

One of Many Books in the Need to Know Series from BrainMass

EVERYTHING YOU **NEED** TO KNOW ABOUT

MySQL[®]

**Ideal for
First Year
Computer
Science
Students**

By Rao V.N.



www.brainmass.com

Everything You Need to Know About MySQL®

By Rao V.N.

© BrainMass Inc. 2012

What Is This Book?

Everything You Need to Know publications are the best way to get a quick but detailed overview of a specific topic. Within the pages of this book, you'll find exactly what you need in order to understand the key concepts of this topic. You'll find yourself completely prepared for the next stage of your learning as it relates to this topic. Within every book, we include a collection of the most important terms and their definitions so that whenever you're in need of a refresher, you can easily refer back and remind yourself of what you're dealing with.

This book includes an introduction to basic concepts such as working with database objects and functions. The first chapter of this book includes an insight into the most commonly used database object (i.e. tables) and will help students in how to work with **tables**. The second chapter includes an insight into various MySQL[®] functions most of which are essential for web programmers. The software used in this book is MySQL[®] version 5.5.24. A free version of the software is available for download in the following link:

<http://dev.mysql.com/downloads/mysql/>

This book is ideal for students in the first year of their undergraduate studies in computer science/information systems or any student aspiring to be a web programmer and intends to work with PHP. This book would be ideal for students familiar with basic concepts of database design such as the differences between relational and non-relational databases.

Table of Contents

What Is This Book?	3
Introduction.....	5
Everything You Need to Know	6
1. Database Objects.....	7
1.1 Structured Query Language	7
1.2 Database Objects	8
1.3 Data Types in MySQL®	8
1.4 MySQL® Data Definition Language Statements.....	9
1.5 MySQL® Data Manipulation Language Statements	15
1.6 MySQL® Transaction Control Language Statements	20
1.7 MySQL® Operators	24
1.8 Common Pitfalls	25
1.9 Sample Exercises 1 with Solutions	26
1.10 Solutions to Sample Exercises 1	28
1.11 Sample Exercises 2 with Solutions	33
1.12 Solutions to Sample Exercises 2	35
2. MySQL® Functions.....	36
2.1 Use of MySQL® Functions	36
2.2 Types of MySQL® Functions.....	36
2.3 Common Pitfalls	49
2.4 Sample Exercises 1 with Solutions	51
2.5 Solutions to Sample Exercises 1	52
2.6 Sample Exercises 2 with Solutions	57
2.7 Solutions to Sample Exercises 2	58
Glossary	59
References.....	60
Additional Resources	61
About the Author	62

Introduction

MySQL[®] is the most popular open source database preferred mainly by web programmers. This book is introductory and is suitable for students intending to master web programming. Students having prior knowledge of basic database design skills would benefit the most from this book. MySQL[®] is often bundled with Apache webserver and PHP into packages such as WAMP and XAMPP. These packages are quite popular with web programmers using open source software. The popularity of PHP for web programming makes it imperative for web programmers to have an understanding of basic MySQL[®]. This book is suitable for all students who want to obtain an understanding of the basic concepts of MySQL[®].

Everything You Need to Know

This book describes basic concepts needed for using MySQL[®]. This book has two chapters dealing with database objects and functions.



Section 1

The topics covered in the first chapter include MySQL[®] database objects with key focus on creating databases, creating tables, populating and manipulating data in the tables, and querying from the tables. The first chapter ends with two sets of sample exercises with solutions.



Section 2

The second chapter presents an introduction to MySQL[®] functions including single row and group functions. There are a number of MySQL[®] functions; the focus in this book has been limited to functions frequently used in web programming. The second chapter also ends with two sets of sample exercises with solutions.

The readers of this book will be able to perform basic data definition and data manipulation activities using MySQL[®] on completion of reading this book. The readers will also be able to work with different MySQL[®] functions on completion of reading this book.

