API call. After successfully setting up the JUnit testing methods, we run the JUnit test suite to compare the expected results with actual results automatically. While running JUnit testing, it is observed that all test cases (SQL statements) are initially failed. With the concept of querying to SimpleDB the SELECT statements can be adapted, but still there are some limitations found. However, storing and manipulating data in SimpleDB are differing from traditional database; these can be done by SDB API call. Comparing with RDMS the SimpleDB lacks supporting SQL JOIN, SQL INDEX, and SQL VIEW operations within the domains. However, domains are already indexed, as the domains are index (key) based structured data. The SQL JOIN and SQL VIEW effects are to be implemented in application layer.

Table A.4 shows the JUnit test results while application accessing SimpleDB data storage service in the Cloud. By analyzing JUnit testing results it is determined that all test cases are initially failed because the data model structure of SimpleDB is differing from a RDMS. However, with a slight adaptation of SQL statement the test case numbers 1, 2, 3, 4, 8, 9, 10, 11 and 12 are achieved the expected outcomes. After proper adaptations they are basically returning the same outputs but in different orders. There are several reasons the SQL statements are required adapting, for example while uploading openCRX MySQL database on Amazon SimpleDB the primary key column OBJECT_ID is converted to indexed column `itemName()`, so that OBJECT_ID is no more valid in SimpleDB. Moreover, SimpleDB is only allowing access text type value, so the statement has to be adapted according to the SimpleDB structure.

Test case numbers 5, 6, and 7 are failed because SimpleDB lacks supporting MySQL DISTINCT and AVG features. The effect of SQL DISTINCT feature can be covered in this scope, as `itemName()` keys are unique. The SQL operation "<>" is missing in SimpleDB data accessing, so that the test case 13 is failed. As SimpleDB also lacks supporting SQL VIEW, SQL INDEX, and SQL JOIN features, the test case numbers 14, 15, 16, 24, 25, 28, 29, 30, 31, 32 and 33 are failed. The features are possible to emulate in application layer. Furthermore, the test case numbers 17, 18, 19, 20, 21, 22, 23, 26 and 27 are failed because storing procedure in SimpleDB differing from a traditional database. Structure of data storage changes in SimpleDB have to be done from application layer using SimpleDB API call, as it does not offer executing statements like traditional one.

4.4. SQL Test Summary

This section describes the summery of JUnit testing results, the execution of SQL statement test cases on different Cloud services. Table 4.3 shows the overview of test scenarios, the evolutionary solutions of SQL adaptations of the Data Access Layer (DAL) to enable data access in the Cloud. The table has only represented the required adaptations comparing with default SQL statements. The adaptations are to be used to evaluate the adapter extending for enabling data access in the Cloud. There are 33 SQL test cases are designed for achieving maximum SQL taxonomy.

Table 4.3.: SQL statement test summery.

Test	Default Query/Statement	Adapted Query/Statement
MySQL on Amazon EC2 or RDS or VPC		

33	SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons FULL JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName	SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id
		UNION
		SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id

Oracle on Amazon EC2 or RDS		
07	SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING (access_level_browse > 4)	SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING AVG (access_level_browse * access_level_delete) > 4
15	ALTER VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address, addr.authority AS account_ FROM oocke1_address addr	CREATE OR REPLACE VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address,addr.authority AS account_ FROM oocke1_address addr
20	CREATE TABLE test_table1 (id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, name VARCHAR(250) NOT NULL, age INTEGER UNSIGNED, marks BIGINT UNSIGNED, PRIMARY KEY (id))	CREATE TABLE test_table1 (id NUMBER(10, 0) NOT NULL, name VARCHAR2(250) NOT NULL, age NUMBER(10, 0), marks NUMBER(19, 0), PRIMARY KEY (id))
22	ALTER TABLE test_table3 ADD COLUMN birthday date	ALTER TABLE test_table3 ADD birthday date
25	DROP INDEX idx ON test_table3	DROP INDEX idx
PostgreSQL on Amazon EC2		
07	SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING (access_level_browse > 4)	SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING AVG (access_level_browse * access_level_delete) > 4
15	ALTER VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address, addr.authority AS account_ FROM oocke1_address addr	CREATE OR REPLACE VIEW oocke1_join_accthasassaddr (assigned_address,account) AS SELECT addr.object_id AS assigned_address, addr.authority AS account FROM oocke1_address addr
20	CREATE TABLE test_table1 (id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, name VARCHAR(250) NOT NULL, age INTEGER UNSIGNED, marks BIGINT UNSIGNED, PRIMARY KEY (id))	CREATE TABLE public.test_table1 (id serial NOT NULL, name varchar(250) NOT NULL, age bigint, marks bigint, PRIMARY KEY (id))
25	DROP INDEX idx ON test_table3	DROP INDEX idx
Amazon SimpleDB		
01		
02	SELECT object_id, access_level_browse FROM oocke1_account	SELECT ACCESS_LEVEL_BROWSE FROM oocke1_account
03	SELECT object_id, access_level_browse FROM oocke1_account WHERE object_id='account/CRX/Standard/admin-Standard'	SELECT ACCESS_LEVEL_BROWSE FROM oocke1_account WHERE itemName()='account/CRX/Standard/admin-Standard'
04	SELECT object_id, name FROM oocke1_activitygroup WHERE p$$parent='activities/CRX/Mohsin' ORDER BY object_id ASC, name DESC	The specified query expression syntax is not valid in SimpleDB

Continued on next page

Table 4.3: SQL statement test summery – (continued from previous page)

Test	Default Query/Statement	Adapted Query/Statement
05	SELECT DISTINCT object_id, name FROM oocke1_-calendar	The specified query expression syntax is not valid in SimpleDB
06	SELECT object_id, AVG (access_level_browse * access_level_delete) FROM oocke1_calendar GROUP BY object_id	The specified query expression syntax is not valid in SimpleDB
07	SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING (access_level_browse > 4)	The specified query expression syntax is not valid in SimpleDB
08	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE IDX NOT BETWEEN 0 AND 1	SELECT OWNER FROM oocke1_activityprocess_-WHERE IDX BETWEEN ’0’ AND ’1’
09	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE IDX NOT IN(0, 2)	SELECT OWNER FROM oocke1_activityprocess_-WHERE IDX IN(’0’, ’2’)
10	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE ((IDX > 0) AND (IDX < 2))	SELECT OWNER FROM oocke1_activityprocess_-WHERE ((IDX > ’0’) AND (IDX < ’2’))
11	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE ((IDX < 1) OR (IDX > 1))	SELECT OWNER FROM oocke1_activityprocess_-WHERE ((IDX < ’1’) OR (IDX > ’1’))
12	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE (IDX = 1)	SELECT OWNER FROM oocke1_activityprocess_-WHERE (IDX = ’1’)
13	SELECT object_id, owner FROM oocke1_activityprocess_ WHERE (IDX <> 1)	The specified query expression syntax is not valid in SimpleDB
14	CREATE VIEW oocke1_join_accthasassaddr (assigned_address, account) AS SELECT addr.object_id AS assigned_address, addr.authority AS account FROM oocke1_address addr	The specified query expression syntax is not valid in SimpleDB
15	ALTER VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address, addr.authority AS account_-FROM oocke1_address addr	The specified query expression syntax is not valid in SimpleDB
16	DROP VIEW oocke1_join_accthasassaddr	The specified query expression syntax is not valid in SimpleDB
17	INSERT INTO oocke1_segment (access_level_-browse, dtype, owner_, access_level_update, access_level_delete, object_id) VALUES (4, ’org:opencrx:kernel:account1:Segment’, 2, 3, 1, ’accounts/CRX/Masud’)	Use SimpleDB API
18	UPDATE oocke1_segment SET access_-level_browse=3, owner_=1 WHERE (object_-id=’accounts/CRX/Masud’)	The specified query expression syntax is not valid in SimpleDB
19	DELETE FROM oocke1_segment WHERE (object_-id=’accounts/CRX/Masud’)	Use SimpleDB API
20	CREATE TABLE test_table1 (id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, name VARCHAR(250) NOT NULL, age INTEGER UNSIGNED, marks BIGINT UNSIGNED, PRIMARY KEY (id))	Use SimpleDB API
21	INSERT INTO test_table3 (id, name) SELECT id, name FROM test_table2 WHERE (id=1)	Use SimpleDB API
22	ALTER TABLE test_table3 ADD COLUMN birthday date	Use SimpleDB API
23	ALTER TABLE test_table3 DROP COLUMN birthday	Use SimpleDB API
24	CREATE UNIQUE INDEX idx ON test_table3 (id, name)	The specified query expression syntax is not valid in SimpleDB
25	DROP INDEX idx ON test_table3	The specified query expression syntax is not valid in SimpleDB
26	TRUNCATE TABLE test_table3	Use SimpleDB API

Continued on next page

Table 4.3: SQL statement test summery – (continued from previous page)

Test	Default Query/Statement	Adapted Query/Statement
27	DROP TABLE test_table3	Use SimpleDB API
28	SELECT access_level_browse, full_name FROM oocke1_account INNER JOIN oocke1_account_ ON (oocke1_account.object_id=oocke1_account_.object_-id)	The specified query expression syntax is not valid in SimpleDB
29	SELECT access_level_browse, full_name FROM oocke1_account INNER JOIN oocke1_account_ ON (oocke1_account.object_id=oocke1_account_.object_-id) WHERE (oocke1_account_.idx=1) ORDER BY oocke1_account.full_name	The specified query expression syntax is not valid in SimpleDB
30	SELECT access_level_browse, full_name FROM oocke1_account LEFT JOIN oocke1_account_ ON (oocke1_account.object_id=oocke1_account_.object_-id) WHERE (oocke1_account_.idx=2)	The specified query expression syntax is not valid in SimpleDB
31	SELECT access_level_browse, full_name FROM oocke1_account RIGHT JOIN oocke1_account_ ON (oocke1_account.object_id=oocke1_account_.object_-id) WHERE (oocke1_account_.idx=0)	The specified query expression syntax is not valid in SimpleDB
32	SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons, Orders WHERE ((Persons.P_Id > 1) AND (Orders.P_Id < 3))	The specified query expression syntax is not valid in SimpleDB
33	SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons FULL JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName	The specified query expression syntax is not valid in SimpleDB

Table 4.3 shows the required adaptations for corresponding SQL statements based on SQL taxonomy to access different Cloud data store services by a traditional application. While executing JUnit testing on different database services, it is observed that there are several categories of results can be achieved. Some of the test cases are passed and some of them are failed. However, in failing categories there are three types of decision can be made. For example, some of the queries are actually returning same expected outcomes but in different order, some of them have to be adapted to enable access data storage in the Cloud and some of them are not valid for an individual database service. However, we are concerning about required SQL adaptations in this work. In this table the blank test contains are returning same results in different order and the invalid statements are to be noted.

During testing MySQL data storage service in the Cloud, it is investigated that all test cases are passed except SQL FULL JOIN features which can be adapted using UNION operation between LEFT JOIN and RIGHT JOIN of two tables. Furthermore, it is determined that MySQL database server always present same behavior for different Cloud services, for example, Amazon EC2, Amazon RDS and Amazon VPC. There are 9 test cases are failed while testing Oracle data store service, however 4 of them are showing same outcomes. This mean that it is necessary to adapt 5 SQL statements when an existing application accessing Oracle database in the Cloud. Also, there are 10 test cases are failed while testing PostgreSQL data store service, however only 4 SQL statements has to be adapted in order of enable accessing database service in the Cloud from an existing application.

With Amazon SimpleDB data store service testing it is observed that all SQL test cases are initially failed. Amazon SimpleDB is a NoSQL based non-relational structured data store service and offers only Text type data storing facility. Moreover, Amazon SimpleDB lacks supporting SQL UPDATE, INDEX, VIEW and JOIN features, they can be emulated using SimpleDB API from the application layer however. The storing procedure in SimpleDB is differ than a traditional SQL database. For database changing (i.e. Create, Insert, Update, Delete, etc) it does not support any statement like SQL-standard rule, it has to be done using SimpleDB API. Nevertheless, the SELECT query can be executed as SQL-standard. By analyzing SELECT query results it is realized that some of the queries are returning same results as expected but in different sorted order. However, some of queries are failed because they are lacks supporting features, for example, DISTINCT, <>, AVG, etc.

5. Concept and Specification

This section describes the systems requirements for adapting a Data Access Layer (DAL) within an application architecture model to enable Cloud data access. Also, it explains how future applications architecture should look like and how to design a complete application system, which can take advantages of accessing Cloud computing resources. Previous chapter has described the evaluation of SQL support of Cloud data services based on SQL statement testing. With the outcomes of the SQL statements testing the required adaptations has been determined. Moreover, the concept of adapting a data access layer to enable Cloud data access is also depicted in this chapter. The key requirements including Cloud-enabled application design as well as the necessity of implementing a data access layer is given in Section 5.1. The Section 5.2 described more detail functional requirements including a use case analysis. Moreover, this chapter also illustrates the non-functional requirements (see in Section 5.4) for giving guidance values for some software qualities.

5.1. System Overview

A Data Access Layer (DAL) has to be extended with implementing required SQL adaptations within a layering application architecture model that allows typical application to access Cloud based data store services. The DAL is responsible for encapsulating the data access functionalities and interacting with the business logic. However, the application has to define the connection configuration explicitly in order to access a particular Cloud data storage. There are different types of Cloud services currently running over the online. According to the infrastructure of Cloud services some are accessible directly using a public URL address with no authentication required (i.e. Amazon RDS), but some has to be accessed using a public key authentication (i.e. Amazon EC2). Therefore, the data access layer has to be implemented considering accordingly. Moreover, to access a particular Cloud data store service the data access layer also has to be adapted with the required SQL statement adaptations. While application send a database query to the data access layer, the data access layer adapt the query with prior knowledge of SQL adaptation and then send it to the specific Cloud service performing SQL execution.

For evaluating the adaptation of data access layer within an application system, a Java Graphical User Interface (GUI) application has to be developed. It is assumes that the application is running on local server and accessing the data and computation from Cloud data store services on-demand through DAL. However, with this approach there are several possible scenarios may be realized for an existing application.

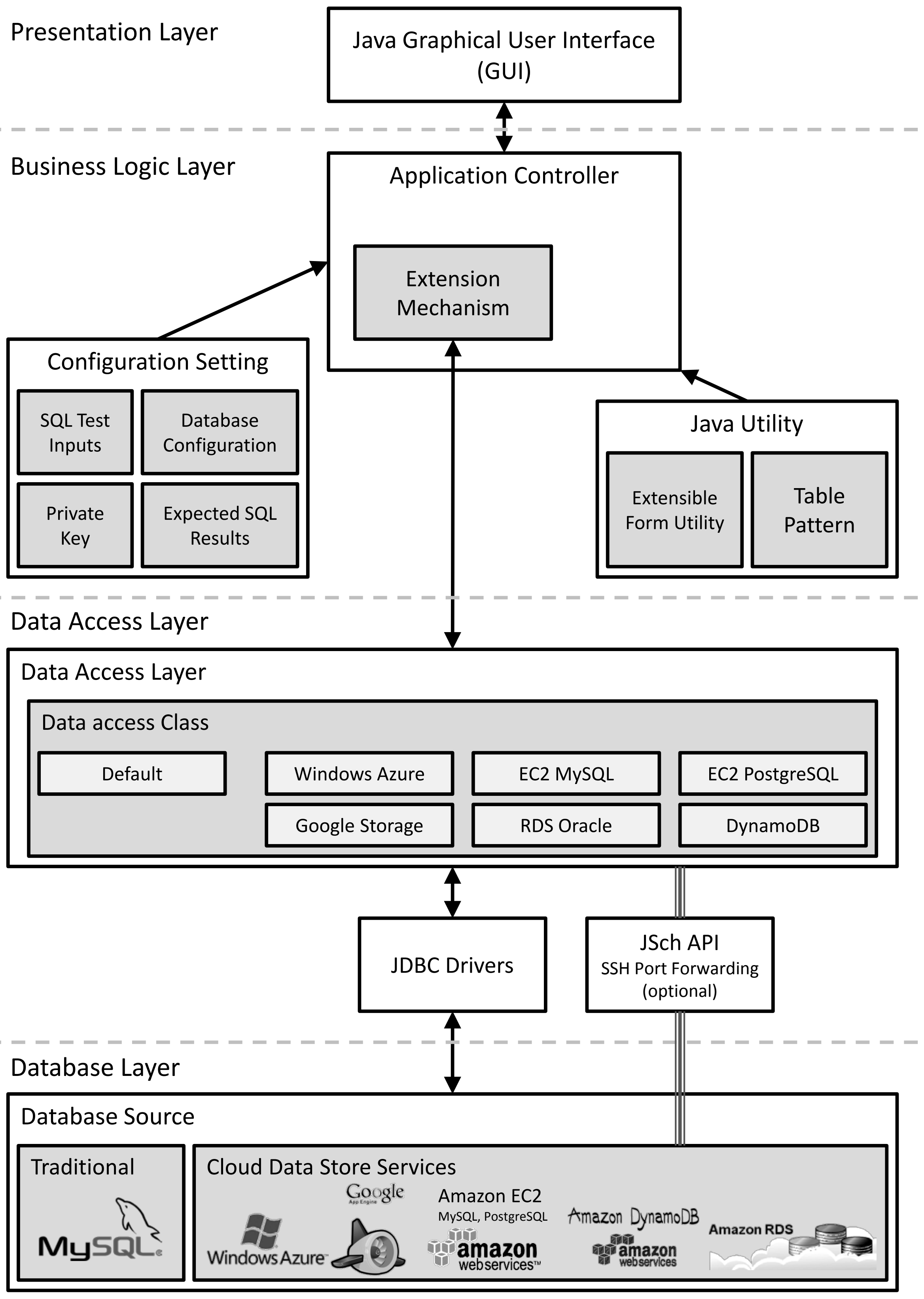


Figure 5.1.: Layered based application architecture model.

For example, if an application accessing a traditional database and the data is increasing continuously, the database server might not able to handle the high amount of data. In this situation the database layer may completely or partially outsourced to the Cloud to easily scale the database layer. Thus will reduce the capital expenditure and maintenance cost for an enterprise IT, as application is still using the current infrastructure and use the Cloud resources only whenever it is required to be used. Another possible example scenario would be an application may move the traditional database to the Cloud and use only Cloud data store services for managing the high volume of data and growing amount of computations.

A layered based application architecture model is shown in Figure 5.1, where application represents a four layer architecture model including a presentation layer, a business logic layer, a data access layer, and a database layer. In this architecture model, the user interface (SQL Evaluation tool) is representing a presentation layer and responsible for showing tool's activities how data access layer performing encapsulation of data access functionalities. Application controller is the core of tooling system, which has to be implemented with a Java extension mechanism in order to extend data access functionality.

