Assignment A2

Date:

TITLE	Design and develop SQL DDL statements which demonstrate the use of SQL objects, such as creation of: Table, View, Index, Sequence, Synonym
PROBLEM STATEMENT / DEFINITION	Implement DDL commands in context of view, index, sequence.
LEARNING OBJECTIVE	• To understand & implement the various DDL Commands. • To understand database concepts like view, index, sequence and synonym.
LEARNING OUTCOME	The students will be able to • Use and Implement the DDL commands like Create the tables, Alter the table structure. • Implement view, index (types of index), synonym and sequence concept.
S/W PACKAGES & HARDWARE APPARATUS USED	• MySQL • 64-bit Linux based open source OS • 8 GB RAM

CONCEPT RELATED THEORY:

DDL COMMANDS: DDL is short form of Data Definition Language, which deals with data schemas and description, of how data can reside in database

Various commands in DDL are:

Create:

Create table command defines each attribute uniquely. Each attribute has 3 mandatory things.

a. Attribute name
b. Attribute size
c. Data type

Syntax:

Create table tablename (Attribute_name attribute_datatype(size),Attribute_name attribute_datatype(size),Attribute_name attribute_datatype(size).....n)

Alter:

By using ALTER command existing table can be modified.

Adding New Columns

Syntax:

ALTER TABLE <table_name> ADD (<NewColumnName> <Data_Type>(<size>), ...n)

Dropping a Column from the Table

Syntax:

ALTER TABLE <table_name> DROP COLUMN <column_name>

*This command will drop a particular column. *

Modifying Existing Table

Syntax:

ALTER TABLE <table_name> MODIFY (<column_name><NewDataType>(<NewSize>))

Restriction on the ALTER TABLE

1. Using the ALTER TABLE clause the following tasks cannot be performed. Change the name of the table

2. Change the name of the column

3. Decrease the size of a column if table data exists

Drop:

The Drop command will destroy table along with the data enteries in it.

Syntax: Drop table <Tablename>

Truncate:

The truncate command deletes all entries existing in tables but keep the structure of table as described.

Syntax: Truncate Table <tablename>

Rename:

The rename command is used to rename the table

Syntax: Rename <OldTableName> <NewTableName>

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

CREATE VIEW view_name AS SELECT column1, column2.....FROM table_name WHERE [condition];

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

SQL > CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;

Now, you can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

SQL > SELECT * FROM CUSTOMERS_VIEW;

This would produce the following result:

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following is an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION:

CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS WHERE age IS NOT NULL WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View:

A view can be updated under certain conditions:

The SELECT clause may not contain the keyword DISTINCT.

The SELECT clause may not contain summary functions.

The SELECT clause may not contain set functions.

The SELECT clause may not contain set operators.

The SELECT clause may not contain an ORDER BY clause.

The FROM clause may not contain multiple tables.

The WHERE clause may not contain subqueries.

The query may not contain GROUP BY or HAVING.

Calculated columns may not be updated.

All NOT NULL columns from the base table must be included in the view for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update a view. Following is an example to update the age of Ramesh:

SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name='Ramesh';

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
```

```
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we can not insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Dropping Views:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

DROP VIEW view_name;

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table

DROP VIEW CUSTOMERS_VIEW;

INDEX

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, you should only create indexes on columns (and tables) that will be frequently searched against.

SQL CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name

ON table_name (column_name)

SQL CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index_name

ON table_name (column_name)

CREATE INDEX Example

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

CREATE INDEX PIndex

ON Persons (LastName)

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX PIndex

ON Persons (LastName, FirstName)

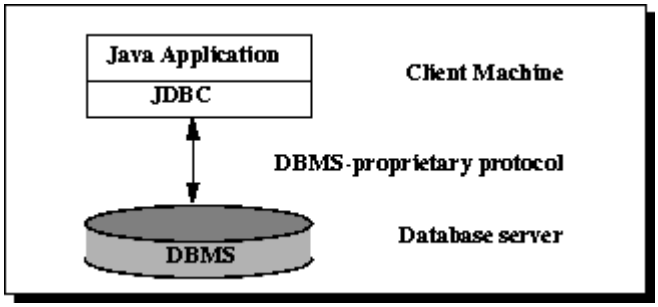
Creating Sequence

Syntax: CREATE Sequence sequence-name start with initial-value increment by increment-value maxvalue maximum-value cycle|nocycle.

TWO-TIER AND THREE-TIER PROCESSING MODELS

The JDBC API supports both two-tier and three-tier processing models for database access.

Two-tier Architecture for Data Access.



In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

JDBC :-

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following**:**

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

These steps are described in detail in the sections that follow.