1. Database Objects

Objectives

After completing this chapter, you should be able to do the following:

- Understand the significance of SQL
- Understand different **database** objects
- State the different data types
- Describe and work with different Data Definition Language (DDL) statements
- Describe and work with different Data Manipulation Language (DML) statements
- Describe and work with different Transaction Control statements
- Understand various MySQL[®] operators

1.1 Structured Query Language

Structured Query Language (SQL) is the standard language for relational databases accepted by both ANSI (American National Standards Institute) and ISO (International Standards Organization). Every relational **database management system** needs to have an implementation of this standard for querying and manipulating the database. All the major database vendors, including Microsoft[®] and Oracle[®] have their own implementation of this standard. For instance, Oracle[®] refers to its implementation as SQL* Plus whereas Microsoft[®] refers to its implementation as Transact SQL (T-SQL[®]). MySQL[®] is the world's most popular open source relational database management system. MySQL[®] also provides a tool for the users through its administration system for supporting SQL statements.

1.2 Database Objects

There are several types of database objects including:

- Tables
- Views
- Sequences
- Synonyms

Since this book is introductory in nature, we shall deal only with tables in this book.

1.3 Data Types in MySQL®

There are several different data types used in SQL. The four most important and common data types are:

- a) Char
- b) Varchar
- c) Numeric and Integer
- d) Date

There are several other data types, but are not necessary for learning basic SQL commands.

Char data type is used for representing fixed length character data. This data type is normally used when a character **field** needs a fixed number of columns, such as gender which is represented as 'm' for male and 'f' for female. In such a case, the "char" data type can be used.

Varchar is used for representing variable length character data. For example, in the case of name of a student, the name field could contain a name such as 'John' which requires only 4 characters, or a 20-character long name. In such a case, it would result in wastage of memory space if we used a fixed length data type to reserve the memory space. In such cases, the variable length character data types offer better options.

The **Numeric** data types are used to describe both integers and floating point numbers. For example, if you wish to define a field called age as a 3-digit number, then you can indicate its data type as number (3). Alternatively the "integer" data type can be used to declare integers whereas "double" or "real" data types can be used to declare floating point numbers. Similarly if you want to define a field called salary as a floating

point number with 4-digits in integer portion and 2 digits in decimal precision, then you can indicate its data type as number (6, 2).

The **Date** data type is used to define a field that stores dates.

1.4 MySQL® Data Definition Language Statements

Data Definition Language statements set up, change and remove structures from tables. The six key data definition language statements are:

1. Create Database
2. Create Table
3. Alter Database
3. Alter Table
4. Drop Table
5. Rename Table
6. Truncate Table

Create Database

The create database command is used to create a new database. The syntax for the create database statement is:

Syntax:
Create database <database name>;

Example:
Create database University;

Figure 1 illustrates the use of create database command in MySQL®. The screenshot also demonstrates how to access the SQL statements pane.