In this master's thesis, we focus on adapting a data access layer that typical application can access both traditional database as well as various Cloud data store services. However, the main target is to be able to access Cloud data store services through the data access layer. Moreover, in this scope, for every Cloud services the access layer has to be designed and deployed with specific data access configuration. The configuration for a Cloud data store service to the data access layer may not interfere with the configuration of other Cloud data store services. Furthermore, an acceptable data access has to be ensured by the data access layer. In order to enable accessing Cloud data store services, usually the application would have using JDBC drivers. According to the Cloud networking infrastructure in some cases the application may require establishing a SSH tunneling with specific database port forwarding to access data store services such as Amazon EC2.

5.2. Functional Requirement

This section describes the detail functional requirements for the SQL Evaluation Tool application systems including and use case analysis. The application extension mechanism describes how application can be facilitated using Java plugins mechanism in order to enable access different Cloud data store services by uploading provider classes in runtime. Moreover, the application has to be ensured accessing heterogeneous Cloud data to enhance the application performance and reliable data computations.

5.2.1. Heterogeneous Data Access

As it is intended to access different Cloud data store services for the evaluation of adaptation of data access layer, the application has to be ensured accessing heterogeneous data in the Cloud. It is observed that each Cloud service has implemented with unique design philosophy and individual purposes. Also, each database service has established with unique characteristics (i.e. data model architecture) and exclusive goal of computations supports. More importantly, data and data structures are differing while comparing with different database systems. Moreover, it is realized that some databases are suitable for transactional data computations and some are appropriate for non-transactional data computations. For example, PostgreSQL is fully Atomicity, Consistency, Isolation and Durability (ACID)-complaint and has the highest priority of reliability, data consistency, and data integrity supports [Man11], whereas MySQL is not fully ACID-complaint but more flexible to database handling than PostgreSQL. Another example is, both MySQL and Oracle provide supports for big data analysis. However, the performance of big data analysis in Oracle is more powerful than MySQL database system. In this case, in order to analyze the big data, the application can particularly take advantage of accessing faster big data computation in Oracle data store service.

Furthermore, it is studied that using Amazon S3 Alaxa Web services [Ama12] stores and delivers a millions of thumbnail images and then use SimpleDB to perform automatic indexing and efficient query to the accumulated images, where SimpleDB performs very faster computations comparing with a traditional database like MySQL, PostgreSQL or even Oracle database. From this above example the application may use SimpleDB data store service when application is needed to be managed a high volume of data objects. From all above examples, it can be easily determined that different data store services are appropriate for different types of data computations. Therefore, now application has to decide which Cloud data store services are to be used for what computation purposes.

5.2.2. Application Extension Mechanism

As a general view, using Java extension mechanism (i.e. plugins mechanism) a Java application can be enhanced its application architecture as well as the functionalities in runtime. An example scenario of Java extension mechanism is shown in Figure 5.2, where extensions act as "add-on" modules to the Java platform. In this application system it is consider being implemented Java application extension mechanism in order to enable accessing the several data access functionalities in application running time.

The specific reason of implementing extension mechanism is that application may allow to include a new Cloud data store service when it necessary to perform particular computations. For example, a Web application has to be running 24x7 hours in a week and can not be interrupted their service to extend a particular functionality. In this scenario, especially if the application is needed to be extended the data access functionality in order to access a new Cloud data store service, using extension mechanism the application can automatically enhance the application architecture (e.g. loading a new provider class) runtime without interrupting their service providing.

Furthermore, the concept is that if the data access functionality is updated by the administrator, the application may reload the updated provider classes upon notifying the updates information. Nevertheless, to ensure these runtime extension facilities within *SQLEvaluationTool* application system, a Java plugins mechanism (i.e. ClassLoader API) has to be

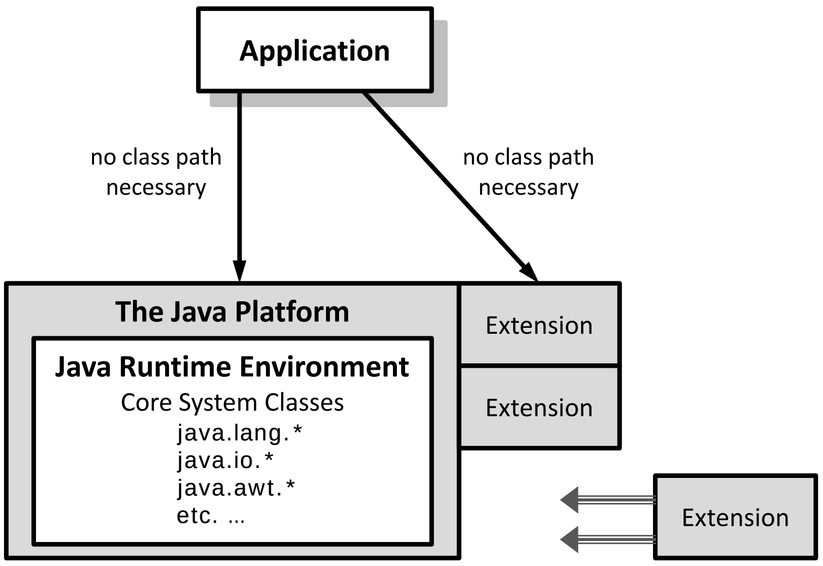


Figure 5.2.: An overview of Java application extension mechanism, where extensions are acting as "add-on" modules to the Java platform. [Som].

implemented, as it is planned to enhance the data access functionalities uploading individual provider class for different Cloud data access. Additionally, as the data access layer is implemented with a bundle of provider classes, the user may choose a number of provider classes to be performed evaluating the SQL adaptations simultaneously in different Cloud data store services.

5.2.3. SSH Connection to Cloud Database

According to the Cloud computing network infrastructure some of the data store services (i.e. MySQL and PostgreSQL on Amazon EC2) are not directly accessible through JDBC drivers form a traditional application. Every Amazon EC2 account provides authentication key, which can be used to connect o the Amazon EC2 Cloud the traditional server or application. It is observed that there are several ways can be followed to establish connection to the Amazon EC2 Cloud database, such as over SSH tunneling or RDP (Remote Desktop Protocol) connection. In order to enable Cloud data access on Amazon EC2, in this system it is consider using SSH tunneling over Transmission Control Protocol (TCP) port 22. To accomplish the SSH tunneling with public key authentication, a Java API called Java Secure Channel (JSch) [JCr] can be used. While establishing a SSH tunneling it is possible forwarding a local port (with -L option) of the target database. Then the target database (i.e. MySQL on Amazon EC2) becomes accessible over the forwarded port through JDBC driver for a traditional application.

5.3. Use Cases

The basic use case illustrating the adaptation of SQL integration for a data access layer is described in following. In this use case analysis an actor, a user takes part.

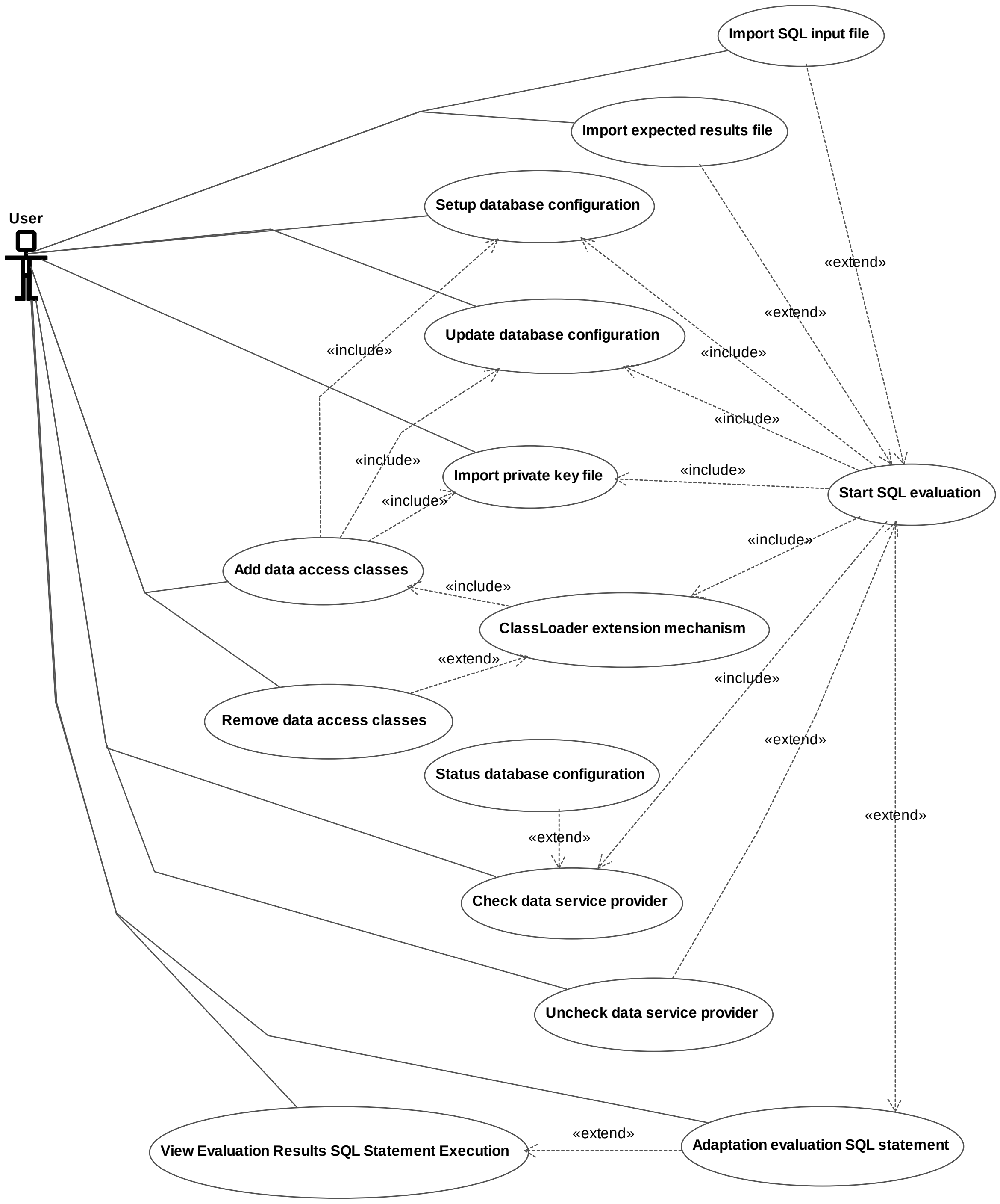
Name	**View Evaluation Results SQL Statement Execution**
Goal	The user wants to review the comparison results of SQL executing in different Cloud database services.
Actor	User
Pre-Condition	The tool launching without prior knowledge of necessary database configuration, a set of SQL statements inputs and expected SQL results, and the target Cloud data store services. Additionally, it shows an example evolutionary results table in an extensible form layout. While starting the execution of SQL statements, the necessary results tables will be drawn dynamically based on checked data service providers, where execution results are shown. Also, it appears another extensible form for representing the chosen data service providers, which is initially empty however.
Post-Condition	The SQL execution results are shown in SQL evaluation results tables by comparing with expected and actual results. And also comment on whether an execution is passed or failed.
Post-Condition in Special Case	JDBC driver returns database connection error for one or more target data store services, then the user fix the necessary configuration and restart the SQL execution for achieving acceptable results.

Normal Case

1. User configures the database connection (i.e. add or update database configuration file) and selects necessary private keys for specific Cloud services.
2. User selects SQL statements input file, which contains a set of selected SQL statements.
   a) User selects a wrong SQL statement file.
   b) The system indicates file loading error; the file can't be loaded in the system.
3. User selects expected SQL results for corresponding SQL statements inputs.
   a) User selects a wrong SQL Results file.
   b) The system indicates file loading error; the file can't be loaded in the system.
4. User selects the target data store services (data access classes) for enhancing data access functionalities within the system.
5. The system extends the application architecture by loading data access classes in extension mechanism.
6. The system represents the checkboxes for every selected data access classes.
   a) User check the checkboxes of data service providers in Cloud providers list, the checked data store services will be evaluated.
7. The System automatically checks the target databases connections.
   a) The system indicates database connection error, in case any target database connection is invalid.
   b) User fixes the database connection error.
8. User starts SQL execution with checked Cloud data store services by pressing on *Start Eval* button.
9. The system asks the user to confirm the necessary configurations in order to start SQL execution.
10. The user confirms.
11. The system removes all previous SQL results tables from the form layout and dynamically creates new tables for checked data store services.
12. The System starts SQL execution based on checked Cloud data service providers by creating multiple process threads.
13. The system adds SQL execution results in the evaluation results tables in real-time.
14. The system shows the progress of execution process in process bar for every checked data store services in percentage value.
15. The system finishes individual SQL execution and then disappears related process bar.

Special Cases	5a. User wants to remove one or more data store service providers (i.e. data access class) from the providers list. a) User checks one or more the service providers in the providers list. b) User clicks on remove button. c) The system removes the checkboxes from the provider list. d) The system deletes selected data access classes from the application extension mechanism. 12a. One or more data store services stopped or Cloud instance is terminated. a) The system indicates the data store services connection error. b) User fixed the database configuration setup. c) User restarts the SQL execution (repeat from the step 8 in normal case).

Table 5.1.: Description of Use Case *View Evaluation Results SQL Statement Execution*.



User
Import SQL input file
Import expected results file
Setup database configuration
Update database configuration
Import private key file
Start SQL evaluation
Add data access classes
ClassLoader extension mechanism
Remove data access classes
Status database configuration
Check data service provider
Uncheck data service provider
View Evaluation Results SQL Statement Execution
Adaptation evaluation SQL statement
«include»
«extend»

Figure 5.3.: Use case diagram.

5.4. Non-Functional Requirements

This section describes the non-functional requirements that the application systems should satisfy with several categories of non-behavioral functionalities. Nevertheless, in this system, the non-functional requirements are ensuring the quality of the software system. Therefore, to ensure the detail and sufficient functional requirements of the application system, this systems shall include several categories of non-functional requirements such as extensibility, reusability, maintainability, and installation ease. The following subsections list these requirements in the specific category and explain them in detail.

5.4.1. Extensibility

In order to enhance the functionality of data access layer (i.e. add a new data store service), the application system has to be implemented with Java extension mechanism. Implementing an extension mechanism the application system then ensures the ability to extend the system in runtime. Then the application can easily add a new data access class into the system to enable accessing Cloud data store service. Moreover, using extension facility the application also can remove data access functionality while it is not needed any more. Thus minimize the complexity of Cloud data access for the application system.

5.4.2. Reusability

The code segment of DAL class can be reused for adding a new Cloud database service with slight modification. The basic class architecture should be same, only may necessary to add the required SQL adaptation in order to access a specific data store service. The classes can be reused, because of the individual data access classes are designed for different specific Cloud data store services, where DAL's classes are responsible for only encapsulating the data access functionalities, but not correcting or modifying the data. In this system, usually DAL classes are receive SQL statement from the application controller and execute them into the target data store service and then send back the achieving SQL execution results to the application controller. From this concept of view, if another application wants to access the same cloud data store service implementing a data access layer, they can reuse the DAL classes without modification what is designed for this system. Furthermore, for representing the SQL comparison results this system creates the same evaluation result tables for all checked data store services. so that, the code segment of generating tables are to be reused for every tables.

5.4.3. Maintainability

The source code has to be well documented and decoupling of system components has to be facilitating changes in functionalities. In this system, the different components are to be designed based on layered application architecture model, where components are working relatively independent. Thus makes the software easy maintainable with reducing a lot of efforts to implement or modify new functionality.

Moreover, implementing extension mechanism make the system more maintainable, as individual provider classes can be loaded in runtime to enable a specific Cloud data access. In order to add a new data store service, the developer has to add only a new provider class and update the data store service configuration in the configuration file, and then application controller can load the new class into the system to access enable Cloud data store service.

5.4.4. Installation Ease

The installation procedure of SQL Evaluation tooling system has to be well documented step by step in detail. Accomplishing all steps should lead to a running system on both Windows and Linux platform.

6. Design

This chapter describes an architectural design including a class diagram and technological solutions for the concepts and specified system requirements in Chapter 5. First an overall system overview of SQL Evaluation tool is given in Section 5.1 and then the various used technologies required by the system are illustrated in separated sections.

6.1. Architectural Overview

The overall system architecture of SQL Evaluation Tool is shown in Figure 6.1, where a Java Graphical User Interface (GUI) application represents the general activities of the adaptation of data access layer and the evolutionary result of SQL statement adaptations. According to the layered based application architecture model, the application user interface (SQL Evaluation Tool) is considered a presentation layer. In more comprehensive view, the tool represents the evolutionary result of accessing different Cloud data store services simultaneously with implementing multi-threaded facility. Moreover, to evaluate the adaptation of SQL statements this tool gives user to choose a set of selected SQL statements from a SQL file and also it provides the functionality to load the data access classes dynamically in runtime. The data access classes contain the data access functionality and the required SQL adaptation in order to access selected Cloud data store services. However, for accessing each data store services it has to be designed with individual data access classes.

According to the application architecture model, the three main components consisting of configuration setting, application controller, and Java utility are holding in business logic layer together. The application interface is directly connected to the application controller, where application controller is representing the business logic functionalities and responsible for interacting with user interface as well as the data access functionalities within the SQL Evaluation tool system. Nevertheless, the application controller is connected with several components including database configuration, selected SQL inputs and expected SQL results, Java utility classes, and private key file. Additionally, within the application controller it is considered implementing Java extension mechanism for loading a number of data access classes in runtime to extend the application architecture and data access functionalities. During running the application, it is also consider loading a database configuration for different Cloud data store services by using XML Document Object Model (DOM) parsing, as necessary database configuration is stored in a XML file format. Moreover, the tool offers the facility to update the database configuration in application running time. Once database configuration file is updated the application can use the updated configuration.

