Assignment 03

Problem Definition

Design and develop SQL DDL statements to demonstrate the use of SQL objects such as table, views, index using client-server database(two-tier) using the following tables:

Prerequisites

- Knowledge of MySQL databases
- Using JDBC to perform creation and deletion of views and indices

Learning Objectives

- To learn RDBMS(Relational Database Management Systems)
- To learn the basic operations in MySQL
- To understand the concept of two-tier architecture
- To use JDBC to connect the front end to the back end

Learning Outcomes

- Understanding and implementing RDBMS concepts by creating a database and performing various operations on it
- Establishing two-tier architecture using JDBC

Software and Hardware Requirements

- Fedora 20
- JDK7
- mysql-connector-java 5.1.31
- MariaDB 10.0.12
- Eclipse platform

Mathematical Model

Let S be the system set of solution for the given problem statement such that
S={s, e , X, Y, f_{me}, DD, NDD ,f_s, shared_mem | ϕ }
where;
s=start state
Initially,
s=db=$\emptyset$
where db is the database
e=end state

db={X_1, X_2, ..., X_n} $X_i \mid 0 < i <= 6$
$\forall$ X_i ϵ db $\Rightarrow$ table in the database

X_1 = employees = (empid, fname, lname, email, phone, hiredate, jobid, sal, depid)
primary key(empid)
empid = employee id ϵ I^+
fname = first name ϵ [a-z][A-Z]
lname = last name ϵ [a-z][A-Z]
email = email id (example@xyz.com)
phone = phone number ϵ [0-9]
hiredate = date of hiring ϵ date(yyyy-mm-dd)
sal = salary ϵ R^+
jobid, depid are foreign keys

X_2 = work = (empid, manid)
primary key(empid,manid)
empid, manid are foreign keys

X_3 = depts = (depid, deptname, locid)
primary key(depid)
depid = department id ϵ I^+
deptname = first name ϵ [a-z][A-Z]
locid is a foreign key

X_4 = jobs = (jobid, title, minsal, maxsal)
primary key(jobid)
jobid = job id ϵ I^+
title = job title ϵ [a-z][A-Z]
minsal = minimum salary ϵ R^+

maxsal = maximum salary ϵ R^+

X_5 = locations = (locid, street, city, state, country)
primary key(locid)
locid = location id ϵ I^+
street, city, state, country = address ϵ [a-z][A-Z]

X_6 = jobhistory = (empid, hiredate, leavedate, sal, jobid, depid)
primary key(empid,hiredate,leavedate)
hiredate = date of hiring in previous job ϵ date(yyyy-mm-dd)
leavedate = date of leaving previous job ϵ date(yyyy-mm-dd)
sal = salary at previous job ϵ R^+
empid, jobid, depid are foreign keys

X is the input set i.e. rows of the tables in the database
X=$\{x_1, x_2, ..., x_n\}$ $x_i \mid 0 < i <= 6$
$\forall$ x_i ϵ X; x_i corresponds to a row in the table X_i ϵ db
Y is the output set; queries executed successfully on the database and result displayed
Y=$\{Y_1, Y_2, ..., Y_n\}$ $Y_i \mid 0 < i <= n$ where each Y_i is a table

f_{me} = main function
f_s = set of all functions
$f_s = \{f_{cview}, f_{cindex}, f_{dview}, f_{dindex}\}$
f_{cview} = function to create view and display view
$f_{cview}(V_i, db) = db \cup \{V_i\}$
f_{dview} = function to drop view
$f_{dview}(V_i) : db - \{V_i\}$
f_{cindex} = function to create index
$f_{cindex}(I_i, db) \longrightarrow$ output: from database
f_{dindex} = function to drop index
$f_{dindex}(I_i) \longleftarrow I(table) - (I_i)$
DD=Deterministic Data = db
NDD=Non Deterministic Data = $\{X, x_{ij}\}$

Theory

Two-tier architecture:
A two-tier architecture is a software architecture in which a presentation layer or interface runs on a client, and a data layer or data structure gets stored on a server. Separating these two components into different locations represents a two-tier architecture, as opposed to a single-tier architecture. Other kinds of multi-tier architectures add additional layers in distributed software design.

MySQL:
MySQL is (as of March 2014) the world's second most widely used open-source RDBMS. MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack. Free-software-open source projects that require a full-featured database management system often use MySQL.

Views:
In database theory, a view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated dynamically from data in the database when access to that view is requested. Changes applied to the data in a relevant underlying table are reflected in the data shown in subsequent invocations of the view. In some NoSQL databases, views are the only way to query data.
Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table. Consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
- Views can join and simplify multiple tables into a single virtual table.
- Views can act as aggregated tables, where the database engine aggregates data (sum, average, etc.) and presents the calculated results as part of the data.

- Views can hide the complexity of data. For example, a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table.
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data that it presents. Depending on the SQL engine used, views can provide extra security.

Just as a function (in programming) can provide abstraction, so can a database view. In another parallel with functions, database users can manipulate nested views, thus one view can aggregate data from other views. Without the use of views, the normalization of databases above second normal form would become much more difficult. Views can make it easier to create lossless join decomposition.

Index:
A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

Algorithm

1. Start
2. Establish connection with MySQL server
3. Create view on tables
4. Display views
5. Drop views
6. Create index on tables
7. Drop index
8. Disconnect from the server
9. Stop

Conclusion

DBMS program for two-tier architecture is successfully implemented using JDBC. Views and indices have been learnt.