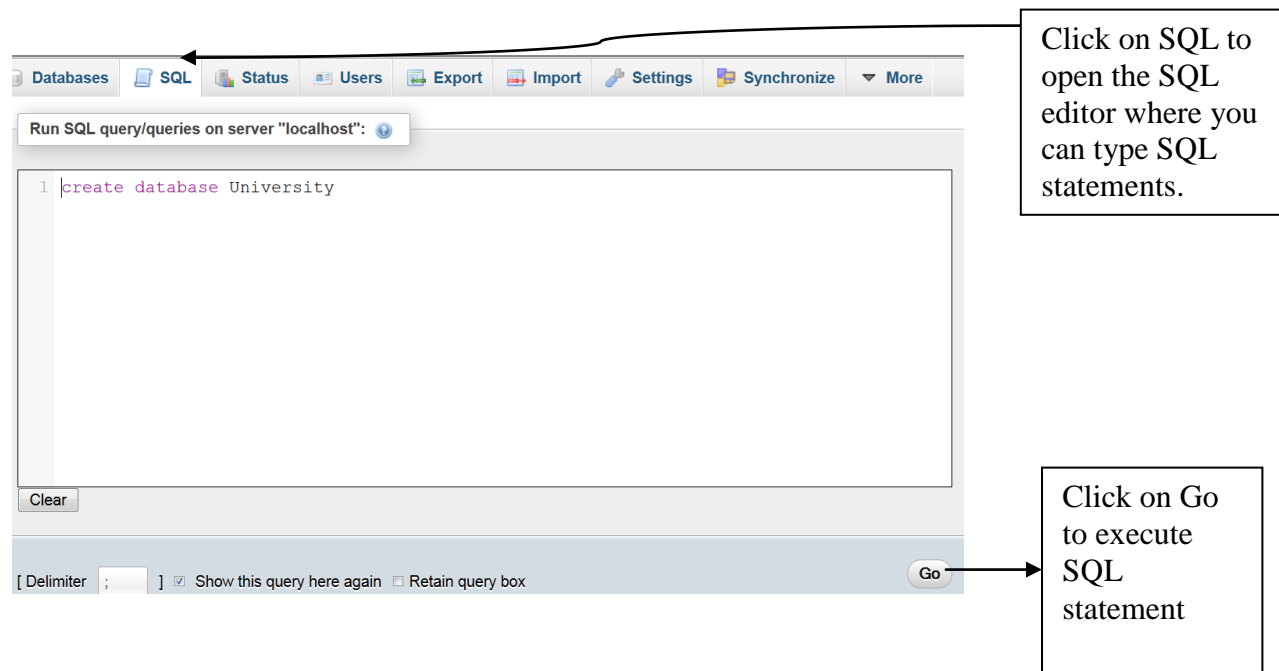


Figure 1: Create Database Statement

Use Command

The use command is used to select the database where the tables which you are going to create are located. In this case we have created the "university" database and we want to create a table "student" in this database. So we need to select the database first. The other option, instead of writing the SQL statement, is to select the database directly from the toolbar at the left end of the window.

Syntax:

Use <database name>

Example:

Use university;

Click on "Go" to execute the above statement. Once the database is selected further DDL, DML or other SQL queries and statements can be issued.

Create Table

This command is used to create a table. Each column name should have its data type and width (if applicable). For instance, if the column represents a date, then width is not required to be mentioned.

Syntax:

Create table <table name> (<column name> <datatype>[<width>]
constraints,...);

Example:

Create table student (idno number(4) , name char(20));

Describe Statement

Describe is a command that is used to display the structure of a table. The describe statement provides information about the structure of the table including the names of the columns, data types etc. A screenshot of the output of the describe statement is given below.

The screenshot shows the output of the MySQL DESCRIBE statement for a table named 'student'. It includes a table with columns: Field, Type, Null, Key, Default, and Extra. The table has two rows: 'idno' with type 'decimal(4,0)' and 'name' with type 'char(20)'. Both columns are nullable and have a default value of NULL. The interface also includes options for showing rows and headers, and a table of actions (Edit, Copy, Delete) for each row.

Field	Type	Null	Key	Default	Extra
idno	decimal(4,0)	YES		NULL	
name	char(20)	YES		NULL	

Figure 2: Describe Statement

Show Columns Statement

The show columns command is another option that can be used to display the information about the columns in a table. The output from the show columns statement is shown in Figure 3, and is quite similar to the describe statement, the describe statement gives more options to the user for

manipulating the structure of the database, which is discussed in the next section.

Syntax:

Show [full] columns [from|in <table-name> [from|in <database name>]

Example:

Show columns from student;

Or

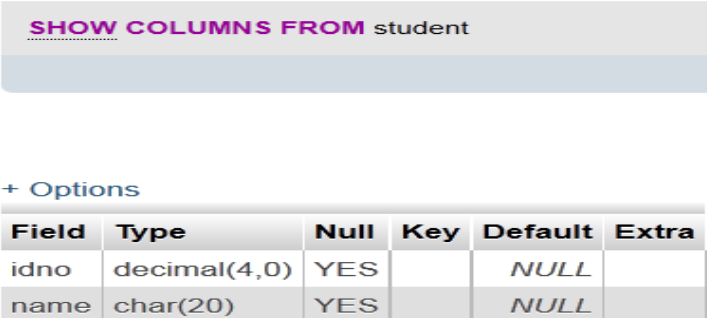
Show columns in student;

Or

Show columns from student from university;

Or

Show columns in student in university;



The screenshot shows a MySQL command prompt with the command `SHOW COLUMNS FROM student` entered. Below the command, a table displays the column details for the 'student' table. The table has six columns: Field, Type, Null, Key, Default, and Extra. The first column is 'idno' with type 'decimal(4,0)', and the second is 'name' with type 'char(20)'. Both columns are nullable and have no keys or defaults.

Field	Type	Null	Key	Default	Extra
idno	decimal(4,0)	YES		NULL	
name	char(20)	YES		NULL	

Figure 3: Show Columns Command

Note: The “in” and “from” clauses in the show columns statement can be used interchangeably. The database name is optional and is not required if the database is already selected using the “use” statement.