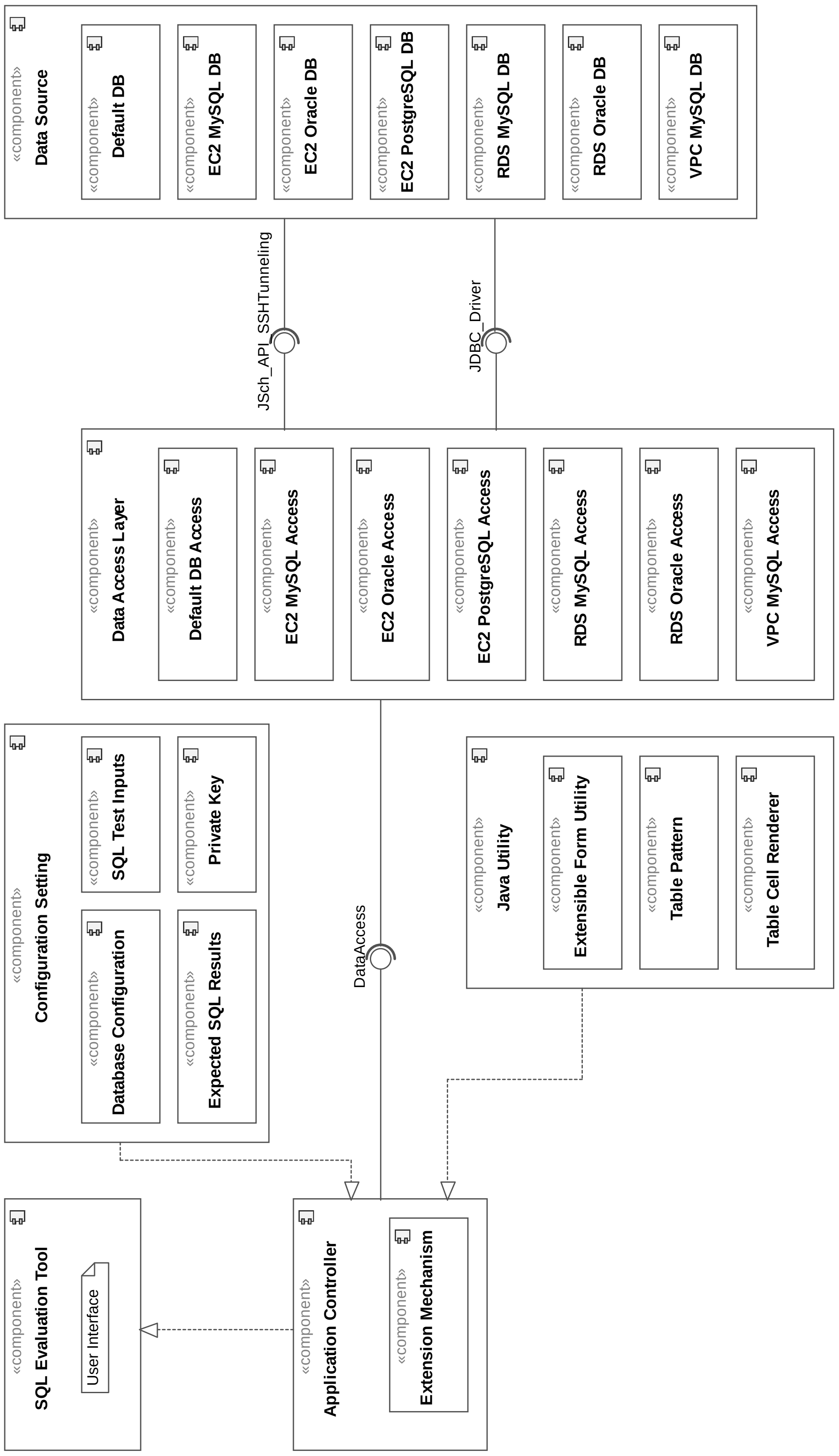

«component» Data Source
«component» Default DB
«component» EC2 MySQL DB
«component» EC2 Oracle DB
«component» EC2 PostgreSQL DB
«component» RDS MySQL DB
«component» RDS Oracle DB
«component» VPC MySQL DB
JSch_API_SSHTunneling
JDBC_Driver
«component» Data Access Layer
«component» Default DB Access
«component» EC2 MySQL Access
«component» EC2 Oracle Access
«component» EC2 PostgreSQL Access
«component» RDS MySQL Access
«component» RDS Oracle Access
«component» VPC MySQL Access
DataAccess
«component» Configuration Setting
«component» SQL Test Inputs
«component» Private Key
«component» Database Configuration
«component» Expected SQL Results
«component» Java Utility
«component» Extensible Form Utility
«component» Table Pattern
«component» Table Cell Renderer
«component» SQL Evaluation Tool
User Interface
«component» Application Controller
«component» Extension Mechanism

Figure 6.1.: Component diagram of SQL Evaluation Tool

Using this facility the user can add or remove any of the data store configuration without interrupting the running application.

In this scope, the database configuration has to be designed with detail provider connection information as well as the information of specific data store service connections. Then, using knowledge of provider's connection the application controller can easily determine routing the SQL statement execution to a particular data store service. Furthermore, performing the SQL statement execution in Cloud data store services and compare the SQL result sets, it is considered uploading a set of selected SQL test input statements and the corresponding SQL results. In this case the result sets are prepared based on default database. Additionally, representing SQL statement evaluation results the tool has to be implemented dynamic and extensible Frame utility, where components can be added or removed dynamically on application running time. For example, user wants to evaluate N number of data store services then he has to choose N number of data access classes. As a result N number of checkboxes will be added in the frame utility and based on checked checkboxes the application will draw the tables to represent the comparison results. Ensuring this facility, a Java form utility (i.e. *GridBagConstraints*) is considered to be implemented and representing the multi-lined table structure, a Java utility called TableCellRenderer is consider implementing with the application controller.

The next component of the application system is a data access layer, which is responsible for encapsulating the data access functionalities for access both traditional database as well as Cloud data store services. In data access layer, it is considered including a list of data access classes corresponding individual data store services. The required SQL statement adaptations are to be stored in these data access classes. While data access layer is received a SQL query request from the application controller, it will investigate whether this statement needs to be adapted or not. If required to be adapted, the data access layer adapts the SQL statement based on adaptation knowledge and send the adapted statement to the target data store service for performing correct SQL execution.

To reduce the application complexity, initially the data access layer is not connected to the application logic. It is assumed that the data access layer will be loaded in runtime of the application system. For enhancing the functionality of application system, a Java extension mechanism (i.e. ClassLoader) will be implemented within application controller. The concept is that in order to access different Cloud data store services the application will use different class interfaces. In this system accessing both traditional database and Cloud data store services, the data access layer usually use JDBC drivers, as currently most of the popular database providers provide the JDBC drivers. Nevertheless, it is observed that according to the Cloud networking infrastructure, some data store services cannot be accessed directly through JDBC driver (i.e. Amazon EC2 Cloud), as they require public key authentication in order to enable Cloud data access.

In this scenario the data access layer may establish a SSH (Secure Shell) communication over port 22 to the target Cloud service with specific database port forwarding. Upon establishing a SSH tunneling the data access layer now can access Cloud data through JDBC drivers using forwarded database port instead of using a local database port. For creating SSH tunneling session a JSch (Java Secure Channel) API is considered to be used in this system, as JSch provides supports for secure remote login with public key authentication as well as the port forwarding facilities. Nevertheless, this tool only represents the enable accessing of Cloud data store services, but it does not ensure the security and performance of Cloud data access, as promising the performance of Cloud data access is out of scope of this master's thesis.

The top down component is the data source within the application system. To evaluate the SQL adaptations, several database services including traditional as well as Cloud data store services has been considered. The aim of adapting a data access layer is that, application can take advantages of accessing Cloud resources including data and computations, while traditional server is not sufficient to perform the necessary computations. However, in this scope using data access layer, application can access both traditional and Cloud data store services depending on their computations needed. The default SQL test inputs are design based on traditional MySQL database. For performing the SQL evaluations, these test inputs are considered to be executed on various RDMS Cloud storage services including MySQL, Oracle, and PostgreSQL on Amazon EC2, MySQL and Oracle on Amazon RDS, and MySQL on Amazon VPC Cloud. Furthermore, we have investigated Amazon SimpleDB, a NoSQL data store service for the evaluation of SQL statements, the test outcomes are described in Chapter 4. As we are focusing on accessing RDMS Cloud storage services for now, in the design of data access layer the data access from Amazon SimpleDB has been omitted. However, in the future work we will implement accessing NoSQL data store services from traditional application.

6.2. Class Diagram

This section gives an overview of the classes of *SQL Evaluation tool* system, which is to be implemented in Java platform. We will summarize the usages of most essential one and provide a short explanation of attributes and methods. Figure 6.2 shows a detail conceptual relationship diagram for the system. The system is designed based on component based software systems, where each component is designed with several classes. Then all components are merged together to build a complete system.

6.2.1. EvaluationWindow

The `EvaluationWindow` is the main class for creating a graphical tool window in the system, which provides an interface for the application layer. This class is responsible for representing and controlling all User Interface (UI) elements related actions. The methods of this class are non-blocking, as different data store services have to be accessed simultaneously independently, where one execution may not wait for another execution to be finished. In order to perform desire computations, it has associated with several significant classes within the system.

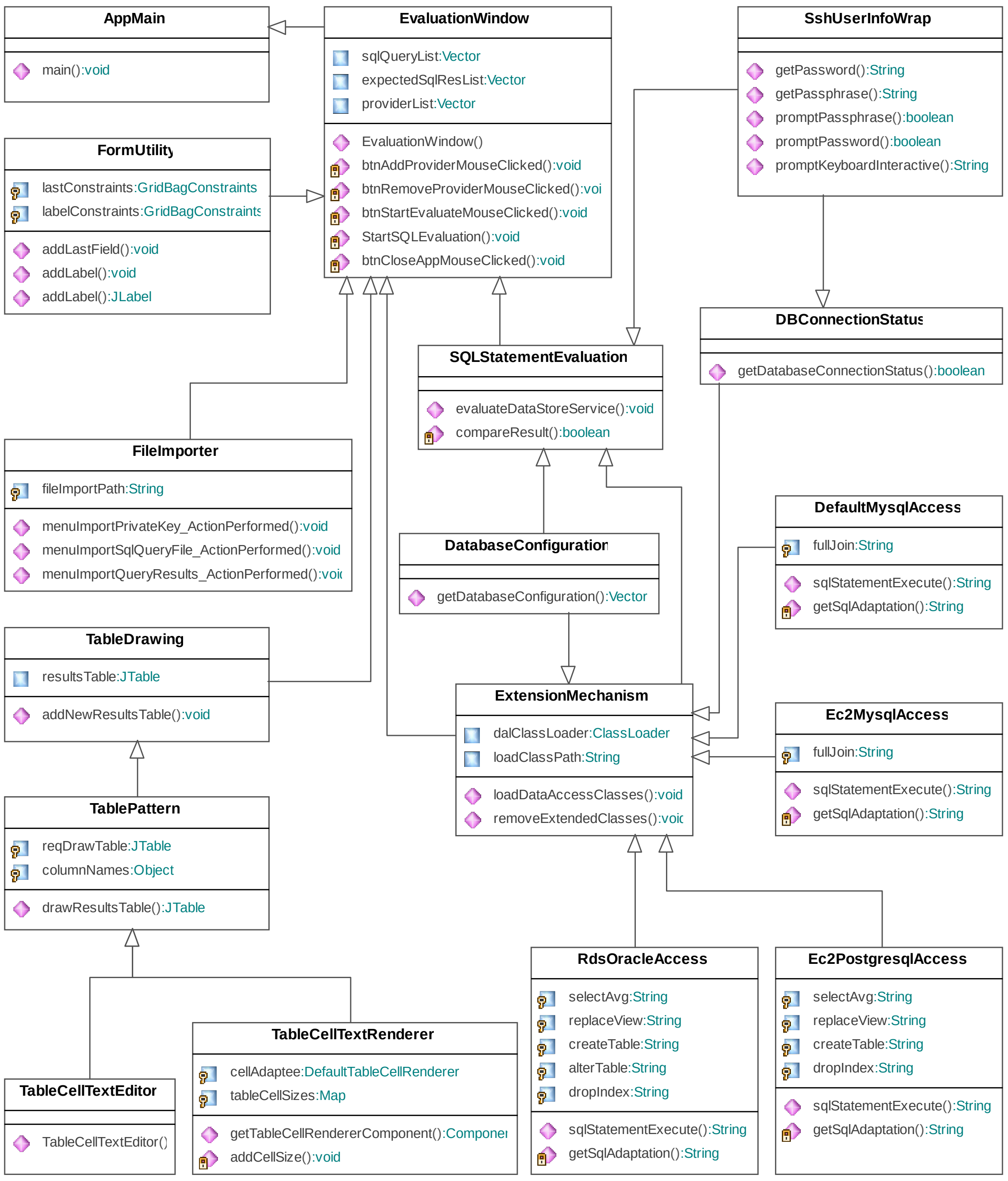

AppMain
main():void
EvaluationWindow
sqlQueryList:Vector
expectedSqlResList:Vector
providerList:Vector
EvaluationWindow()
btnAddProviderMouseClicked():void
btnRemoveProviderMouseClicked():voi
btnStartEvaluateMouseClicked():void
StartSQLEvaluation():void
btnCloseAppMouseClicked():void
SshUserInfoWrap
getPassword():String
getPassphrase():String
promptPassphrase():boolean
promptPassword():boolean
promptKeyboardInteractive():String
FormUtility
lastConstraints:GridBagConstraints
labelConstraints:GridBagConstraints
addLastField():void
addLabel():void
addLabel():JLabel
DBConnectionStatus
getDatabaseConnectionStatus():boolean
SQLStatementEvaluation
evaluateDataStoreService():void
compareResult():boolean
FileImporter
fileImportPath:String
menuImportPrivateKey_ActionPerformed():void
menuImportSqlQueryFile_ActionPerformed():void
menuImportQueryResults_ActionPerformed():voi
DefaultMysqlAccess
fullJoin:String
sqlStatementExecute():String
getSqlAdaptation():String
DatabaseConfiguration
getDatabaseConfiguration():Vector
TableDrawing
resultsTable:JTable
addNewResultsTable():void
ExtensionMechanism
dalClassLoader:ClassLoader
loadClassPath:String
loadDataAccessClasses():void
removeExtendedClasses():voic
Ec2MysqlAccess
fullJoin:String
sqlStatementExecute():String
getSqlAdaptation():String
TablePattern
reqDrawTable:JTable
columnNames:Object
drawResultsTable():JTable
RdsOracleAccess
selectAvg:String
replaceView:String
createTable:String
alterTable:String
dropIndex:String
sqlStatementExecute():String
getSqlAdaptation():String
Ec2PostgresqlAccess
selectAvg:String
replaceView:String
createTable:String
dropIndex:String
sqlStatementExecute():String
getSqlAdaptation():String
TableCellTextRenderer
cellAdaptee:DefaultTableCellRenderer
tableCellSizes:Map
getTableCellRendererComponent():Componen
addCellSize():void
TableCellTextEditor
TableCellTextEditor()

Figure 6.2.: Class diagram.

In explicit view, this class is to be considered as an application controller, as it is controlling the loading of data access layer classes to the extension mechanism and gives instruction for the SQL statement execution. Moreover, it holds the information about loaded data access classes in the system.

- In class `EvaluationWindow`, several attributes are to be defined. However, the three main attributes are `SqlQueryList`, `expectedSqlResList`, and `providerList`. The `Sql-QueryList` attribute is used to store a set of defined SQL statement inputs, which have to be executed evaluating the Cloud data store service. Similarly, the `expectedSqlRes-List` attribute stores the corresponding expected SQL results. These expected results are to be used for comparing with actual SQL executed results, while SQL statements are executing in different Cloud data store services. Moreover, the `providerList` attribute is employing to keep the updated information about loaded data access classes.
- While running the application, the `EvaluationWindow` class constructs a graphical SQL Evaluation tool window, where the tool has to be implemented two internal Frames associated with extensible form layout. One is for adding data service provider's identity and another one is for representing the SQL evaluation results. Both Frames have to be capable organizing the UI components dynamically. It is observed that In order to draw an extensible form a Java form utility (i.e. `GridBagConstraints`) may be used. Moreover, the tool has also included several buttons such as *Add Provider*, *Remove*, *Start Eval*, and *Close App* to perform different actions. The Add Provider button provides the functionality to add one or more data access classes in the system, where Remove button removes one or more data access classes from the system. Furthermore, the Start Eval button is to be used to start the SQL statement executing in different selected data store services.
- While user wants to add a new data access class to the system and clicked on Add Provider button, the method `btnAddProviderMouseClick()` is called to perform the action, such as add a data access class to the extension mechanism. This method will open a Java native file chooser window to add one or more data access classes. Also, it will add the data service provider's name including a checkbox to the provider list. Moreover, while adding data access classes to the extension mechanism, this method checks the data access connections status to the target data store services with the help of `DBConnectionStatus` class. Furthermore, this method can also be used for updating the added data access classes in the system.
- When a data access class needs to be removed from the system, the `btnRemoveProvider-MouseClicked()` method is to be used. With clicking on Remove button, this method is called to remove the data access classes from the system and also remove the data service provider name from the provider list.
- Configuring the tool setup properly, importing all necessary configuration file (i.e. a set of SQL statement inputs, corresponding expected SQL results, and required private keys), now it is time to start the SQL execution evaluation. The `btnStartEvalulate-MouseClicked()` method is responsible for starting the SQL execution preparations in different Cloud data store services based on checked data service providers. Therefore,

clicking on Start Eval button this method will be associated to start the SQL execution evaluation preparation by calling `startSQLEvaluation()` method, as this method is associated with Start Eval button.

- As it is planed to run the SQL statements simultaneously, the `startSQLEvaluation()` method creates individual threads for every Cloud data accesses based on checked data service providers. Then multiple threads will be run together simultaneously while several data store services have to be accessed. Implementing multi-threaded facility, the tool now can execute the SQL statements to the explicit Cloud data store services independently. Therefore, one execution will not interfere to the other executions.
- Upon finishing the SQL statement evaluation the application can be exited using `btnCloseAppMouseClicked()` method, as the method implemented an action listener for Close App button.

6.2.2. DatabaseConfiguration and DBConnectionStatus

As data store service configuration setting is important for Cloud data accesses and can be changed over time while updating the database setup in the Cloud, the tool provides opportunity for user updating the database configuration in the system in runtime. The `getDatabaseConfiguration()` method is used to retrieve both traditional database as well as the Cloud data store service access configuration. The necessary configurations are stored in `dbconfig.xml` file. Using DOM parser this method extracts the required configuration for a specific Cloud data store service and returns the configuration information. Additionally, if the database configuration is required to be updated in the system, the user will update the configuration information in `dbconfig.xml` and then the system will automatically use the updated configuration in runtime.

The `DBConnectionStatus` class is responsible for confirming the user about Cloud data access connection status. When adding a data access class to the system, the `getDatabaseConnectionStatus()` method informs the database connection status to the system. If the data access is possible to establish the method returns success confirmation, otherwise it returns unsuccessful. Using this method, user can determine about the data access configuration setup before start evaluating the SQL statement execution in the Cloud.

6.2.3. FileImporter and SshUserInfoWrap

In order to evaluate the SQL statement execution in the Cloud, a set of SQL statement inputs as well as the corresponding SQL results have to be given in the system. Using `FileImporter` class, the tool can load the selected SQL inputs and the corresponding results. Moreover, some Cloud infrastructures are required public key authentication accessing Cloud data store services, this class also provides method to load the necessary private keys for performing user authentications.

The SshUserInfoWrap is an interface and a wrapper class, which is used to establish SSH tunneling by authenticating appropriate private key using JSch API. This wrapper is responsible for validating the user authentication while a secure channel is required to be established for Cloud data access.