Import JDBC Packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

Additionally, depending on the features being used, Oracle-supplied JDBC packages might need to be imported. For example, the following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;
```

```
import java.sql.*;
```

Load and Register the JDBC Driver

This is for establishing a communication between the JDBC program and the Oracle database. This is done by using the static `registerDriver()` method of the `DriverManager` class of the JDBC API. The following line of code does this job:

```
String url = "jdbc:mysql://192.168.5.101:3306/test";
String userName = "abc";
String password = "abc123";
DriverManager.getConnection(url, userName, password);
```

JDBC Driver Registration

For the entire Java application, the JDBC driver is registered only once per each database that needs to be accessed. This is true even when there are multiple database connections to the same data server.

Alternatively, the `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
String driver = "com.mysql.jdbc.Driver";
Class.forName(driver).newInstance();
```

However, the `forName()` method is valid for only JDK-compliant Java Virtual Machines and implicitly creates an instance of the Oracle driver, whereas the `registerDriver()` method does this explicitly.

Connecting to a Database

Once the required packages have been imported and the Oracle JDBC driver has been loaded and registered, a database connection must be established. This is done by using the `getConnection()` method of the `DriverManager` class. A call to this method creates an object instance of the `java.sql.Connection` class. The `getConnection()` requires three input parameters, namely, a connect string, a username, and a password. The connect string should specify the JDBC driver to be yes and the database instance to connect to.

The `getConnection()` method is an overloaded method that takes

- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

The following lines of code illustrate using the `getConnection()` method:

```
Connection conn = DriverManager.getConnection(URL, username, passwd);
Connection conn = DriverManager.getConnection(URL);
```

where URL, username, and passwd are of String data types.

Querying the Database

Querying the database involves two steps: first, creating a statement object to perform a query, and second, executing the query and returning a `resultSet`.

Creating a Statement Object

This is to instantiate objects that run the query against the database connected to. This is done by the `createStatement()` method of the conn Connection object created above. A call to this method creates an object instance of the Statement class. The following line of code illustrates this:

```
Statement sql_stmt = conn.createStatement();
```

Executing the Query and Returning a ResultSet

Once a Statement object has been constructed, the next step is to execute the query. This is done by using the `executeQuery()` method of the Statement object. A call to this method takes as parameter a SQL SELECT statement and returns a JDBC `ResultSet` object. The following line of code illustrates this using the `sql_stmt` object created above:

```
ResultSet rset = sql_stmt.executeQuery

("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");

Alternatively, the SQL statement can be placed in a string and
then this string passed to the executeQuery() function. This
is shown below.

String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER
BY ename";

ResultSet res = sql_stmt.executeQuery(sql);
```

Statement and ResultSet Objects

Statement and ResultSet objects open a corresponding cursor in the database for SELECT and other DML statements.

The above statement executes the SELECT statement specified in between the double quotes and stores the resulting rows in an instance of the ResultSet object named rset.

Processing the Results of a Database Query That Returns Multiple Rows

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

The first step is done using the next() method of the ResultSet object. A call to next() is executed in a loop to fetch the rows one row at a time, with each call to next() advancing the control to the next available row. The next() method returns the Boolean value true while rows are still available for fetching and returns false when all the rows have been fetched.

The second step is done by using the getXXX() methods of the JDBC rset object. Here getXXX() corresponds to the getInt(), getString() etc with XXX being replaced by a Java datatype.

The following code demonstrates the above steps:

```
while (res.next()) {

int id = res.getInt("id");

String msg = res.getString("ename");

System.out.println(id + "\t" + ename);

}
```

Specifying get() Parameters

The parameters for the getXXX() methods can be specified by position of the corresponding columns as numbers 1, 2, and so on, or by directly specifying the column names enclosed in double quotes, as getString("ename") and so on, or a combination of both.

Closing the ResultSet and Statement

Once the ResultSet and Statement objects have been used, they must be closed explicitly. This is done by calls to the close() method of the ResultSet and Statement classes. The following code illustrates this:

```
res.close();

sql_stmt.close();
```

If not closed explicitly, there are two disadvantages:

- Memory leaks can occur
- Maximum Open cursors can be exceeded

Closing the ResultSet and Statement objects frees the corresponding cursor in the database.

Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the close() method of the Connection class.