Alter Database

The alter database command permits changing several characteristics of the database such as the national language characteristics and upgrading

database versions. However, as these chapters on SQL are introductory, at this stage, we shall not explore further into this topic. Furthermore, this statement is used by the database administrator for making changes on the overall characteristics of the database at selective intervals and several administrator privileges are required for executing this statement. The generic syntax for this statement is:

Syntax:

```
alter database <database name> <alter specification(s)>
```

Alter table

This statement is used to alter the structure of an existing table. There are two options:

- a) Add
- b) Modify

The add option is used along with alter table statement to add a new column to the table. The modify option is used to modify an existing column of an already existing table.

Using modify clause we can:

- a) Increase or decrease the width of an existing column.
- b) Change the data type of an existing column.

However, there are some restrictions while changing the data type and decreasing the width of a column. However there are no restrictions while increasing the width of a data type. The restrictions will be dealt in detail at a later stage.

Syntax:

```
Alter table <table-name><add/modify> <column-name><data  
type>(<width>),....;
```

Example:

```
alter table student add gender char(2);
```

In the above example we are adding a new column called as gender to an existing table. You can observe the change in structure as shown in Figure 4.

Field	Type	Null	Key	Default	Extra
idno	decimal(4,0)	YES		NULL	
name	char(20)	YES		NULL	
gender	char(2)	YES		NULL	

Figure 4: Table structure after alter command

Example:

If we want to modify the width of "name" field in the above table from 20 characters to 25 characters, then we have issue the following command:

```
alter table student modify name varchar(25);
```

This would modify the width from 20 characters to 25 characters.

Rename Table

The rename statement is used to rename a table. Once the rename statement is issued, the table can be accessed only using the new name.

Syntax:

```
Rename <old-table-name> to <new-table-name>;
```

Example:

```
rename table student to stud;
```

Note: After changing the name of the table, if you still see the name in the toolbar at the left of the window, go to the top of the window and refresh the view to see the new updated name of the table.

Truncate Table

Truncate table is used to delete all the records of a table, but the structure of the table remains intact.

Syntax:

Truncate table <table-name>

Example:

```
truncate table student;
```

Drop Table

Drop Table statement is used to the columns of a table. This command deletes the table physically from the disk and must be used with caution. Once a table is dropped, it cannot be retrieved back.

Syntax:

Drop table <table name>

Example:

```
drop table stud;
```

1.5 MySQL® Data Manipulation Language Statements

Data Manipulation Language statements retrieve data from the database, enter new rows, change existing rows, and remove unwanted rows from tables in the database. The four data manipulation language statements are:

- a) Insert
- b) Update
- c) Delete
- d) Select

These commands affect a single or a group of records.

Insert

The Insert statement is used to insert a single **record** into the database. It can also be used to insert multiple records at a time.

Syntax:

Insert into <table-name> values (value1, value2,...);

Note: While inserting character and date values, they must be enclosed in single quotes. The numeric values do not have to be inserted in single quotes.

Example:

As we had dropped the student table in the previous exercise, you need to create it again before continuing with the remaining examples. So first create the student table with command given below:

```
create table student (idno numeric(4), name varchar(25), gender char(2));
```

After the table is created, we can continue with inserting new records.

```
insert into student(idno, name, gender) values (1,"John","m");
```

In the above statement we had given the column names as well. The column names are optional and not needed if you are inserting values for all the columns. If you are inserting a selective number of values (for example: only the idno and name), the column names are mandatory to avoid ambiguity on what the values correspond to. Including the field names in the insert statements is strongly recommended. However, the statement could also be written as:

```
insert into student values (1,"John","m");
```

Note: The numeric values are normally given without quotations, whereas character, string, and date values must be enclosed in double quotations ("**<value>**").

Inserting Multiple Values

In case we have to insert 10 records, inserting them with 10 insert commands would be cumbersome. Instead, we can issue the following command, for instance to insert five records:

Example:

```
insert into student (idno, name, gender) values  
    (2,"Jack", "m"),  
    (3,"Mary", "f"),  
    (4,"Steven","m");
```

Note: Each record has to be separated by comma when more than one record is inserted with a single insert statement.

Update

The update statement is used to update the values of a particular record or a group of records.