6.2.4. FormUtility and TableDrawing

The FormUtility class is a simple wrapper class for creating extensible form layout applying GridBagConstrains objects. This class hides the details of using GridBagLayout for the GUI development tasks and capable to add several types of components including label, checkbox, and table in the form [Ise06]. Using this FormUtility wrapper, it makes feasible to create the GUI elements dynamically according the need. For example, representing the evaluation results of SQL statement execution in different Cloud data store services, this system has considered drawing individual tables for every checked data service providers. Using this class, it makes feasible to add a dynamic number of tables in a form for individual Cloud services. Moreover, adding a component to the FormUtility form, it will not necessary to explicitly specify the relative positions and the sizing semantics, as the details of each added component's size and position are controlling by a GridBagConstrains object associated with each component.

As it is considered representing the individual comparison results in a different table, the TableDrawing class is responsible for creating a new table for every selected data service providers. It is realized that every table has to be created with same number of rows and columns for each data service provider, as the same set of SQL statements are to be executed in every data store services. Therefore, using addNewResultsTable method of this class, it can be added any number of tables. In addition, the number of rows of a table is to be determined by the number of SQL statements inputs size. The numbers of columns are fixed, as it will represent the similar results for every SQL execution such as Expected results, Actual results and outcomes (pass or fail comments). As the size of the tables is always same, a template table may be created. The number of rows of the template is equal to the number of SQL statements inputs size. Then the same template can be used for every comparison results representation. Furthermore, representing a multi-line table cell, the TableCellTestRenderer method can be used.

6.2.5. ExtensionMechanism

This class is responsible for the managing of plugins mechanism by adding or updating the data access classes in the system. It provides methods to add or remove a number of data access classes in the extension mechanism from any directory. On startup of the system, the data access classes are not binded automatically, the classes has to be loaded in runtime. The concept of implementing extension mechanism is that system can load any number of data access classes in order to evaluate SQL statement executions in the Cloud.

- In order to make the application runtime extensible, a `Classloader` is to be implemented, where several classes can be loaded depending on data access needed. The `dalClassLoader` attribute invokes the class extension in the system.
- The method `loadDataAccessClass` is used to add or update the data access classes in the extension mechanism. While adding a new data access class, this method informs the `EvaluationWindow` class about the data store service connection status. However, in order to load a data access class to the extension mechanism all classes have to be in common structure, but can be modified one.
- To make the system robust the extended classes may require to remove from the extension mechanism. Using `removeExtendedClasses` method, it is possible to remove an added data access class in runtime.

Removing an added class from the extension mechanism is tricky, as it is realized that once a class is added to the `ClassLoader`, it cannot be removed from the system without system termination. However, this can be done using a temporary `ClassLoader`. The concept is that creating a temporary `ClassLoader` and copies all the loaded classes except removed one and then replace the original `ClassLoader` with the temporary `ClassLoader`. As a result the intended class has removed from the original `ClassLoader`. So that after removing a class form the `ClassLoader`, the same class can be loaded again if necessary. Moreover, while removing a data access class form the `ClassLoader`, it also removes that data service providers' name from the provider list.

6.2.6. SQLStatementEvaluation

It is one of the most important classes in the system, as the SQL execution is started from this class. Based on receiving target data store services name, this class invoke the targeted execution methods performing SQL execution. It provides the methods for evaluating the SQL data store services and comparing achieving SQL execution results with the expected results. Moreover, to determine the accurate execution results, this class implements individual evaluation method for each data store service. The method called `evaluateDataStoreSerivce()` is used to route the SQL execution to the target data store service. While start evaluating the SQL execution, a process bar is appeared for every checked data service providers. The process bar indicates the process executions in percentage and it disappears upon finishing the complete set of SQL statement executions. The execution processes are controlling by this class.

During running the SQL execution, for each statement this method receives a result set of SQL statement execution and then results are compared with predefined expected results through `compareReslult()` method. The comparison results are then updated to the results tables in real-time. It is realized that some of the achieving results are same as it is expected, but in different order format. The `compareResult()` method is also responsible for taking care of them. Furthermore, depending on the Cloud storage infrastructure some Cloud services are directly accessible, but some are not. They can be accessed establishing secure channel (SSH tunneling). This class provides supports for both direct data access as well as through SSH tunnel with specific database port forwarding. For establishing SSH tunneling, the JSch API may be used.

6.2.7. Data Access Classes

A package of data access classes builds the data access layer in the system, where these classes are implementing the concept of data access functionalities. These classes are responsible for performing SQL statements executions in a specific Cloud data store service. The concept of designing data access classes are that, a specific cloud data store service has to be accessed by using an individual data access class. This means that for every Cloud data store services, it has to be designed corresponding access classes and then the classes are loaded in the extension mechanism in application running time to access Cloud data. Additionally, all data access classes are designed with a common structured and the required adaptations are implemented in these classes. Different data store services may require different adaptations. So that, this approach makes the system simple.

The data access class provides the methods for SQL statements executions and adapting the statements. The `sqlStatementExecute()` method is used to execute the SQL statement in the Cloud data store service, where it returns the SQL execution results usually in SQL ResultSet format. In addition, before executing a SQL statement, it will verify whether the statement requires adapting or not. If requires, the `getSqlAdaptation()` method adapts the statement accordingly. And then send the adapted SQL statement to the Cloud data store service achieving acceptable results.

7. Implementation

This chapter illustrates the implementation details of SQL Evaluation tool based on requirements defined in Chapter 5 and the design issues discussed in Chapter 6. The software tool development was performed using standard Java platform, as Java is one of the most popular and common programming language for Cloud based application.

First of all, the Section 7.1 gives a detail implementation outline including code listings and references to the usage of most important classes and methods. Then the next section will define various technologies which are required for implementing a complete software system. Furthermore, a software tool manual including installation procedure as well as user manual is given in Section 7.3.

7.1. Software Tool Implementation

This section describes a detailed implementation of the core functionalities in SQL Evaluation tool system. The implementation is performed based on conceptual class diagram described in Figure 6.2, where the classes are designed according to the component based Software application system (see Fig. 6.1). However, the software implementation follows the Model View Controller (MVC) design, where wrapping objects are dealing with a model, the tool user interface representing a view and the core functionalities are performing actions for controlling the application system. Furthermore, this section explicitly describes the implementation of configurations setting and the design implementation of a data access layer.

7.1.1. Application Controller

The application controller is the main component of SQL Evaluation tool system. It has implemented various core functionalities of business logic for controlling core application system. It has included several important classes such as `EvaluationWindow`, `ExtensionMechanism` and `SQLStatementEvaluation`. The `EvaluationWindow` Class is responsible for representing a Graphical User Interface (GUI) window for performing user interactions. Moreover, all necessary User Interface (UI) element and UI related actions are performing by this class. Whereas, the `ExtensionMechanism` class deals with the implementation of plugins mechanism for providing opportunity to enhance application functionalities in runtime and the `SQLStatementExecution` class controls the SQL statement execution related tasks.

In order to represent the SQL statement execution results and data service provider list, the tool has two extensible Java forms using `GridBagLayout`. These forms allow application adding various Java UI components dynamically, which brings a great advantage for the application systems representing a number of data services providers' identity based on chosen a number o data access classes. Also, using this facilities a variable number of SQL statement evaluation results table are drawn. These results tables are created based on checked data service providers. The access control of both data forms are implemented in class `EvaluationWindow`. Using `GridBagLayout` form, the advantages is that the component does not necessarily has to define explicitly related position and size, as the size and positions are controlling by the `GridBagConstraints` object associated with each component. So that, every new component will be added in the end position of the form. However, it is also possible to recognize the components position in runtime.

```
...

// SQL execution progress bar is visible
sqlExcProcessBar.setVisible(true);

new SwingWorker<Void, Void>() {

  // Method to perform the background computation
  protected Void doInBackground() throws Exception {
    /*
     * Method calling form SQLStatementEvaluation class to execute
     * a set of SQL statements while the progress bar is running
     */
    sqlStateEval.evaluateDataStoreService("dataServiceName");
    return null;
  };

  // Mehod is called when the SwingWorker's doInBackground finishes
  protected void done() {
    // Hide SQL execution progress bar
    sqlExcProcessBar.setVisible(false);
  };
}.execute();

...
```

Listing 7.1: Excerpt from `EvaluationWindow.java` focusing on implementation of Java multi-threaded application using Swing. Only `SwingWorker` implementation is shown and the detail implementations are omitted.

Providing opportunity to execute several data store services simultaneously this tool has implemented multi-thread method execution. To ensure this facility, and to make the process visible, a Java `SwingWorker` has been implemented, where `SwingWorker` is an abstract class to carry out long running GUI interacting tasks in an enthusiastic thread. Implementing `doInBackgroud()` method in `SwingWorker` class, it makes feasible a long running tasks (i.e. a set of SQL statement execution) run in a background thread and gives update to the process bar. In this implementation, it is considered that while a set of SQL statement execution process is running the corresponding process bar will be visible to ensure the execution process updates and upon completing the tasks the process bar will invisible from the window. The code example in Listing 7.1 shows a sample of SwingWorker implementation, the SwingWorker is used to implement a Java multi-threaded application.

Moreover, the ExtensionMechanism class is responsible for providing functionalities to extend the data access functionality by loading a data access class to the extension mechanism in runtime. Also, it provides the functionality to remove an added class from the extension mechanism container. As it is considered extending the application functionality by loading only classes, a ClassLoader has been implemented. The implementation of ClassLoader is straightforward and in implemented ClassLoader a multiple number of classes can be loaded. So that it makes feasible adding any number of data access classes in the application systems for accessing different data store services. Listing 7.2 shows a part of ExtensionMechanism class, where a Java ClassLoader has been implemented. Usually, a as standard Java extension mechanism offers the possibility to add a complete Java Archive (JAR) API into the extension mechanism. As we are implementing individual data access classes for different data service providers, it is considered implementing Java ClassLoader for loading different data access classes.

```java
public class ExtensionMechanism {

  public ClassLoader dalClassLoader;      // Java ClassLoader
  public Class<?> dalClass;               // Associated class object
  private String classToBeLoaded = classPath + "." + dalClassName;  // Class loading path
  ...

  public void loadDataAccessClasses(){
    ...
    try{
      /*
       * Load data access class to the extension mechanism and
       * instantiate with local class object
       */
      dalClass = dalClassLoader.loadClass(classToBeLoaded);
    }
    ...
    catch (ClassNotFoundException e) {
      /*
       * Thrown when application tries to load in a class through its string name,
       * but no definition for the class with the specified name could be found.
       */
    }
    ...
  }
}
```

Listing 7.2: Excerpt from ExtensionMechanism.java emphasizing the implementation of Java plugins mechanism to extend application functionality of data access layer. Only ClassLoader implementation is shown and the detail implementations are omitted.

Furthermore, the class SQLStatementExecution is the main class for starting SQL statement execution. It provides the methods for routing statement execution to a specific Cloud data store service and comparing the execution results with expected results for representing the outcomes of the execution, whether the SQL statement execution is passed or failed. The `evaluateDataStoreService()` methods routes SQL statement execution by calling explicit execution method to a specific Cloud data store service based on given service name.

```java
private void evaluteRdsOracleDataStoreService(){
  ...
  try{
    // Create new instance for rdsOracleClass which is loaded in extension mechanism
    Object rdsOracleInstance = extMechanism.rdsOracleClass.newInstance();
    // Data access method instanciation
    Method sqlExecMethod = extMechanism.rdsOracleClass.getMethod("sqlStatementExecute",
        new Class[] { Vector.class, String.class, int.class}
    );
    // Receive query results with method invocation for Cloud data access
    String queryResult = (String) sqlExecMethod.invoke(rdsOracleInstance,
        new Object[] { dssConf, sqlStatement });
    // Comparing actual SQL statement execution results with expected results
    boolean resComp = compareResult(expectedResult, queryResult);
    ...
  }catch (SecurityException e) {

  }
  ...
}
```

Listing 7.3: Excerpt from `SQLStatementEvaluation.java` focusing on data access method invocation in the extension mechanism. The detail implementations are omitted and only a pseudo code of `evaluteRdsOracleDataStoreService()` method is shown.

Listing 7.3) shows an abstract overview how application instantiate with the data access functionality in order of evaluating SQL statement execution in Oracle data store service in Amazon RDS and compare the results. First of all, the method of data access class has been instantiate with a method variable with given specific number of parameters. Then invoke the data access method calling with defined types of values (i.e. database configuration and SQL statement) to perform the data access method execution. So that the data access method returns the SQL statement execution results set in string format. Then the achieved results are compared to the expected results for determining the outcomes decision, the status of the SQL execution (i.e. passed or failed).

7.1.2. Java Utility

In order to represent a dynamic number of Cloud data service providers name and draw a number of extensible tables for SQL statements evaluation, a Java extensible form layout as well as a table template has been implemented in this system. The extensible form layout is implemented with the help of `FromUtility` wrapper class, where `FromUtility` class provides functionalities to add any component (e.g. label, checkbox) and Java table. In order to ensure this extensible facility, the `FromUtility` class has implemented applying `GridBagLayout` objects. So that, the component can be added dynamically into the form grid and then the `GridBagLayout` automatically control the component size and position for each added component.

While running the SQL statements execution on several data store service simultaneously, is it quite important to represent the different execution results in a fashioned manner. As the SQL statement test inputs are variable the table size must be varies. The rows of the table are equal to the number of SQL statement test inputs. However, the columns are always same.

The all table representations are in same size, as the SQL input size is always same. Therefore, a table template is implemented for drawing a variable size of table. Moreover, the tables rows are consider representing a multi-lined table cell. Because of, the achieving result sets of a SQL statement execution can be a table. It does mean, often we have to represent a table inside a table.

7.1.3. Configurations Setting

This section describes the implementation of detailed configurations setting including database configuration, SQL inputs, and expected results.

While Cloud data store services have to be accessed remotely through Internet, within this systems a database configuration has been implemented explicitly. All required data service configuration are stored in allocated database configuration file called `dbconfig.xml` with XML data format. Listing 7.4 shows an example database configurations for a few Cloud data store services, where the configurations are designed explicitly for every individual data store services.

In every configuration setting, it is considered including both Cloud service connection information as well as the specific data store service configuration. However, it is investigated that some of the Cloud data store services are feasible accessing directly through public URL. In this case the Cloud service connections information fields are holding null value, but the data store services configuration fields are stored with necessary values. For example, the database host would be the public URL of data store service. Furthermore, it is also realized that some for the data store services have to be accessed implementing with a public key authentication. In this case the application creates SSH tunneling session with specific database port forwarding and then access Cloud data through forwarded port. Implementing XML DOM parsing the `getDatabaseConfiguration()` method in `DatabaseConfiguration` class, the system retrieves that necessary data store service configuration for a specific data service provider.

In addition, the database configuration can be updated in system running time. Once the database configuration is updated the system will automatically notify about the update and will use the updated configuration for the further data access. Nevertheless, for every data service provider the database configuration file has designed with fixed number of elements item. If it is required to change the structure of database configuration, then also

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dbconf>

  <defaultmysql id="0" name="defaultmysqldb">
    <cloudurl>NULL</cloudurl>
    <clouduser>NULL</clouduser>
    <dburl>jdbc:mysql://localhost:3306/</dburl>
    <dbname>dbName</dbname>
    <dbusername>dbUser</dbusername>
    <dbpassword>dbPass</dbpassword>
  </defaultmysql>

  <ec2postgresql id="1" name="ec2postgresqldb">
    <cloudurl>ec2-xxx-xxx-xxx-xxx.eu-west-1.compute.amazonaws.com</cloudurl>
    <clouduser>ec2-user</clouduser>
    <dburl>jdbc:postgresql://localhost:5432/</dburl>
    <dbname>dbName</dbname>
    <dbusername>dbUser</dbusername>
    <dbpassword>dbPass</dbpassword>
  </ec2postgresql>

  <rdsoracle id="2" name="rdsoracledb">
    <cloudurl>NULL</cloudurl>
    <clouduser>NULL</clouduser>
    <dburl>jdbc:oracle:thin:@dbName.ctdc03yqnvpy.us-east-1.rds.amazonaws.com:1521:</dburl>
    <dbname>dbName</dbname>
    <dbusername>dbUser</dbusername>
    <dbpassword>dbPass</dbpassword>
  </rdsoracle>

</dbconf>
```

Listing 7.4: A part of data store service configuration `dbconfg.xml` for enable accessing both traditional as well as the Cloud data.

it is necessary to update the data retrieving method in the system level accordingly, as currently the `getDatabaseConfiguration()` method is implemented to get the fixed number of configuration elements.

As we consider evaluating Cloud relational data store services, a set of SQL statement inputs have been designed based on SQL standard 2003. In order to evaluate the adaptation of data access layer, the same SQL statements are to be executed in different data store services. Moreover, a set of corresponding expected SQL results are defined to the system for comparing with actual SQL statement execution results. The expected results are prepared based on traditional database server. The Table 7.1 shows a part of usages important SQL statements inputs and corresponding expected results.

Table 7.1.: A part of usages important SQL statements inputs and corresponding expected results.

SQL Statement	Expected Result
SELECT object_id, access_level_browse FROM oocke1_-account	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeMf4vj8cA 3
SELECT object_id, access_level_browse FROM oocke1_account WHERE object_-id='account/CRX/Standard/admin-Standard'	account/CRX/Standard/admin-Standard 3
SELECT object_id, AVG (access_level_browse * access_-level_delete) FROM oocke1_calendar GROUP BY object_id	calendar/CRX/Mohsin/9LPADU3ML74YP2C3LAQ 7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusinessCalendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29BeMf4vj8cA 6.0000
SELECT access_level_browse, full_name FROM oocke1_account INNER JOIN oocke1_account_ ON (oocke1_account.object_id=oocke1_account_.object_id) WHERE (oocke1_account_.idx=1) ORDER BY oocke1_-account.full_name	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private
SELECT access_level_browse, full_name FROM oocke1_-account LEFT JOIN oocke1_account_ ON (oocke1_-account.object_id=oocke1_account_.object_id) WHERE (oocke1_account_.idx=2)	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,
SELECT persons.LastName, persons.FirstName, orders.OrderNo FROM persons FULL JOIN orders ON persons.P_Id=orders.P_Id ORDER BY persons.LastName	Hansen Ola 22456 Hansen Ola 24562 Pettersen Kari 77895 Pettersen Kari 44678 Svendson Tove null null null 34764

7.1.4. Data Access Layer

As within Data Access Layer (DAL) it is considered implementing individual data access classes for every data service provider, the data access classes are implemented based on required SQL statement adaptations found in Chapter 4. The main concern of implementing data access classes individually for every specific Cloud data service provider is that the system can be extended data access functionalities by adding a new data access class in run-time. The another reason for designing data access classes individually is that, for accessing different Cloud data store services, it requires adapting the data access classes with different tapes of SQL statement adaptations. Thus reduces the system complexity and ensuring the correctness of data access feasibility.

In the design structure of data access classes, every class has implemented with two essential methods, such as `sqlStatementExecution()` and `getSqlAdaptation()`. However, the classes can be included more functionalities due to perform internal processing tasks but above described methods are the least requirements. The `sqlStatementExecute()` method performs executing SQL statements to the specific Cloud data store service, where as the `getSqlAdaptation()` method performs adapting the SQL statement, if require. For every SQL statements execution, the `sqlStatementExecute()` method asks the `getSqlAdaptation()` whether the statement requires adapting or not. If adaptation is required the `getSqlAdaptation()` method adapt the statement according to the adaptations described in Chapter 4 and returns the adapted SQL statement, otherwise it will confirm the statement is already valid. As a result the `sqlStatementExecute()` only perform executing the valid statements. So that, according to this concept, the system wills always adequate SQL results.

Original SQL Statement	Adapted SQL Statement
CREATE TABLE test_table1 (id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, name VARCHAR(250) NOT NULL, age INTEGER UNSIGNED, marks BIGINT UNSIGNED, PRIMARY KEY (id))	CREATE TABLE test_table1 (id NUMBER(10, 0) NOT NULL, name VARCHAR2(250) NOT NULL, age NUMBER(10, 0), marks NUMBER(19, 0), PRIMARY KEY (id))

Table 7.2.: An example of SQL `CREATE TABLE` Adaptation from default to Oracle.