The following line of code does this:

```
conn.close();
```

DIAGRAM

INPUT:

```
CREATE TABLE `customer` (
  `cust_no` int NOT NULL AUTO_INCREMENT,
  `cust_fname` varchar(20),
  `cust_lname` varchar(20),
  `cust_company` varchar(20),
  `cust_addr` varchar(100),
  `city` varchar(20),
  `cust_phone` varchar(10),
  PRIMARY KEY (`cust_no`)
);

CREATE TABLE `orders` (
```

```
  `order_no` int NOT NULL AUTO_INCREMENT,
  `cust_no` int ,
  `isbn` bigint ,
  `qty` int ,
  `odate` date ,
  PRIMARY KEY (`order_no`),
  UNIQUE KEY `isbn` (`isbn`),
  KEY `fk_cust` (`cust_no`),
  CONSTRAINT `fk_cust` FOREIGN KEY (`cust_no`) REFERENCES `customer`
(`cust_no`) ON DELETE CASCADE
);

CREATE TABLE `book` (
  `isbn` bigint ,
  `title` varchar(20) ,
  `unit_price` int ,
  `author_no` varchar(20) ,
  `publisher_no` varchar(20) ,
  `publisher_year` year(4) ,
  UNIQUE KEY `author_no` (`author_no`),
  UNIQUE KEY `publisher_no` (`publisher_no`),
  KEY `fk_order` (`isbn`),
  CONSTRAINT `fk_order` FOREIGN KEY (`isbn`) REFERENCES `orders` (`isbn`)
ON DELETE CASCADE
);

CREATE TABLE `author` (
  `author_no` varchar(20),
  `author_name` varchar(20),
  `country` varchar(20),
  KEY `fk_book` (`author_no`),
  CONSTRAINT `fk_book` FOREIGN KEY (`author_no`) REFERENCES `book`
(`author_no`) ON DELETE CASCADE
);

CREATE TABLE `publisher` (
  `publisher_no` varchar(20) ,
  `publisher_name` varchar(20) ,
  `publisher_addr` varchar(100) ,
  `year` year(4) ,
  KEY `fk_pub` (`publisher_no`),
  CONSTRAINT `fk_pub` FOREIGN KEY (`publisher_no`) REFERENCES `book`
(`publisher_no`) ON DELETE CASCADE
);

CREATE VIEW customer_view AS SELECT cust_fname, cust_lname, cust_phone
FROM customer;

CREATE INDEX fname_index ON customer (cust_fname);
```

OUTPUT:

```
mysql> show tables;
```

```
+-----------------+
| Tables_in_dbmsl |
+-----------------+
| author          |
| book            |
| customer        |
| orders          |
| publisher       |
+-----------------+
5 rows in set (0.01 sec)

mysql> desc customer;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| cust_no      | int         | NO   | PRI | NULL    | auto_increment |
| cust_fname   | varchar(20) | YES  |     | NULL    |                |
| cust_lname   | varchar(20) | YES  |     | NULL    |                |
| cust_company | varchar(20) | YES  |     | NULL    |                |
| cust_addr    | varchar(100)| YES  |     | NULL    |                |
| city         | varchar(20) | YES  |     | NULL    |                |
| cust_phone   | varchar(20) | YES  |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql> desc orders;
+----------+--------+------+-----+---------+----------------+
| Field    | Type   | Null | Key | Default | Extra          |
+----------+--------+------+-----+---------+----------------+
| order_no | int    | NO   | PRI | NULL    | auto_increment |
| cust_no  | int    | YES  | MUL | NULL    |                |
| isbn     | bigint | YES  | UNI | NULL    |                |
| qty      | int    | YES  |     | NULL    |                |
| odate    | date   | YES  |     | NULL    |                |
+----------+--------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> desc book;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| isbn           | bigint      | YES  | MUL | NULL    |       |
| title          | varchar(20) | YES  |     | NULL    |       |
| unit_price     | int         | YES  |     | NULL    |       |
| author_no      | varchar(20) | YES  | UNI | NULL    |       |
| publisher_no   | varchar(20) | YES  | UNI | NULL    |       |
| publisher_year | varchar(4)  | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> desc author;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
```

```
| author_no   | varchar(20) | YES  | MUL | NULL    |       |
| author_name | varchar(20) | YES  |     | NULL    |       |
| country     | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> desc publisher;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| publisher_no   | varchar(20)  | YES  | MUL | NULL    |       |
| publisher_name | varchar(20)  | YES  |     | NULL    |       |
| publisher_addr | varchar(100) | YES  |     | NULL    |       |
| year           | varchar(4)   | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> desc customer_view;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| cust_fname | varchar(20) | YES  |     | NULL    |       |
| cust_lname | varchar(20) | YES  |     | NULL    |       |
| cust_phone | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> show index from customer;
+----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table    | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| customer |          0 | PRIMARY     |            1 | cust_no     | A         |           8 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| customer |          1 | fname_index |            1 | cust_fname  | A         |           7 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
+----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.74 sec)
```

CONCLUSION:

After this assignment, I was able to use and Implement the DDL commands like Create the tables, Alter the table structure and implement view, index (types of index), synonym and sequence concept.