Syntax:

```
Update <table-name> set < column-name>=<value> where <condition>;
```

Example:

```
update student set name="Johnson" where idno=1;
```

Note: If the update command is given without “where” clause, then all the records in the table are updated. You can also modify more than column in a record or a set of records at the same time. For instance, if you want to change the name and gender for the student then you can give the following command:

```
update student set name="Jenny", gender="f" where idno=1;
```

Note: The column key-value pairs would have to be separated by a comma if more than one column is modified using the update statement.

Delete

The delete statement is used to delete a single record or a group of records from the table.

Syntax:

Delete from <table-name> where<condition>

Example:

delete from student where idno=1;

Note: If the delete command is given without “where” clause, then all the records in the table are updated.

Select

Select statement can be used for selection (selecting set of rows), projection (selecting set of columns) as well as joins (data from multiple tables).

In its basic form, a SELECT statement must include the following:

- a) A SELECT clause, which specifies the columns to be displayed
- b) A FROM clause, which identifies the table containing the columns that are listed in the SELECT clause

Syntax:

SELECT *|{[DISTINCT] column|expression [alias],...} FROM table;

In the syntax:

SELECT	is a list of one or more columns
*	selects all columns
DISTINCT	suppresses duplicates
column expression	selects the named column or the expression
alias	gives the selected columns different headings
FROM table	specifies the table containing the columns

Table 1: Select Syntax

Basic Select statement can be used in different situations:

- a) Selecting all columns
- b) Selecting specific columns
- c) Selecting records meeting specific criteria

We can display all columns of data in a table by using the select statement with an asterisk (*) sign.

The first example statement given below would display the idno and name fields for all the records in the student table. The second example would display all the fields for all the records in the student table. In the second example, the asterisk symbol ("*") indicates that all columns in the table would be displayed. So if we want the information from all the columns for a particular record then we use the asterisk symbol.

Examples:

```
select idno, name from student;
```

```
select * from student;
```

Removing duplicates using select statement

Unless you indicate otherwise, SQL displays the results of a query without eliminating the duplicate rows. To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause.

Example:

```
select distinct gender from student;
```

Assuming that gender has only two values "m" for male and "f" for female, the above statement would only display these two values without any repetitions or duplicates. Hence if we give the above query without the distinct option it would present duplicate values for gender column. The above query would remove all the duplicates.

Defining a Column Alias

When displaying the result of a query, MySQL[®] normally uses the name of the selected column as the column heading. This heading may not be descriptive and therefore may be difficult to understand. You can change a column heading by using a column alias.

Specify the alias after the column in the SELECT list using blank space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces or special characters (such as # or \$), or if it is case-sensitive, enclose the alias in double quotation marks (" ").

Example:

```
select name as "Student Name", gender from student
```

The above query would display the student names under the caption "Student Name" while displaying the gender under the same caption as the column name. The output of this query is shown in Figure 5.

Student Name	gender
Jack	m
Mary	f
Steven	m
Alan	m

Figure-5: Example of Column Alias

1.6 MySQL[®] Transaction Control Language Statements

Transaction control statements manage the changes made by DML statements. Changes to the data can be grouped together into logical transactions.

- a) Commit
- b) Rollback
- c) Savepoint

Transactions

Before explaining the transaction control statements, it is advisable to understand transactions. Transactions give more flexibility while at the same time ensure consistency in the event of system failure. A transaction begins when the first DML statement is encountered and ends in the following scenarios:

- a) Transaction Control statements such as commit or rollback
- b) DDL statement
- c) Data Control Statement
- d) System failure or exit from the local environment

Commit

The commit statement ends the current transaction by making all the pending changes permanent. In the case of unexpected system failure, the changes could be lost if either an implicit or an explicit commit statement is not issued.

Syntax:

Commit

Example:

Commit;

Rollback

The rollback statement ends the current transaction by removing all the pending data changes. For example if we have issued a DML statement by mistake and wish to correct the mistake (akin to the undo feature in word processing applications), we can always rollback the changes either to a particular savepoint or previously committed state.

Syntax:

Rollback [<to savepoint savepoint-name>]

Examples:

```
rollback;
```

or

```
rollback to savepoint abc;
```

In the above statement, abc is the name of the savepoint. Savepoints are explained next.