```java
/**
 * Method to get the adaptation on SQL CREATE TABLE statement.
 * SQL Statement migration from default to Oracle
 *
 * @param sqlStatement  The SQL statement is to be adapted
 * @return              Returns the adapted SQL statement
 */
private String getCreateTableAdaptation(String sqlStatement){
  /*
   * SQL statement is to be adapted by replacing with specific data types
   */
  String adaptedStatement = wordReplace(sqlStatement, "␣UNSIGNED", "");
  adaptedStatement = wordReplace(adaptedStatement, "␣AUTO_INCREMENT", "");
  adaptedStatement = wordReplace(adaptedStatement, "BIGINT", "NUMBER(19,␣0)");
  adaptedStatement = wordReplace(adaptedStatement, "INTEGER", "NUMBER(10,␣0)");
  adaptedStatement = wordReplace(adaptedStatement, "VARCHAR", "VARCHAR2");
  ...
  // Return the adapted SQL statement
  return adaptedStatement;
}
```

Listing 7.5: Excerpt from `Ec2OracleAccess.java` focusing on implementation of SQL CREATE TABLE Adaptation from default to Oracle. Only an example implementation is shown based on Table 7.2 described and the detail implementations are omitted.

Table 7.2 shows an example SQL CREATE TABLE statement and the possible adaptation, where the Listing 7.5 describes the possible implementation solution when statement is migrating to Oracle compatible.

As it is observed that some of the Cloud data store services have to be accessed through SSH tunneling with public key authentication, the initial though was implementing SSH session in data access layer. However, due to the fact that if the SSH session is created for every statements executions, the system will take a large amount of time to finish the complete set of SQL statement execution. Therefore, the implementation of SSH session has moved to the `SQLStatementEvaluation` class within the system, where from the SQL statements are started to be executing. The implementation steps are that create a SSH session for from the system level and execute the complete set of statements and then release the SSH session while all executions are done. Nevertheless, with this approach one problem could appear that if the SSH session is broken for some reasons the system cannot automatically establish a SSH session for completing the upcoming executions.

7.1.5. JSch Based Authentication

As some of the Cloud computing instances (i.e. Amazon EC2) act as a remote server and have to be accessed through SSH tunneling, several Java libraries could be used, for example JSch, MindTerm, orion-ssh2, Ganymed SSH-2, ssj, etc. For some reasons, it is considered using JSch library in order to establish a SSH tunneling by public key authentication for accessing Cloud data. Implementing JSch SSH tunneling session is very straightforward, as several documentations could be found on online, which already stated the basic implementation.

Listing 7.6 describes the necessary steps for establishing SSH tunneling over port 22 and forwarding MySQL database connection port 3306.

1. Create a new instance of JSch API class.
2. Add the private key identity to the JSch instance.
3. Create JSch session using specific Cloud host name and user name over SSH port 22.
4. Overwrite the JSch user information by implementing sshUserInfoWrap (SSH user information wrapper) class.
5. Connect to the JSch SSH session.
6. Forward database port (i.e. 3306) with L option.
7. Access MySQl data store service using forwarding port.
8. Release the JSch SSH session.

```java
...
try{
  // Create new instance of JSch API class
  JSch jsch = new JSch();
  // Add private key identity to the JSch instance
  jsch.addIdentity(privKeyPath);
  // Create JSch session using Cloud host and user over SSH port 22
  Session session = jsch.getSession(user, host, 22);
  // Overwrite the JSch user information with wrapper interface
  UserInfo uiWrapper = new SshUserInfoWrap();
  // Set the wrapping user information to the session
  session.setUserInfo(uiWrapper);
  // Connect to the JSch SHH session
  session.connect();
  if(session.isConnected()){
    // Forward database port with L option
    session.setPortForwardingL(3307, "localhost", 3306);
    ...
    // Release the JSch SSH session
    session.disconnect();
  }
}catch(Exception e){
  System.out.println(e);
}
...
```

Listing 7.6: Java SSH tunneling with JSch API.

7.2. Development Environment

This section describes detail development environment including usages tool, platform and technologies for the implementation of SQL Evaluation tool. The software application was developed in Java programming language (Java 7) using Eclipse IDE (Integrated development Environment) (Eclipse Juno version 4.2) under windows operating system. Moreover, several Java technologies such as Java libraries APIs (Application Programming Interface) have been used for building a complete tool system. The description of development environment and usages technologies are given in the following subsections.

7.2.1. Eclipse

While SQL Evaluation tool has been implemented in Java, we have decided to develop the application in Eclipse development environment using Java integrated development environment (Java Development Tools (JDT)). As the application systems had build on standard Java platform, it is consider using Eclipse IDE for Java developers, Eclipse Juno version 4.2.

Eclipse IDE is a multi-platform supported open source development environment for building Java application. It is one of the wide accepted software development environment within Java developers [Gee05], as it is an open source platform and have offered various significant features such auto completing Java source code, integrated debugger, integrated Java compiler, symbol reference, declaration locator and team development applying version control system.

7.2.2. Java Platform

Developing a software tool was a major part of this master's thesis, where the tool is using for proofing the concept of adaptation of data access layer in order to enable access Cloud data. The software development was based on layering application architecture model standards described in Chapter 5 and 6. However, the tool is platform independent and runs on both Windows and Linux platform, as Java applications are normally run on any supported operating system platform such Windows, UNIX, Macintosh, and Solaris. The software application was developed in Java platform, Standard Edition Development Kit (JDK) version 7, which includes Java standard edition runtime environment 1.7. Additionally, the tool has implemented some special features for example string switch cases. So that it makes limitation for using Java runtime environment, as it can only run on JDK 7, not on earlier version of JDK.

The reason for selecting Java programming language for the SQL Evaluation tool development was that Java is an open source language platform itself and helps on developing open source application. Also, Java is a popular and widely accepted for building Cloud based applications nowadays. An additional reason for selecting Java was that the adaptation concept of data access layer will be a part of Enterprise Service Bus (ESB) Apache ServiceMix, where Apache ServiceMix is a Java based platform.

7.2.3. Java Libraries

This section illustrates the various technologies (i.e. Java libraries) that were used during software tool implementation in order to provide that necessary functionalities in the system. A short description of most important usages libraries are given in following in this section.

JDBC

While Java application accessing Cloud Relational Database Management System (RDMS), a JDBC (Java Database Connectivity) database connector library was used. JDBC is an API for building low-level JDBC drivers, where low-level driver is responsible for connecting and accessing data source from the database system. It is standard for most of the popular RDMS and currently a part of JDK.

As different database systems have implemented with its unique design structure philosophy, every database has its own API for access data resources. For example, accessing MySQL database resources, a MySQL Connector/J JDBC driver can be used. Similarly, ojdbc driver is used for accessing Oracle database and postgresql-jdbc driver is used for PostgreSQL database. Therefore, accessing different relational Cloud data store services, following JDBC drivers are used in this system.

Data Store Service	Driver version
MySQL	mysql-connector-java-5.1.18-bin.jar
Oracle	ojdbc6.jar
PostgreSQL	postgresql-9.1-902.jdbc4.jar

Table 7.3.: List of JDBC drivers for connecting different relational data store services.

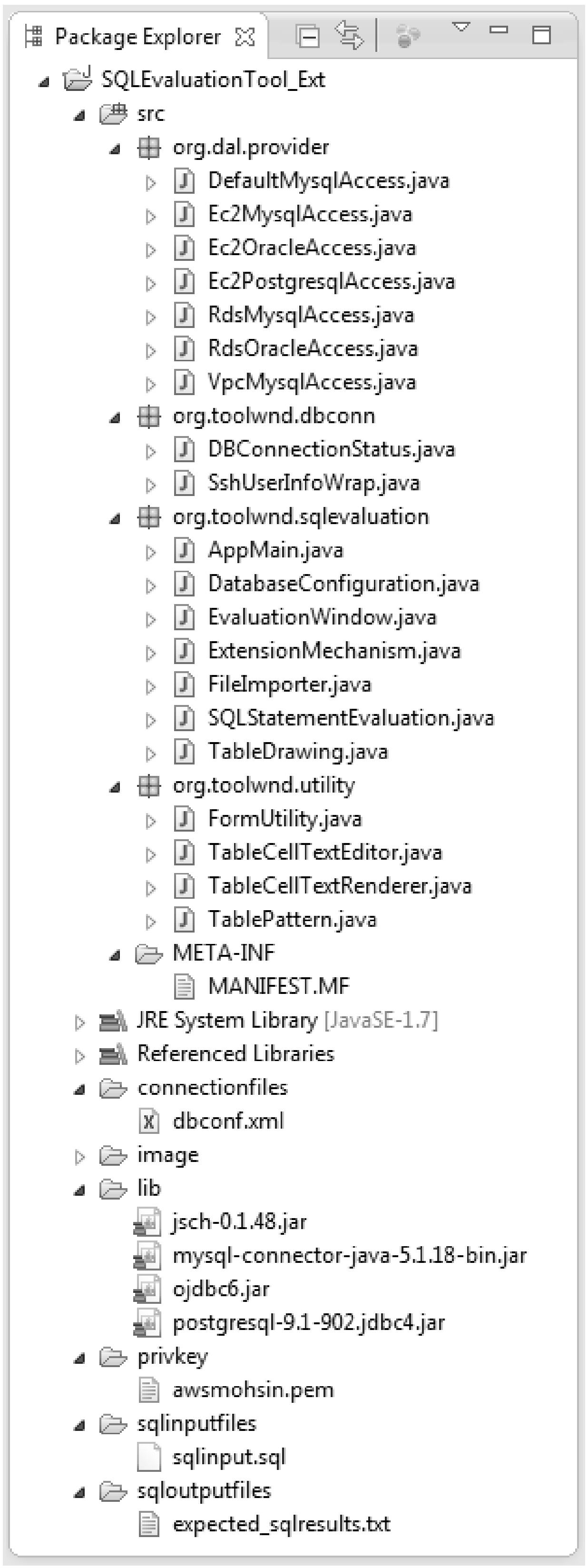
JSch API

As SSH is not supported in Java native implementation, a third party API called JSch (Java Secure Channel) was used to implement a SSH tunneling for accessing Cloud data, when needed accessing the data through a secure channel. This means that using JSch, a secure channel is established and application accessing Cloud data through the channel. Implementing SSH tunneling, this application has considered using JSch API version jsch-0.1.48.

JSch API is implemented in Java programming language, which can be used to connect to remote sshd server for various purposes, such as SSH key authentication, using port forwarding, X11 forwarding, SSH tunneling, file transfer, etc. However, in SQL Evaluation tool system, it has implemented this API for accessing Cloud data storage (i.e. MySQL and PostgreSQL database on Amazon EC2). The concept of using this API is that application creates a secure channel (i.e. SSH tunneling) between local server and Cloud Instance over port 22 and forwarding a specific database port. Then application connect to the data store service using forwarded port [JCr].

7.3. SQL Evaluation Tool Manual

This section deals with detail software tool installation and user manual. The installation illustrates a detail installation procedure and prerequisites for running the software tool system. The user's manual shows the steps, how to use the tool for evaluating SQL statement adaptations while application accessing Cloud data. This tool has been developed for accessing relational data store services in the Cloud.



```
SQLEvaluationTool_Ext
  src
    org.dal.provider
      DefaultMysqlAccess.java
      Ec2MysqlAccess.java
      Ec2OracleAccess.java
      Ec2PostgresqlAccess.java
      RdsMysqlAccess.java
      RdsOracleAccess.java
      VpcMysqlAccess.java
    org.toolwnd.dbconn
      DBConnectionStatus.java
      SshUserInfoWrap.java
    org.toolwnd.sqlevaluation
      AppMain.java
      DatabaseConfiguration.java
      EvaluationWindow.java
      ExtensionMechanism.java
      FileImporter.java
      SQLStatementEvaluation.java
      TableDrawing.java
    org.toolwnd.utility
      FormUtility.java
      TableCellTextEditor.java
      TableCellTextRenderer.java
      TablePattern.java
    META-INF
      MANIFEST.MF
  JRE System Library [JavaSE-1.7]
  Referenced Libraries
  connectionfiles
    dbconf.xml
  image
  lib
    jsch-0.1.48.jar
    mysql-connector-java-5.1.18-bin.jar
    ojdbc6.jar
    postgresql-9.1-902.jdbc4.jar
  privkey
    awsmohsin.pem
  sqlinputfiles
    sqlinput.sql
  sqloutputfiles
    expected_sqlresults.txt
```

Figure 7.1.: An Overview of Eclipse project explorer for SQL Evaluation tool development.

7.3.1. Installation

The software tool has been exported with a runnable JAR file, which is compatible running on both Windows and Linux platform. However, the prerequisites and configurations setting are important to meet in order of running the application system.

First of all, it is necessary to have the Java runtime environment within the operating system and the application requires Java JDK 7.0 environment. Therefore, we will install Sun Java JDK 7.0. The software tool has been developed in Java 7 using some new features such as String based switch case structure and updated Java extension mechanism, etc. So that, the early version of Java will not work. Moreover, installing a JRE only is not sufficient to run the application. It is required to have a full-blown of JDK. While installing JDK on windows, it is better to avoid paths containing blanks, as application may sometimes get confuse to determine the Java runtime. However, it is investigated that this system is running perfectly while installing Java in default directory (i.e. "C:/Program Files").

After installing JDK and Java runtime, it is necessary to set the environment variable JAVA_-HOME, where environment variable is pointed to the JDK installation directory, e.g. "C:/Java /jdk1.7.0/_06" on windows. Additionally, it is also necessary to set the path value for running Java applications, such as "C:/Java/jdk1.7.0/_06/bin". After setting up Java environment variable and path, it is time to test the Java runtime environment. On windows, open a Console application and execute the command simply "Java -version". It will show the Java version and runtime environment. Thus confirm Java environment is set up correctly.

The next step is to configure the application system to make ready the application for the evaluation of SQL statements executions in different Cloud data store services. In order to perform predictable computations, the application requires various configurations setting. Figure 7.1 shows a project explorer view for SQL Evaluation tool development, where it indicates the necessary packages required for the application system and the additional directories for defining configuration settings. The necessary configurations are described in the following:

1. Setup database configuration file (i.e. dbconfig.xml) in "/conectionfiles" directory.
2. Setup all required database connectivity libraries (i.e. JDBC drivers) in "/lib" directory.
3. Setup necessary private keys (i.e. awsmohsin.pem) in "/privkey" directory. The private keys must be with *.pem extension.
4. Setup SQL statement input file (i.e. sqlinput.sql) in "/sqlinputfiles" directory. The SQL input file must be with *.sql extension.
5. Setup corresponding SQL results file (i.e. expected_results.txt) in "/sqloutputfiles" directory. The results file must be with *.txt extension.

Now we are ready to run the application and evaluate the adaptation of SQL statement in order to access Cloud data. The next section will describe the steps of using the application in detail.

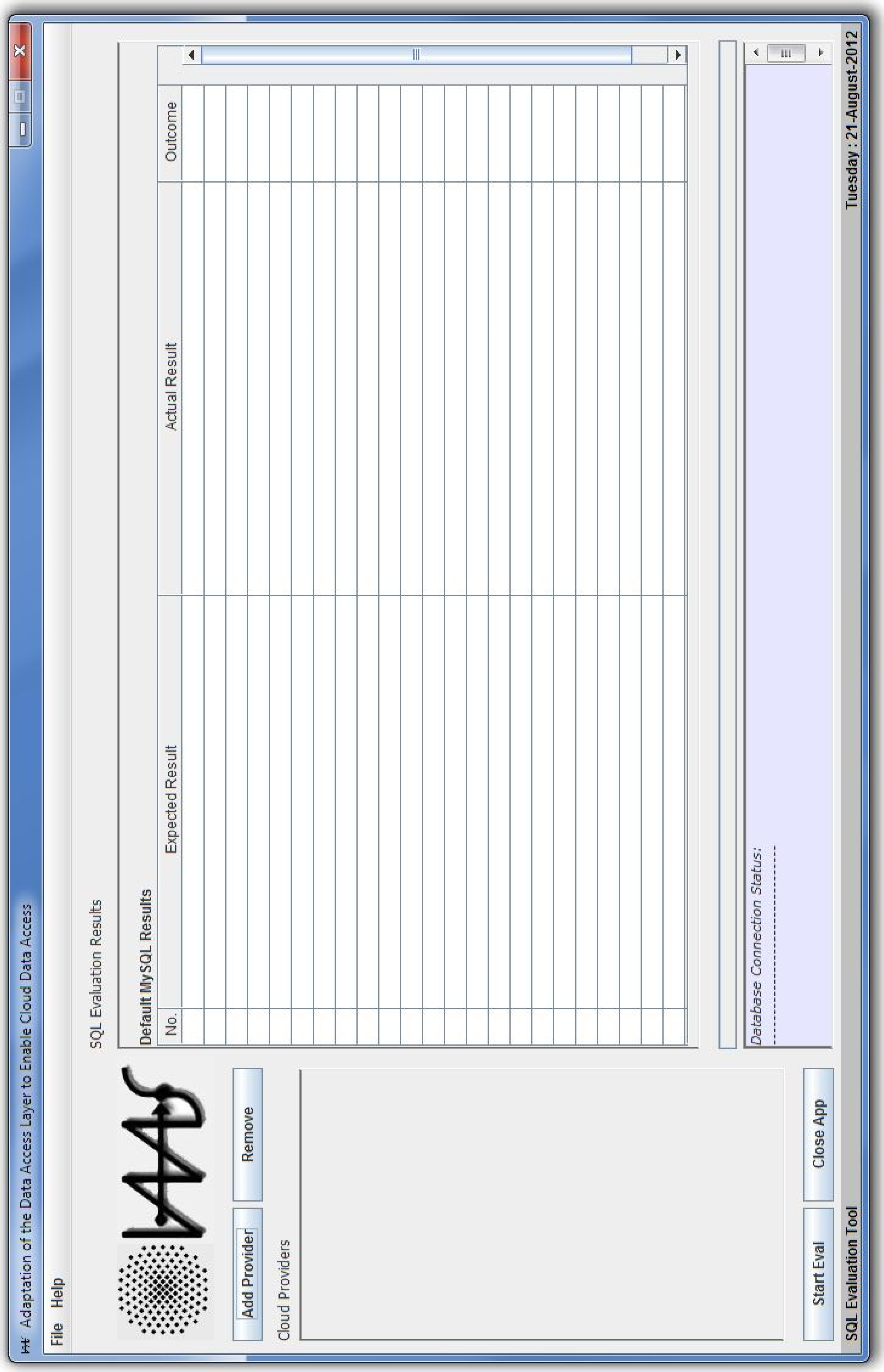


Figure 7.2.: Default application window of SQL Evaluation tool.

7.3.2. User's Manual

This section explains how to use the main functions of SQL Evaluation tool. The guide provides supports in detail, from launching the application to evaluate the adaptation of SQL statements.

Default Window

After installing detail setup configurations and met the prerequisites, the software tool launches with the graphical window shown in Figure 7.2. In this main window it has included various options such as menus, buttons and especially two extensible forms to perform user interactions.

File Menu

The *File Menu* contains four submenus including *Import Query File*, *Import Query Results*, *Import Private Key* and *Exit*. Figure 7.3 shows the File Menu with all submenus. Short descriptions of submenus are given in the following:

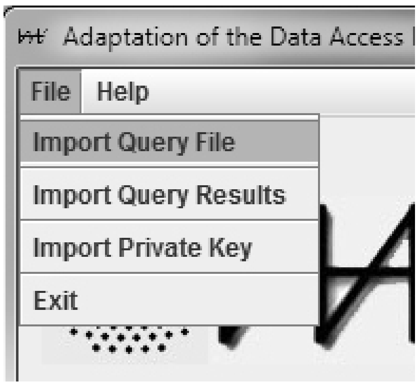
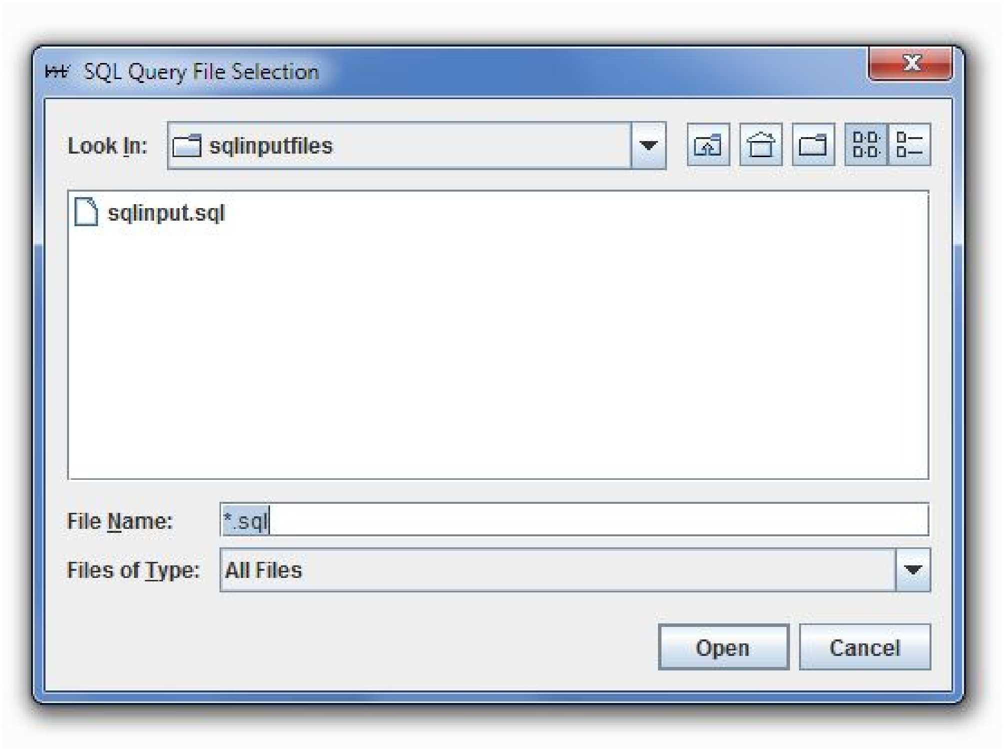


Figure 7.3.: File Menu.

Import Query File: To perform SQL statement execution in the Cloud data store services, a set of SQL statements has to be imported in the system. Using Import Query File menu user may upload a SQL statement file (see Fig. 7.4), which contains a set of SQL statements. While uploading a SQL statement input file, it is important to consider that the file must be with *.sql extension.


SQL Query File Selection

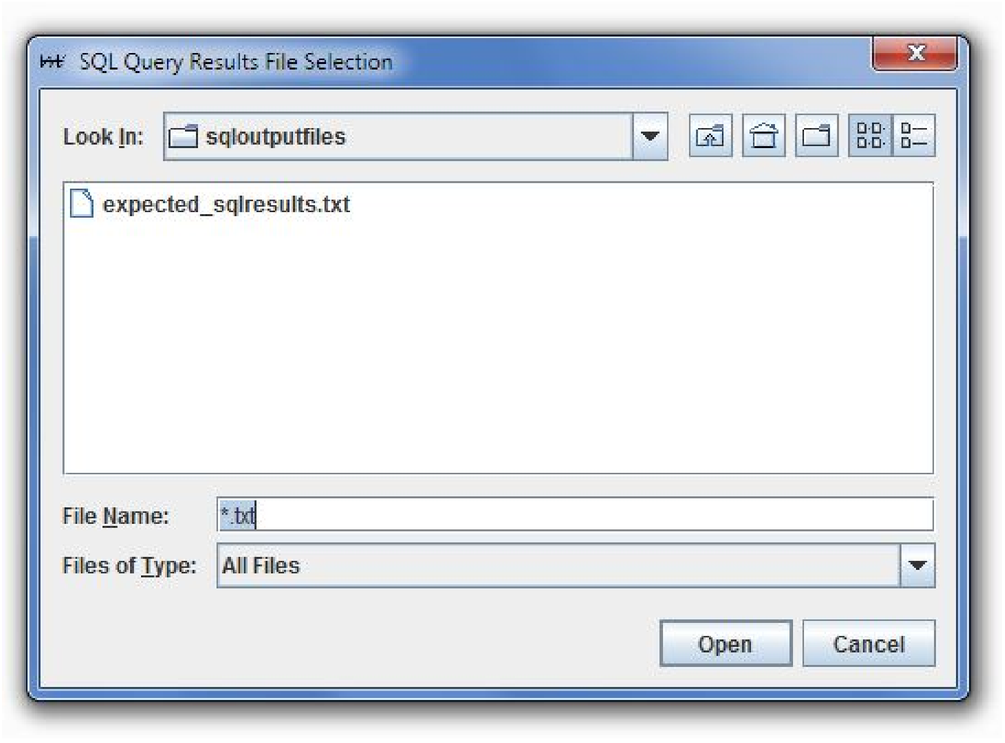
Look In: sqlinputfiles

sqlinput.sql

File Name: *.sql

Files of Type: All Files

Figure 7.4.: Import SQL statements for executing in different data store services.


SQL Query Results File Selection

Look In: sqloutputfiles

expected_sqlresults.txt

File Name: *.txt

Files of Type: All Files

Figure 7.5.: Import corresponding SQL statement results for comparison with actual execution results.

Import Query Results: In order to compare the SQL statements executions results, it is necessary to import a set of corresponding SQL results based on selected SQL statements input in the system. Using Import Query Results menu user will upload the corresponding SQL results (see Fig. 7.5). However, the SQL results file must have *.txt extension.

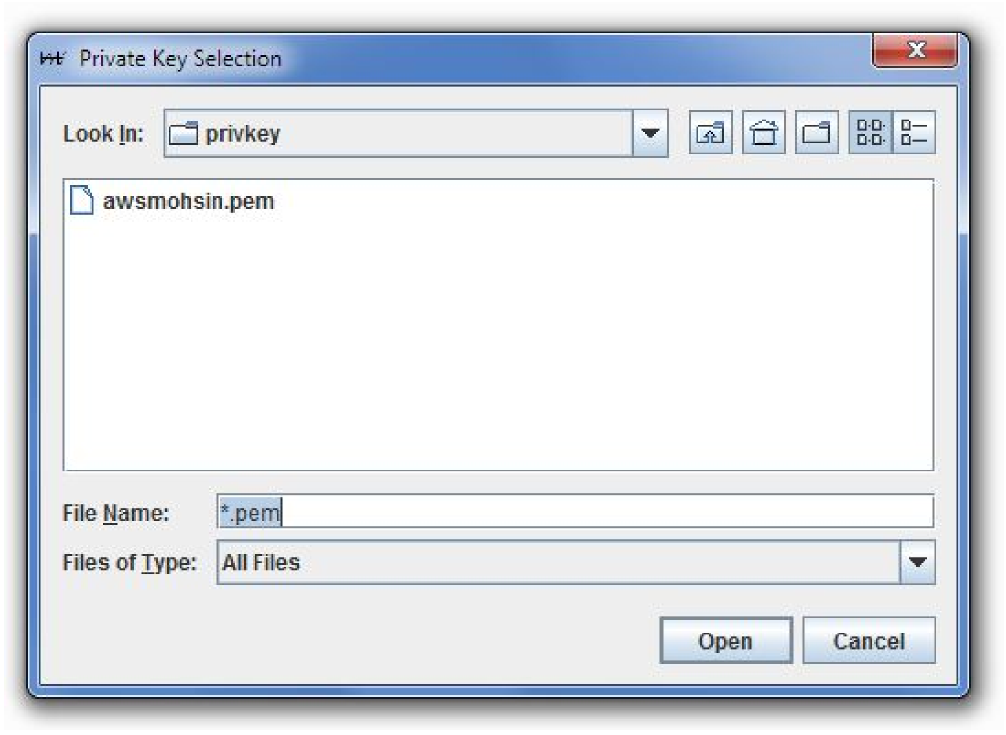


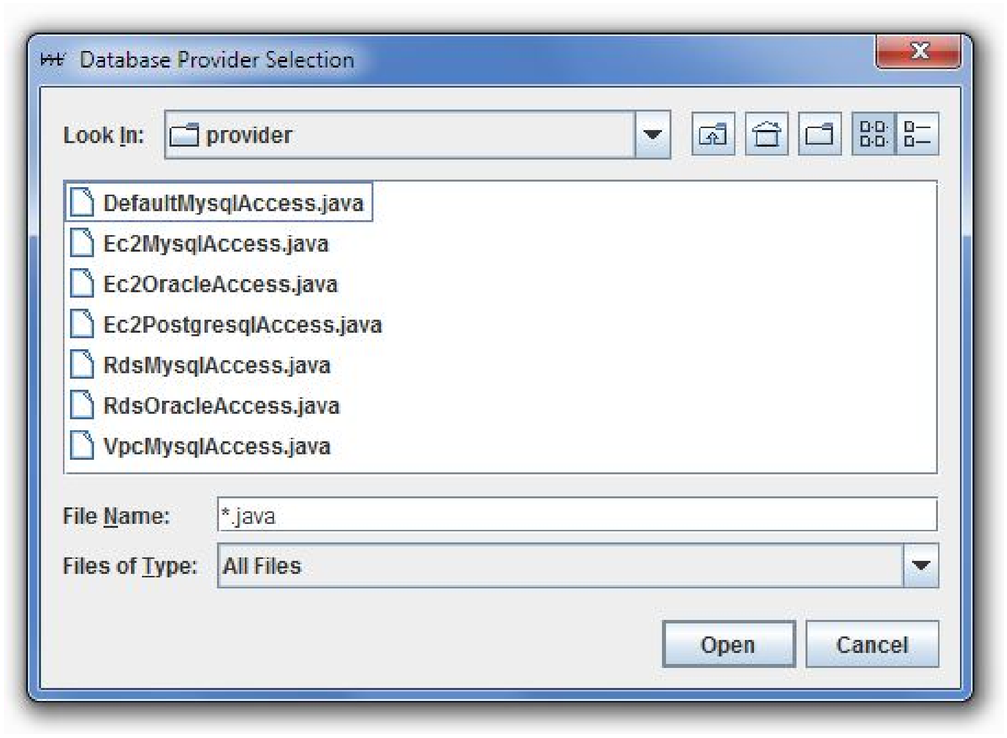
Figure 7.6.: Import private key file for establishing SSH tunneling session.

Import Private Key: As some of the Cloud infrastructures requires public key authentication, the user has to select an appropriate private key file (see Fig. 7.6) for establishing a secure channel in order to access remote instances. Import Private Key menu allows user to upload the private key file in the system. Moreover, it is important to have the private key with *.pem extension.

Exit: Exit menu let the user close the application.

Add Provider

Add Provider button offers the possibility adding one or more data access classes to the extension mechanism from any directory in order to enhance the data access functionalities within the system. The data access classes files must be in *.Java extension. Clicking on *Add Provider* button, a Java file chooser window appears (see Fig. 7.7), where user may chose one or more data access classes. While adding data access classes to the system, the data service providers name will be attached to the Cloud providers list with individual checkboxes for every chosen data service providers. Additionally, during adding data access classes to the system, it checks the data store service connection status for every service providers. The status reports are given in data connection status text area. Furthermore, a process bar has set in above of connection status, which ensures the status checking processes, while adding several data access classes.


Database Provider Selection

Look In: provider

DefaultMysqlAccess.java
Ec2MysqlAccess.java
Ec2OracleAccess.java
Ec2PostgresqlAccess.java
RdsMysqlAccess.java
RdsOracleAccess.java
VpcMysqlAccess.java

File Name: *.java

Files of Type: All Files

Figure 7.7.: Adding data access classes to the extension mechanism for enhancing data access functionalities within application system.

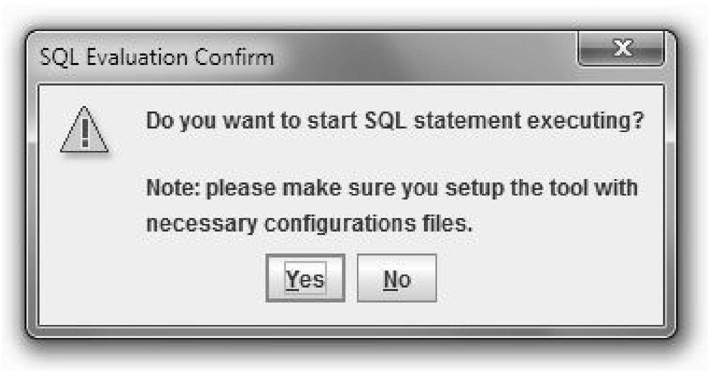
Remove Provider

Remove provider button permits user to remove the added data access classes from the extension mechanism. In order to remove a data service provider from the system, the user has to check the checkbox belongs to the data service provider name in Cloud providers list. Then by clicking on Remove button the data service provider is removed from the Cloud providers list as well as the extension mechanism.

Start Evaluation

After adding data access classes in the extension mechanism, it is time to select the checkboxes based on success database connection status performing SQL statements execution. While selecting data service providers in the Cloud providers list, it must be considered that the failed connections cannot enable accessing Cloud data.

After selecting data service providers in Cloud providers list, the *Start Eval* button has to be used for starting SQL statement execution. While clicking on *Start Eval* button, the system asks user confirmation (see Fig. 7.8) for starting execution and then user has to confirm by clicking Yes. Upon confirming the execution, the system starts performing SQL statement execution. Moreover, ensuring the SQL statements execution process, a process bar appears for every selected data service providers behind their name. The process bar shows the execution progress calculating percentage values. However, while finishing the execution the process bar disappears. This has to be done for every individual process executions.


SQL Evaluation Confirm

Do you want to start SQL statement executing?

Note: please make sure you setup the tool with necessary configurations files.

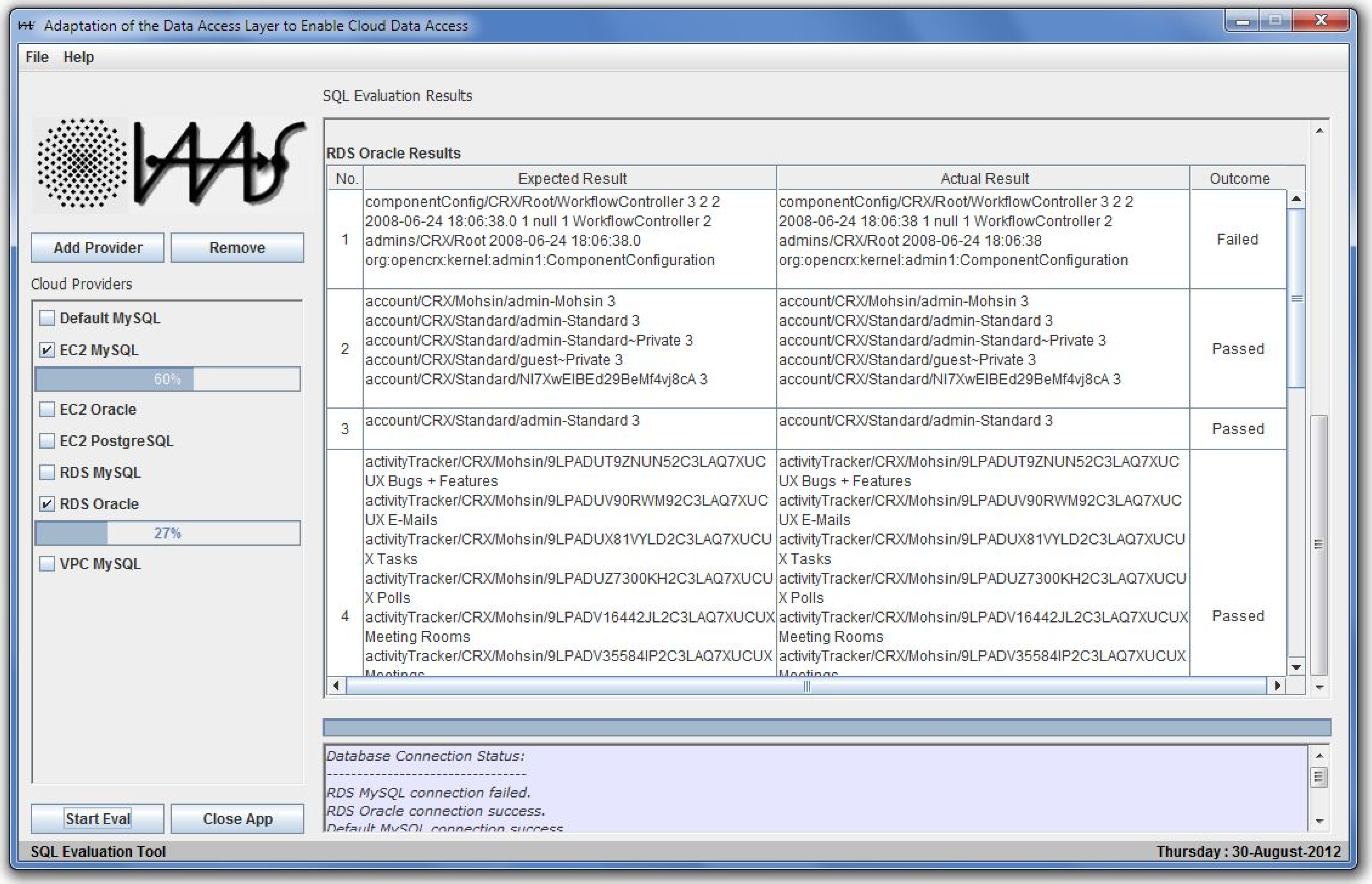
Figure 7.8.: SQL execution starting confirmation dialogue.

SQL Evaluation Results

While starting SQL statement executions on different Cloud data store services, the individual results table has to be created based on selected data services providers. The results table shows the comparison results including expected and actual results (see Fig. 7.9). It also shows the outcomes (i.e. passed or failed) for every individual SQL statement executions. Furthermore, when running the SQL execution processes in different Cloud data store services, all results table are updating in run-time with comparing results.

Close Application

After finishing all processes for SQL statement evaluation, the *Close App* button may use to close the application.



SQL Evaluation Results

RDS Oracle Results

No.	Expected Result	Actual Result	Outcome
1	componentConfig/CRX/Root/WorkflowController 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfiguration	componentConfig/CRX/Root/WorkflowController 3 2 2 2008-06-24 18:06:38 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38 org:opencrx:kernel:admin1:ComponentConfiguration	Failed
2	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard~Private 3 account/CRX/Standard/guest~Private 3 account/CRX/Standard/NI7XwEIBEd29BeMf4vj8cA 3	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard~Private 3 account/CRX/Standard/guest~Private 3 account/CRX/Standard/NI7XwEIBEd29BeMf4vj8cA 3	Passed
3	account/CRX/Standard/admin-Standard 3	account/CRX/Standard/admin-Standard 3	Passed
4	activityTracker/CRX/Mohsin/9LPADUT9ZNUN52C3LAQ7XUC UX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90RWM92C3LAQ7XUC UX E-Mails activityTracker/CRX/Mohsin/9LPADUX81VYLD2C3LAQ7XUCU X Tasks activityTracker/CRX/Mohsin/9LPADUZ7300KH2C3LAQ7XUCU X Polls activityTracker/CRX/Mohsin/9LPADV16442JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV35584IP2C3LAQ7XUCUX	activityTracker/CRX/Mohsin/9LPADUT9ZNUN52C3LAQ7XUC UX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90RWM92C3LAQ7XUC UX E-Mails activityTracker/CRX/Mohsin/9LPADUX81VYLD2C3LAQ7XUCU X Tasks activityTracker/CRX/Mohsin/9LPADUZ7300KH2C3LAQ7XUCU X Polls activityTracker/CRX/Mohsin/9LPADV16442JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV35584IP2C3LAQ7XUCUX	Passed