Savepoint

The savepoint statement is used to set a marker within the current transaction. This allows the user to go back to a particular stage within the transaction i.e. to roll back to a particular point of the transaction

Syntax:

```
Savepoint <savepoint-name>
```

Example:

```
savepoint abc;
```

Example:

This example will explain the function of commit and rollback. Figure-6 is a snapshot of the records in the table at this point.

idno	name	gender
2	Jack	m
3	Mary	f
4	Steven	m

Figure-6: Records in the Student table

Before we start working on this exercise, we need to set autocommit to false. By default, MySQL[®] settings have autocommit turned on i.e., the records will automatically be committed. To test rollback and savepoint we

would need to set automatic commit off. This can be done through the following statement:

```
SET autocommit=0;
```

Now we can proceed and test the functioning of rollback command.

First, we will insert a record with the following values:

```
Idno=5, name="Alan", gender="m"
```

```
Insert into student (idno, name, gender) values (5,"Alan","m")
```

Now, we can check if the record was inserted through the following select statement.

```
Select * from student
```

Figure-7 shows the snapshot of the records in the table after the record was inserted.

idno	name	gender
2	Jack	m
3	Mary	f
4	Steven	m
5	Alan	m

Figure-7: Records in the Student table

At this point if we give the rollback command the student record that was just inserted will be rolled back i.e., the record will be deleted from the table. We can test this by giving the rollback command:

```
Rollback
```

And then checking the records in the table through the select statement:

```
Select * from student
```

Now you would notice that the record that was inserted has just been rolled back.

If you want to save the changes made to the table you would have to type commit.

As discussed before, it is important to note that autocommit is enabled by default.

1.7 MySQL® Operators

Operators

While you might be familiar with the basic arithmetic operators (addition+, multiplication *, subtraction -, and division/), there are some other operators, you would need to be familiar with to work in SQL.

Comparison Operators

Given below in Table 2 is a list of comparison operators, with the respective meaning. You will understand them better as we use these operators in the lab exercises later in this book.

=	Equal To
<>	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
BETWEEN...AND	Between two values
LIKE	Match a character pattern
IN	Match any of a list of values
IS NULL	Is a null value

Table 2: Comparison Operators

1.8 Common Pitfalls

The most common errors while executing SQL queries include:

- a) Not placing character and date fields in double quotes

All character and date field values should be placed in double quotes. Numeric fields on the other hand do not have to be placed in double quotes even though MySQL[®] will accept numeric values in double quotations as well.

- b) Incorrect usage of Select Statement

Remember that the asterisk sign (*) indicates all columns, and hence cannot be used in conjunction with other field names.

Example:

```
select ename,* from emp;
```

The above query would result in a syntax error as we cannot use both * and column names at the same time. In case you need only a few fields from the table, do not use the asterisk sign and instead list all the fields in the select query.

1.9 Sample Exercises 1 with Solutions

1. Write SQL statement to create a database named Company.
2. Write SQL statement to start using the database company created in the previous question.
3. Create a table called as Employees having the following structure:

Field Name	Data Type	Width
EmpID	Number	4
FName	Character	15
LName	Character	15

4. Insert at least 5 employee records in the table. Use both the methods for inserting records.
5. Write a query to display the structure of the Employees table.
6. Write a query to display all the employee records in the table.
7. Write a query to display the Employee ids and the first names of all employees in the table.
8. Write a query to display the first name and the last name of all the employees.
9. Write a SQL statement to delete a particular single Employee record from the table. In this case delete the details of employee with employee id 122.
10. Write a SQL statement to increase the width of the field "FName" from 15 to 20 characters.
11. Write a SQL statement to add a new field called as gender, of data type char and width 1.
12. Write a SQL statement to update all the records already in the table with the gender.

13. Write a query to display the last name and the gender of all the female employees in the company.
14. Write a query to add a new column called as age, with data type as number and width 2.
15. Write a query to update all the records with their respective age.
16. Write a query to display the first names of all the male employees whose age is greater than 20.
17. Write a query to display the last names of all female employees who are in the age group 20-23 years.
18. Write a query to sort all the employee records on the basis of their age.
19. Write a query to display all the employee records on alphabetical order of their first names.
20. Write a query to display all the female employee records on alphabetical order of their first names and then their last names.