```
Database Connection Status:
---------------------------------
RDS MySQL connection failed.
RDS Oracle connection success.
Default MySQL connection success.
```

Figure 7.9.: SQL statements evaluation results.

8. Outcome and Future Work

In this master's thesis, we have originated concepts and implementation strategies for a Data Access Layer (DAL) with typical application architecture model in order to enable access to Cloud data. However, implementing a DAL requires proper adaptations for accessing different Cloud data store services. The main contribution of this thesis work is to investigate the adaptations of SQL statements required for accessing various Relational Database Management Systems RDMS data store services in the Cloud. In order to determine the necessary adaptations, a JUnit test [Mil05] application has been implemented where JUnit performs a set of SQL statements execution in selected data store services. The SQL test cases are designed based on SQL: 2003 standard [Ame03]. Moreover, to find out the required adaptations, SQL test cases have been examined on different RDMS data store services including MySQL, Oracle, and PostgreSQL in Amazon EC2, Amazon RDS, and Amazon VPC Cloud. The JUnit test results of every test scenarios are described in Chapter 4. In Chapter 2, we have presented relevant fundamentals, such as Cloud computing, Cloud data hosting solutions [SKLU11], different Cloud services focusing on RDMS, ISO standard SQL taxonomy [Ame03], and a CRM application (i.e. openCRX) [CRM12]. Furthermore, we have investigated previous works related to implement a data access layer and multiple Cloud services accesses within an application. An overview of related works is presented in Chapter 3. Nevertheless, after investigating the previous works of accessing multiple Cloud services in a single application system, we have introduces a concept for adapting SQL statements in a data access layer in Chapter 5.

Commonly, a typical application is built today using three layer architecture model [Eck95], however introducing a data access layer, now the application will lead a four layer architecture model consisting of a presentation layer, a business logic layer, a data access layer, and a database layer. Considering a four layer application, a conceptual application architecture model has been described in Section 5.1. While comparing with typical application model, the presentation layer and business logic layer are remaining identical, however the data layer has subdivided into two layers such as data access layer (DAL) and database layer (DBL). However, implementing a DAL within application system, the business logic has to be adapted accordingly, as the data access layer is only considered component, which is communicating to the data store services.

The fundamental concept of implementing a data access layer is that while typical database server is not able to manage a high amount of data and computations, an application may move the data layer to Cloud data store services for taking the advantages of Cloud computing. Additionally, in this work we have been paying attention to accessing different data store services with an application, as individual Cloud services have been established with unique philosophy and computations goal. Therefore, the application may take advantages of accessing different data store services for different computation purposes. This master's thesis has conceived developing a Java based application called SQL Evaluation tool, which realizes the logic to adapt SQL statements in a prototypical implementation of a SQL testing tool for discovering the adaptations found in Chapter 4. An use-case analysis has brought out the evaluation concept for SQL statement adaptations. Furthermore, we have conceived implementing an extension mechanism in the application system in order to ensure the extension of data access functionality in runtime. Together, these concepts targeted a new generation of application system.

These concepts have lead to a system design in Chapter 6 that describes a complete software system, a Java application tool for evaluating SQL statements adaptations, where application accessing different Cloud data store services. Additionally, an extension mechanism [Som] has been designed for enhancing the data access functionality to the system. In extension mechanism the application usually load the different data access classes depending on different data and computations needed. In addition, the application system has designed based on component-based applications [Aß03], which follows a MVC design pattern. However, the data access layer has designed with a number of individual independent data access classes for every different data service providers. So that, the application can access several data store services simultaneously in runtime and thus ensures data isolation of accessing heterogeneous Cloud. To confirm this facility the application has designed with multi-threading method execution by using Java SwingWorker [Orab] wrapper implementation. Moreover, the application has taken care of accessing both public and private Cloud data. As some of the Cloud computing infrastructures (i.e. Amazon EC2) have to be accessed through user key based authentication, this system has also taken care of implementing SSH tunneling with port forwarding using a Java library called JSch (Java Secure Channel) [JCr]. The detail implementation of SSH tunneling session is described in Section 7.1.5.

8.1. Future Works

Currently, this system does not consider supporting for non-relational data store services, as this master's thesis focused on only relational database systems and we investigated the adaptations of SQL statements required for several RDMS data store services. However, the system has designed to handle the data access functionalities for as many as the data store services needed to access by the application system. The concept is that whenever an application wants to access a new data store service, the application will add a new data access class with necessary SQL statement adaptations in the data access layer and load it to the system extension mechanism and then the system will automatically enable data access to the target Cloud data store service. In addition, we have investigated a NoSQL database system called Amazon SimpleDB [SDB], which performs faster computations comparing with a traditional one and provide solutions on strict requirements of a relational database system. Therefore, in future, this system can be extended for non-relational database support as well. For example, NoSQL based data store services such as Amazon SimpleDB, Amazon DynamoDB, etc.

While implementing data access layer for both relational and non-relational data services supports, one has to implement a prototypical reference of the adapters (Apache ServiceMix [ASM] middleware, an open source ESB) which would extend the data access layer in order to enable access to the database layer hosted in the Cloud using storage and database services of established Cloud providers. The ESB is one of the achievable building blocks of a Platform-as-a-Service (PaaS), where ESB provides integration capabilities for Service-Oriented Architectures (SOA). However, ensuring this possibility, the ESB requires adaptation to enable access Cloud data store services. The concept is that the required adaptations will be stored in service engine inside the ESB and with the knowledge of stored adaptations, the ESB will route each appropriate message (i.e. SQL statement) to the target data store service, while application request to execute a query.

Moreover, at present this system does not ensure the performance of accessing Cloud data store services. But we have realized that accessing multiple Cloud data simultaneously it takes more time, while comparing with traditional database accesses. Therefore, the performance should be kept in mind when extending an ESB Apache ServiceMix to be used as DAL.

Appendix A.

Results of SQL Statements Investigation

This chapter illustrates the experimental results of SQL statements testing, which is performed by using JUnit test. The detail procedure of test setup and overview scenario of SQL statement testing for every selected data store services (See Table 4.1) are described in Chapter 4. When a test case is failed, the results describes why it failed and how to adapt the statement to enable access Cloud data store services. Furthermore, a possible solution is proposed in case a test case is failed.

A.1. JUnit Test Results for Default Database

This section describes the experimental results of JUnit test for SQL statement testing on default database. The detailed tentative procedure is described in Section 4.3.1.

Table A.1.: JUnit test for default SQL statement testing on local machine.

No.	Expected Result	Actual Result	Outcomes
01	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admin-s/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfigu	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admin-s/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfigu	Passed
02	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	Passed
03	account/CRX/Standard/admin-Standard 3	account/CRX/Standard/admin-Standard 3	Passed

Continued on next page

Table A.1: JUnit test for default SQL statement testing on local machine – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
04	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	Passed
05	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	Passed
06	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	Passed
07	calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	Passed
08	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	Passed
09	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd Fea-tureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd Fea-tureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
10	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd Fea-tureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd Fea-tureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed

Continued on next page

Table A.1: JUnit test for default SQL statement testing on local machine – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
11	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed
12	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
13	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed
14	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
15	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
16	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
17	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
18	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
19	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
20	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed

Continued on next page

Table A.1: JUnit test for default SQL statement testing on local machine – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
21	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
22	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
23	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
24	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
25	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
26	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
27	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
28	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	Passed
29	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	Passed
30	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	Passed
31	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	Passed
32	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Passed

Continued on next page

Table A.1: JUnit test for default SQL statement testing on local machine – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
33	Unknown column 'Persons.LastName' in 'field list' *Adaptation:* SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id UNION SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id *Query Result:* Hansen Ola 22456 Hansen Ola 24562 Svendson Tove null Pettersen Kari 77895 Pettersen Kari 44678 null null 34764	Unknown column 'Persons.LastName' in 'field list' *Adaptation:* SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id UNION SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id *Query Result:* Hansen Ola 22456 Hansen Ola 24562 Svendson Tove null Pettersen Kari 77895 Pettersen Kari 44678 null null 34764	Failed

A.2. JUnit Test Results for Oracle in Amazon EC2

This section describes the experimental results of JUnit test for SQL statement testing on Oracle data store service in Amazon EC2. The detailed tentative procedure is described in Section 4.3.3.

Table A.2.: JUnit test for Oracle statement testing on Amazon EC2.

No.	Expected Result	Actual Result	Outcomes
01	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfigu	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38 org:opencrx:kernel:admin1:ComponentConfigu	Failed
02	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	Passed
03	account/CRX/Standard/admin-Standard 3	account/CRX/Standard/admin-Standard 3	Passed

Continued on next page

Table A.2: JUnit test for Oracle statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
04	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	Passed
05	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	Passed
06	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3 calendar/CRX/Standard/DefaultBusine ss-Calendar 6 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6	Failed
07	calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	Error: ORA-00979: not a GROUP BY expression Adaptation: SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING AVG (access_level_browse * access_level_delete) > 4 calendar/CRX/Standard/DefaultBusine ssCalendar 6 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6	Failed
08	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	Passed
09	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed

Continued on next page

Table A.2: JUnit test for Oracle statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
10	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
11	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed
12	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
13	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed
14	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed

Continued on next page

Table A.2: JUnit test for Oracle statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
15	Your SQL query has been executed successfully	Error: ORA-00922: missing or invalid option Adaptation: CREATE OR REPLACE VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address,addr.authority AS account_ FROM oocke1_address addr => Your SQL query has been executed successfully	Failed
16	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
17	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
18	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
19	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
20	Your SQL query has been executed successfully	Error: ORA-00907: missing right parenthesis Adaptation: CREATE TABLE test_table1 (id NUMBER(10, 0) NOT NULL, name VARCHAR2(250) NOT NULL, age NUMBER(10, 0), marks NUMBER(19, 0), PRIMARY KEY (id)) => Your SQL query has been executed successfully	Failed
21	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
22	Your SQL query has been executed successfully	Error: ORA-00904: : invalid identifier Adaptation: ALTER TABLE test_table1 ADD birthday date => Your SQL query has been executed successfully	Failed
23	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
24	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
25	Your SQL query has been executed successfully	Error: ORA-00933: SQL command not properly ended Adaptation: DROP INDEX idx => Your SQL query has been executed successfully	Failed
26	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
27	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed

Continued on next page

Table A.2: JUnit test for Oracle statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
28	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	Passed
29	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	3 Guest, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private	Failed
30	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	Passed
31	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	Passed
32	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Passed
33	Unknown column 'Persons.LastName' in 'field list' *Adaptation:* SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id UNION SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id *Query Result:* Hansen Ola 22456 Hansen Ola 24562 Svendson Tove null Pettersen Kari 77895 Pettersen Kari 44678 null null 34764	Hansen Ola 22456 Hansen Ola 24562 Pettersen Kari 44678 Pettersen Kari 77895 Svendson Tove null null null 34764	Failed

A.3. JUnit Test Results for PostgreSQL in Amazon EC2

This section describes the experimental results of JUnit test for SQL statement testing on PostgreSQL data store service in Amazon EC2. The detailed tentative procedure is described in Section 4.3.4.

Table A.3.: JUnit test for PostgreSQL statement testing on Amazon EC2.

No.	Expected Result	Actual Result	Outcomes
01	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfigu	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38 org:opencrx:kernel:admin1:ComponentConfigu	Failed
02	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeMf4vj8cA 3 account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3	Failed
03	account/CRX/Standard/admin-Standard 3	account/CRX/Standard/admin-Standard 3	Passed
04	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	activityTracker/CRX/Mohsin/9LPADUT9ZN UN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90R WM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81V YLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ730 0KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV1644 2JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV3558 4IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546C 6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737G 8GX2C3LAQ7XUCUX Public	Passed
05	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	Passed
06	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000000000000000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000000000000000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000000000000000	Failed

Continued on next page

Table A.3: JUnit test for PostgreSQL statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
07	calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	ERROR: column "oocke1_calendar.access_-level_browse" must appear in the GROUP BY clause or be used in an aggregate function Adaptation: SELECT object_id, AVG (access_level_browse * access_level_delete) AS access_level_browse FROM oocke1_calendar GROUP BY object_id HAVING AVG (access_level_browse * access_level_delete) > 4 *Query Result:* calendar/CRX/Standard/DefaultBusinessCaler 6.0000000000000000 calendar/CRX/Standard/e5uf4EIBEd29BeMf4v 6.0000000000000000	Failed
08	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	Passed
09	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
10	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
11	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed

Continued on next page

Table A.3: JUnit test for PostgreSQL statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
12	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Passed
13	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Passed
14	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
15	Your SQL query has been executed successfully	Error: ERROR: syntax error at or near "(" Assuming an equivalent adaptation: CREATE OR REPLACE VIEW oocke1_join_accthasassaddr (assigned_address,account_) AS SELECT addr.object_id AS assigned_address, addr.authority AS account_ FROM oocke1_address addr Error: ERROR: cannot change name of view column "account" to "account_" Adaptation: CREATE OR REPLACE VIEW oocke1_join_accthasassaddr (assigned_address,account) AS SELECT addr.object_id AS assigned_address, addr.authority AS account FROM oocke1_address addr => Your SQL query has been executed successfully	Failed
16	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
17	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
18	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
19	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed

Continued on next page

Table A.3: JUnit test for PostgreSQL statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
20	Your SQL query has been executed successfully	Error: ERROR: syntax error at or near "UNSIGNED" Adaptation: CREATE TABLE public.test_table1 (id serial NOT NULL, name varchar(250) NOT NULL, age bigint, marks bigint, PRIMARY KEY (id)) => Your SQL query has been executed successfully	Failed
21	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
22	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
23	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
24	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
25	Your SQL query has been executed successfully	Error: ERROR: syntax error at or near "ON" Adaptation: DROP INDEX idx Your SQL query has been executed successfully	Failed
26	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
27	Your SQL query has been executed successfully	Your SQL query has been executed successfully	Passed