1.10 Solutions to Sample Exercises 1

1. create database company
2. use company
3. create table Employees (EmpID numeric(4), FName varchar(15), LName varchar(15))

Explanation:

In the above create table statement, EmpID can be any valid 4-digit number, while first and last name fields can have up to 15 characters. Note that the "varchar" data type has been used for character fields. The "char" data type could have been used as well. In order to understand the reason for using "varchar2" data type, study the section of chapter explaining the advantages of "varchar2" as compared to "char".

4. **Method 1:** Inserting records one by one
insert into Employees values(121,'John','Walters');

Method2: Inserting multiple records using single insert statement separating the records by a comma as shown below:

```
insert into Employees values(122,'Harry','Winters'),  
(123,'Jackie','McKenzie'),  
(124,'Asin','Malik'),  
(125,'Alison','McNamara');
```

Explanation:

The records can either be inserted one-by-one, or through a batch by separating each record with a comma.

5. desc Employees;
Or
describe Employees;
6. select * from Employees;

7. `select EmpID, FName from employees;`

Explanation:

If you want to display all the fields of a table, use the asterisk (*) character. If you want to display only some of the fields of the table, then use the respective field names.

8. `select FName,Lname from Employees;`
9. `delete from Employees where EmpID=122;`

Explanation:

To check if the record has been deleted from the table, you can display all the records of the table:

```
select * from Employees;
```

10. `alter table Employees modify FName varchar(20);`

Explanation:

The modify option of alter table command allows changing the width of a data type. However, do note that since we are increasing the width there are no problems. However, if we decrease the width of a data type, we should ensure that there are no records that would be affected by this change, otherwise there is a possibility of data loss.

In this case you can check the change in the width by checking the structure of the database:

```
desc Employees;
```

11. `alter table Employees add (gender char(1));`

Explanation:

The alter table statement has two options: add and modify. In this case we are adding a new field called as gender, hence we are using add clause.

You can issue the describe statement to see the changes in the table.
desc Employees;

12. update Employees set gender='m' where EmpID=121;
update Employees set gender='f' where EmpID=123;
update Employees set gender='f' where EmpID=124;
update Employees set gender='m' where EmpID=125;

Explanation

There are two records in the table. Hence, we have to give two update statements to update both the records.

The output can be verified by issuing the select statement:
select * from Employees;

13. select lname, gender from Employees where gender='f';

Explanation:

This query is similar to the previous query, except for change in the fields being displayed as well as the change in the where condition.

14. alter table Employees add age numeric(2);

Explanation:

The add option of the alter table statement is used to change the structure of a table. In this case age is taken as a 2-digit number.

15. update Employees set age=42 where EmpID=121;
update Employees set age=23 where EmpID=123;

Assuming that the remaining two students are of the same age then you can give a single update statement using the IN operator:

update Employees set age=27 where EmpID in (124,125);

Explanation

The above two statements will update the two records with respective ages. You can check the output by displaying all the records with the select statement:

```
select * from Employees;
```

16. `select fname from Employees where gender='m' and age>20;`

Explanation:

In this question, we have to take two criteria into consideration. First, the gender must be male and second, the age must be greater than 20.

17. `select lname from Employees where gender='f' and age between 20 and 23;`

Explanation:

In this question, the “between” statement is used in the where clause of the select statement to display the last name of all female employees in the age group. Note that between includes both the lower and upper limit.

18. `select * from Employees order by age asc;`

Explanation:

The above SQL statement sorts all the employee records in the ascending order of age. Note that in this case all the records are sorted in the ascending order of age. The following query would also result in the same result:

```
select * from Employees order by age;
```

19. `select * from Employees order by fname;`

Explanation:

The above SQL statement sorts all the records in the ascending order of first name. Because no option was mentioned in the above select statement, all queries are sorted in ascending order of first names because this is the default option. The following select query also accomplishes the same:

```
select * from Employees order by fname asc;
```

20. `select * from Employees where gender='f' order by fname asc, lname desc;`

Explanation:

In order to understand this question better, insert the following record into the employees table:

```
insert into Employees values(129, 'Jackie','Abraham','f',30);
```

Now, you might notice that there are two female employees with the same first name but different last names. This query addresses such records, and in such cases it first sorts the first names in ascending order and the last names in descending order.

```
select * from Employees where gender='f' order by fname asc, lname desc;
```


1.11 Sample Exercises 2 with Solutions

1. Write a SQL command to create a table called as Product with the following fields:

Field Name	Data Type	Width
ProductID	Number	4
ProductName	Character	20
ProductPrice	Number	8,2