28	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest, 3 Guest, 3 admin-Standard Private 3 guest Private 3 admin-Mohsin, 3 Guest,	Failed
29	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	Passed
30	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	Passed
31	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 admin-Mohsin, 3 Guest,	Failed
32	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Svendson Tove 22456 Svendson Tove 24562 Pettersen Kari 22456 Pettersen Kari 24562	Failed

Continued on next page

Table A.3: JUnit test for PostgreSQL statement testing on Amazon EC2 – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
33	Unknown column 'Persons.LastName' in 'field list' *Adaptation:* (SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id) UNION (SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id) *Query Result:* Hansen Ola 22456 Hansen Ola 24562 Svendson Tove null Pettersen Kari 77895 Pettersen Kari 44678 null null 34764	Hansen Ola 22456 Hansen Ola 24562 Pettersen Kari 77895 Pettersen Kari 44678 Svendson Tove Null null null 34764	Failed

A.4. JUnit Test Results for Amazon SimpleDB

This section describes the experimental results of JUnit test for SQL statement testing on Amazon SimpleDB data store service. The detailed tentative procedure is described in Section 4.3.8.

Table A.4.: JUnit test for SQL statement testing on Amazon SimpleDB.

No.	Expected Result	Actual Result	Outcomes
01	componentConfig/CRX/Root/WorkflowContr 3 2 2 2008-06-24 18:06:38.0 1 null 1 WorkflowController 2 admins/CRX/Root 2008-06-24 18:06:38.0 org:opencrx:kernel:admin1:ComponentConfigu	componentConfig/CRX/Root/WorkflowContr 2 2008-06-24 18:06:38.0 2 WorkflowController 2008-06-24 18:06:38.0 2 org:opencrx:kernel:admin1:ComponentConfigu 3 admins/CRX/Root 1 1	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
02	account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3	Return the same results as we achieve after adaptations. However, uploading data table on SimpleDB, the column OBJECT_ID becomes itemName(), a key for the entire row. The itemName() column is automatically selected while executing the query. Adaptation: SELECT ACCESS_LEVEL_BROWSE FROM oocke1_account account/CRX/Mohsin/admin-Mohsin 3 account/CRX/Standard/NI7XwEIBEd29BeM f4vj8cA 3 account/CRX/Standard/admin-Standard 3 account/CRX/Standard/admin-Standard Private 3 account/CRX/Standard/guest Private 3	Failed
03	account/CRX/Standard/admin-Standard 3	Adaptation: SELECT ACCESS_LEVEL_BROWSE FROM oocke1_account WHERE itemName()='account/CRX/Standard/admin-Standard' Result: account/CRX/Standard/admin-Standard 3	Failed
04	activityTracker/CRX/Mohsin/9LPADUT9Z NUN52C3LAQ7XUCUX Bugs + Features activityTracker/CRX/Mohsin/9LPADUV90 RWM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUX81 VYLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUZ73 00KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADV164 42JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADV355 84IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV546 C6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV737 G8GX2C3LAQ7XUCUX Public	Client error : The specified query expression syntax is not valid. Invalid because neither itemName() nor NAME is constrained by a predicate in the where clause. SELECT NAME FROM oocke1_activitygroup WHERE P$$PARENT='activities/CRX/Mohsin' ORDER BY NAME DESC Client error : Invalid sort expression. The sort attribute must be present in at least one of the predicates, and the predicate cannot contain the is null operator. SELECT NAME FROM oocke1_activitygroup WHERE P$$PARENT='activities/CRX/Mohsin' ORDER BY P$$PARENT DESC Result: activityTracker/CRX/Mohsin/9LPADV737 G8GX2C3LAQ7XUCUX Public activityTracker/CRX/Mohsin/9LPADV546 C6HT2C3LAQ7XUCUX Phone Calls activityTracker/CRX/Mohsin/9LPADV355 84IP2C3LAQ7XUCUX Meetings activityTracker/CRX/Mohsin/9LPADV164 42JL2C3LAQ7XUCUX Meeting Rooms activityTracker/CRX/Mohsin/9LPADUZ73 00KH2C3LAQ7XUCUX Polls activityTracker/CRX/Mohsin/9LPADUX81 VYLD2C3LAQ7XUCUX Tasks activityTracker/CRX/Mohsin/9LPADUV90 RWM92C3LAQ7XUCUX E-Mails activityTracker/CRX/Mohsin/9LPADUT9Z NUN52C3LAQ7XUCUX Bugs + Features	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
05	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX Default Business Calendar calendar/CRX/Standard/DefaultBusine ss-Calendar Default Working Calendar calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA Default Business Calendar	Client error : The specified query expression syntax is not valid. DISTINCT feature is already covered in this scope, as the itemName() keys are unique	Failed
06	calendar/CRX/Mohsin/9LPADU3ML74YP2C 3LAQ7XUCUX 3.0000 calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	Client error : The specified query expression syntax is not valid. The query can't be implemented in this scope.	Failed
07	calendar/CRX/Standard/DefaultBusine ss-Calendar 6.0000 calendar/CRX/Standard/e5uf4EIBEd29B eMf4vj8cA 6.0000	Client error : The specified query expression syntax is not valid. The query can't be implemented in this scope.	Failed
08	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE IDX BETWEEN '0' AND '1' The default table has to be adapted because the column OBJECT_ID contain same id more than once. But SimpleDB can only allow unique key. Result: activityProcess/CRX/Mohsin/9LPADSS 9UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJ WA2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAn dFeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DY EIBEd29BeMf4vj8cA Standard:admin-Standard.User activityProcess/CRX/Mohsin/9LPADSS 9UHTK12C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJ WA2L7L2C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAn dFeatureRequestProcess1 Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DY EIBEd29BeMf4vj8cA1 Standard:admin-Standard.User	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
09	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE IDX IN('0', '2') The default table has to be adapted because the column OBJECT_ID contain same id more than once. But SimpleDB can only allow unique key. Result: activityProcess/CRX/Mohsin/9LPADSS 9UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJ WA2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAn dFeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DY EIBEd29BeMf4vj8cA Standard:admin-Standard.User activityProcess/CRX/Mohsin/9LPADSS 9UHTK12C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJ WA2L7L2C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAn dFeatureRequestProcess1 Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DY EIBEd29BeMf4vj8cA1 Standard:admin-Standard.User	Failed
10	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE ((IDX > '0') AND (IDX < '2')) The default table has to be adapted because the column OBJECT_ID contain same id more than once. But SimpleDB can only allow unique key. Result: activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAnd FeatureRequestProcess1 Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA1 Standard:admin-Standard.User	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
11	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE ((IDX < ’1’) OR (IDX > ’1’)) The default table has to be adapted because the column OBJECT_ID contain same id more than once. But SimpleDB can only allow unique key. Result: activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX2 Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX2 Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAnd FeatureRequestProcess2 Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA2 Standard:admin-Standard.User	Failed
12	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Users activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:Administrators activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:Administrators	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE (IDX = ’1’) The default table has to be adapted because the column OBJECT_ID contain same id more than once. But SimpleDB can only allow unique key. Result: activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX1 Mohsin:admin-Mohsin.User activityProcess/CRX/Standard/BugAnd FeatureRequestProcess1 Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA1 Standard:admin-Standard.User	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
13	activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADSS9 UHTK12C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:admin-Mohsin.User activityProcess/CRX/Mohsin/9LPADTJW A2L7L2C3LAQ7XUCUX Mohsin:Administrators activityProcess/CRX/Standard/BugAnd FeatureRequestProcess Standard:admin-Standard.User activityProcess/CRX/Standard/eW3DYE IBEd29BeMf4vj8cA Standard:admin-Standard.User	Client error : The specified query expression syntax is not valid. Adaptation: SELECT OWNER FROM oocke1_activityprocess_ WHERE (IDX <> ’1’) The query can’t be implemented in this scope.	Failed
14	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. SimpleDB lacks supporting SQL VIEW functionality.	Passed
15	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. SimpleDB lacks supporting SQL VIEW functionality.	Failed
16	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. SimpleDB lacks supporting SQL VIEW functionality.	Failed
17	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Inserted item (data) in SimpleDB data storage has to be done using SimpleDB API call.	Failed
18	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Update item (data) in SimpleDB data storage has to be done using SimpleDB API call.	Failed
19	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Update item (data) in SimpleDB data storage has to be done using SimpleDB API call.	Failed
20	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Create domain in SimpleDB data storage has to be done using SimpleDB API call. First we have to create a domain, and then have to create necessary Attributes.	Failed
21	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Copy a domain value into another similar domain has to be done using SimpleDB API call. First we have to retrieve the item data using select operation from the domain we want to copy and then store to a destination domain.	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
22	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Adding an Attribute in a domain has to be done using SimpleDB API call.	Failed
23	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Deleting an Attribute from a domain has to be done using SimpleDB API call.	Failed
24	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ INDEX feature.	Failed
25	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ INDEX feature.	Failed
26	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. SQL TRUNCATE effect can be implemented in application layer using SimpleDB API call.	Failed
27	Your SQL query has been executed successfully	Client error : The specified query expression syntax is not valid. Delete a domain in SimpleDB data storage has to be done using SimpleDB API call.	Failed
28	3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard, 3 admin-Standard Private 3 admin-Standard Private 3 guest Private 3 guest Private 3 Guest, 3 Guest, 3 Guest,	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ JOIN feature.	Failed
29	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 Guest, 3 guest Private	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ INNER JOIN feature.	Failed
30	3 admin-Mohsin, 3 admin-Standard, 3 Guest,	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ LEFT JOIN feature.	Failed
31	3 admin-Mohsin, 3 admin-Standard, 3 admin-Standard Private 3 guest Private 3 Guest,	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ RIGHT JOIN feature.	Failed
32	Svendson Tove 22456 Pettersen Kari 22456 Svendson Tove 24562 Pettersen Kari 24562	Client error : The specified query expression syntax is not valid. This query can't be implemented in SimpleDB data storage in this scope.	Failed

Continued on next page

Table A.4: JUnit test for SQL statement testing on Amazon SimpleDB – (continued from previous page)

No.	Expected Result	Actual Result	Outcomes
33	Unknown column 'Persons.LastName' in 'field list' *Adaptation:* SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons LEFT JOIN Orders ON Persons.P_-Id=Orders.P_Id UNION SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN Orders ON Persons.P_-Id=Orders.P_Id *Query Result:* Hansen Ola 22456 Hansen Ola 24562 Svendson Tove null Pettersen Kari 77895 Pettersen Kari 44678 null null 34764	Client error : The specified query expression syntax is not valid. The query can't be implemented, as SimpleDB lacks supporting SLQ FULL JOIN feature.	Failed

Bibliography

[Aß03] U. Aßmann. *Invasive Software Composition*. Springer, 2003.

[AAB$^+$11] M. Allalouf, A. Averbuch, L. Bonelli, P. Brand, M. Dao, A. Eckert, M. C. Jaeger, H. Kolodner, E. Levy, J. M. Lopez, M. Lorenz, A. Manieri, A. R. Messina, M. Neumann, K. Ramasamybalraj, E. Salant, A. Shulman-Peleg, F. Solsvik, and X. Su. *Data Access Layer: Design and Open Spec*. VISION Cloud, 2011. VISION Cloud: Virtualized Storage Services Foundation for the Future Internet, `http://visioncloud.eu/content.php?s=248,389`.

[AKM$^+$12] A. Amies, S. Kusturica, M. M. (Max), Q. G. Tong, and Y. S. Wang. Use the Apache Libcloud Python API to manage resources on IBM SmartCloud Enterprise, 2012. `http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/5bfbf50420a468df86257a14006248f6/$FILE/sce_libcloud_june2012.pdf`.

[Ama09] Amazon Web Services. Amazon SimpleDB Developer Guide, 2009. API Version, `http://awsdocs.s3.amazonaws.com/SDB/latest/sdb-dg.pdf`.

[Ama12] Amazon Web Services. AWS Case Study: Alexa, 2012. Alexa: The Web Information Company, `http://aws.amazon.com/en/solutions/case-studies/alexa/`.

[Ame03] American National Standards Institute, ANSI. Database languages SQL - Part 2: Foundation(SQL/Foundation). International Organization for Standardization, ISO, 2003.

[Ana10] S. Anand. Netflix's Transition to High-Availability Storage Systems, 2010. Netflix's transition to AWS SimpleDB and S3, `http://aws.amazon.com/articles/3662538304152045`.

[Apa] Apache Software Foundation. Apache Libcloud: A Unified Interface to the Cloud. A standard Python library, `http://libcloud.apache.org/index.html`.

[Apa11] Apache Software Foundation. Deltacloud: Many Cloud. One API. No problems, 2011. Deltacoud API, `http://deltacloud.apache.org/index.html`.

[ASM] The Apache Software Foundation. Apache ServiceMix. `http://servicemix.apache.org/`.

[Bak10] G. Baker. Why CIOs Should Shift from Capex to Opex, 2010. `www.cioupdate.com/budgets/article.php/3905476/Why-CIOs-Should-Shift-from-Capex-to-Opex.htm`.

[Ber10] W. W. Berry. Move Access Data to the Cloud, 2010. Microsoft SQL Server Migration Assistant for Access, `http://social.technet.microsoft.com/wiki/contents/articles/1575.move-access-data-to-the-cloud.aspx`.

[CAP] The Open Group. Building Return on Investment from Cloud Computing: Discussion - Financial Value Perspective of Moving from CAPEX to OPEX and Pay-as-you-go. `http://www.opengroup.org/cloud/whitepapers/ccroi/disc1.htm`.

[Che12] J. Chen. Bookkeeping With Libcloud, 2012. A Simple Example of Libcloud, `http://undertitled.com/2012/04/11/bookkeeping-with-libcloud.html`.

[CRI12] CRIXP Corp. openCRX Server Installation, 2012. `http://www.opencrx.org/opencrx/2.9/installerServer/installer_openCRX_server.html`.

[CRM12] CRIXP Corp. The Professional Enterprise Class Open Source CRM Solution, 2012. openCRX v 2.9.1, `http://www.opencrx.org/`.

[Dar05] H. Darwen. More on Relational Algebra versus Claculus, 2005. Database Ebunkings, `http://www.dbdebunk.com/page/page/1897740.htm`.

[DBC] DB Gurus Pvt. Ltd. A Comparison of the Common Database Systems. Professional Database Solution, `http://www.dbgurus.com.au/Files/A%20Comparison%20of%20the%20Common%20Database%20Systems.pdf`.

[DEF$^+$08] J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, and A. Koschel. *Systemarchitekturen für Verteilte Anwendungen - Client-Server, Multi-Tier, SOA, Event-Driven Architectures, P2P, Grid, Web 2.0.* Hanser Verlag, 2008.

[EBS] Amazon Web Services. Amazon Elastic Block Store (EBS). `http://aws.amazon.com/ebs/`.

[EC2] Amazon Web Services. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[Eck95] W. W. Eckerson. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*, 10, 1995.

[Fis10] S. Fisher. Introducing Database, 2010. `http://blog.database.com/blog/2010/12/06/introducing-database-com-2/`.

[Gee05] D. Geer. Eclipse Becomes the Dominant Java IDE. volume Volume 38. IEEE Computer Society., 2005.

[INT] w3schools.com. The SQL SELECT INTO Statement. `http://www.w3schools.com/sql/sql_select_into.asp`.

[Ise06] P. Isenhour. GridBagLayout Example: A Simple Form Layout, 2006. FormUtility, `http://javatechniques.com/blog/gridbaglayout-example-a-simple-form-layout/`.

[jcl11a] jclouds, Inc. BlobStore Guide, 2011. BlobStore API, http://www.jclouds.org/documentation/userguide/blobstore-guide/.

[jcl11b] jclouds, Inc. JClouds Overview, 2011. Multi-cloud Framework, http://www.jclouds.org/.

[jcl11c] jclouds, Inc. User Guide: How to Use the Compute API and Tools, 2011. Compute API, http://www.jclouds.org/documentation/userguide/compute/.

[JCr] JCraft, Inc. JSch - Java Secure Channel. SSH Port Forwarding, http://www.jcraft.com/jsch/.

[Lor10] M. Lorenz. Vision Cloud: The Fact Sheet, 2010. http://www.visioncloud.eu/content.php?s=30,47.

[Man11] A. Mandelbaum. PostgreSQL vs. MySQL: Which Is the Best Open Source Database?, 2011. http://www.openlogic.com/wazi/bid/188125/PostgreSQL-vs-MySQL-Which-Is-the-Best-Open-Source-Database.

[MG11] P. Mell and T. Grance. Cloud Computing Definition. National Institute of Standards and Technology, 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[Mic12] Microsoft Corporation. SQL Azure Migration Wizard v3.9 & v4.0.3, 2012. SQLAzureMW Documentation, http://sqlazuremw.codeplex.com/.

[Mil05] A. J. Mills. JUnit Testing Utility Tutorial, 2005. https://supportweb.cs.bham.ac.uk/docs/tutorials/docsystem/build/tutorials/junit/junit.html.

[MWA10] Amazon Web Services. Migration Scenario: Migrating Web Applications to the AWS Cloud, 2010. Web Application Architecture, http://d36cz9buwru1tt.cloudfront.net/CloudMigration-scenario-wep-app.pdf.

[MYS] Oracle Corporation. MySQL Database. http://www.mysql.com/.

[MyS09] Amazon Web Services. Migrating from MySQL to Amazon SimpleDB, 2009. http://aws.amazon.com/code/Amazon-SimpleDB/2996.

[Ocr12] Geeknet, Inc. How to Migrate an Existing openCRX Database, 2012. http://sourceforge.net/apps/trac/opencrx/wiki/Sdk29.DatabaseMigration.

[ORAa] Oracle Corporation. Oracle Database. http://www.oracle.com/index.html.

[Orab] Oracle and/or its affiliates. Worker Threads and SwingWorker. Concurrency in Swing, http://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html.

[Pat06] T. Patton. Use the Data Access Layer to Simplify Architecture, 2006. Implementing a Data Access Layer, http://www.techrepublic.com/article/use-the-data-access-layer-to-simplify-architecture/6078128.

[PSQ] The PostgreSQL Global Development Group. PostgreSQL. http://www.postgresql.org/.

[Qui12] QuinStreet Inc. Cloud Database, 2012. http://www.webopedia.com/TERM/C/cloud_database.html.

[RDS] Amazon Web Services. Amazon Relational Database Service (Amazon RDS). http://aws.amazon.com/rds/.

[RG02] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Science Engineering, third edition edition, 2002.

[RW12] L. Rath-Wiggins. Activity III - Data Access Layer for Storage Cloud, 2012. VISION Cloud, http://www.visioncloud.eu/content.php?s=248,389.

[SBK+12] S. Strauch, U. Breitenbücher, O. Kopp, F. Leymann, and T. Unger. Cloud Data Patterns for Confidentiality. In *Proceedings of the 2nd International Conference on Cloud Computing and Service Science (CLOSER'12)*. SciTePress, 2012.

[SDB] Amazon Web Services. Amazon SimpleDB. http://aws.amazon.com/simpledb/.

[SKLU11] S. Strauch, O. Kopp, F. Leymann, and T. Unger. A Taxonomy for Cloud Data Hosting Solutions. In *Proceedings of the IEEE International Conference on Cloud and Green Computing, CGC 2011*. IEEE Computer Society, 2011.

[Som] A. Sommerer. The Java Extension Mechanism. Oracle: The Java Tutorials, http://docs.oracle.com/javase/tutorial/ext/index.html.

[Str12] S. Strauch. A Novel Architecture and Methodology for Migration of the Data Layer to the Cloud, 2012. Institute of Architecture of Application Systems, University of Stuttgart, http://www.summersoc.eu/wp-content/uploads/2012/07/Steve-Strauch_poster_PhD-topic-SummerSOC-2012.pdf.

[Tao01] L. Tao. Shifting Paradigms with the Application Service Provider Model. *Computer*, 34, 2001.

[Tea12] P. Team. Database Abstraction Layer, 2012. MySQL Is the World's Most Used RDBMS, http://phalconphp.com/documentation/db.

[Vil12] P. M. Villari. Storage Cloud Challenges: VISION Cloud EU Project. In *Cloud Standards Customer Council, CSCC Quarterly Meeting: Cloud in the Public Sector*. VISION Cloud, 2012.

[VPC] Amazon Web Services. Amazon Virtual Private Cloud (Amazon VPC). http://aws.amazon.com/vpc/.

[VPC11] Amazon Web Services. Get Started with Amazon VPC (Documentation), 2011. Amazon VPC API version, http://docs.amazonwebservices.com/AmazonVPC/latest/GettingStartedGuide/GetStarted.html.

[W3S] w3schools.com. SQL Tutorial. http://www.w3schools.com/sql/default.asp.

All links were last followed on August 30, 2012.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Stuttgart, August 31, 2012 ――――――――――――

(S.M. Mohsin Reza)