2. Write a SQL command to insert the following records in the table:

ProductID	ProductName	ProductPrice
1234	Pens	1.50
3456	Pencils	2.80
1298	Markers	5.50
4323	Erasers	4.20
2857	Staplers	4.90
2857	Highlighters	4.60

3. As you can see in question 2, the record with ProductID as 2857 is inserted twice. Delete the record with the second occurrence of the ProductID.
4. Write a SQL command to increase the width of ProductPrice from 8,2 to 10,2.
5. Write a query to display the names of all products in ascending order of their price.
6. Write a SQL command to update the price of Staplers from 4.90 to 4.40.

7. Write a query to display the names of all products whose price is in the range 2.50 and 4.
8. Write a SQL command to increase product price of all products by 10%.
9. Write a SQL command to increase by 20% the price of all the products the price of which is less than 2000.
10. Write a SQL command to insert a new column called as "expirydate", of data type "date".
11. Write a SQL command to update all the records by updating the expiry date.
12. Write a query to display the names of all products in ascending order of their expiry date.

1.12 Solutions to Sample Exercises 2

1. create table Product (ProductID numeric(4), ProductName varchar(20), ProductPrice numeric(8,2))
2. insert into Product values (1234, 'Pens', 1.50),
(3456, 'Pencils', 2.80),
(1298, 'Markers', 5.50),
(4323, 'Erasers', 4.20),
(2857, 'Staplers', 4.90),
(2857, 'Highlighters', 4.60)
3. select rowid from Employee;
delete from Product where ProductName='Highlighters'
4. alter table Product modify ProductPrice numeric(10,2)
5. select * from Product order by ProductPrice
6. update Product set ProductPrice=4.40 where ProductName='Staplers'
7. select ProductName from Product where ProductPrice between 2.50 and 4
8. update Product set ProductPrice=ProductPrice*1.1
9. update Product set ProductPrice=ProductPrice*1.2 where ProductPrice<2000
10. alter table Product add expirydate date
11. update Product set expirydate='2012-11-01' where ProductID=1234;
update Product set expirydate='2013-10-11' where ProductID=3456;
update Product set expirydate='2012-08-11' where ProductID=1298;
update Product set expirydate='2014-07-21' where ProductID=4323;
update Product set expirydate='2011-02-25' where ProductID=2857;
12. select ProductName from Product order by expirydate asc

2. MySQL[®] Functions

Objectives

After completing this chapter, you should be able to do the following:

- Understand the use of MySQL[®] Functions.
- Understand different types of MySQL[®] Functions.
- Understand the difference between single row and group functions.
- Use different numeric, character, date and general functions.
- Use Group by with having clause.
- Avoid common pitfalls while using group by clause.

2.1 Use of MySQL[®] Functions

Functions are a very powerful feature of MySQL[®]. They can be used to do the following:

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

2.2 Types of MySQL[®] Functions

There are two types of SQL functions:

- a) Single-row functions
- b) Group Functions

Single Row Functions

Single row functions operate only on single rows and return one result per row. The key single-row functions are:

- a) Character
- b) Number
- c) Date
- d) Conversion

Character Functions

Single-row functions accept character data as input and can return both character and numeric values. There are two key categories of character functions:

- a) Case-conversion functions
- b) Character-manipulation functions

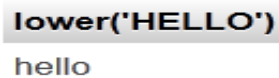
Case Conversion functions

The two important case-conversion functions are:

- i) **Lower:** Converts mixed-case or uppercase character strings to lowercase.

Example:

```
select lower('HELLO')
```



A screenshot of a MySQL query result. The first row shows the query 'lower('HELLO')' in a grey header box, and the second row shows the result 'hello'.

lower('HELLO')
hello

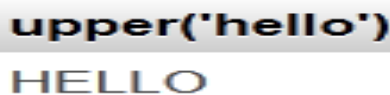
Figure 8: Lower Function

The Lcase() function is a synonym for the Lower() function.

- ii) **Upper:** Converts mixed-case or lowercase character strings to uppercase.

Example

```
select upper('hello')
```



A screenshot of a MySQL query result. The first row shows the query 'upper('hello')' in a grey header box, and the second row shows the result 'HELLO'.

upper('hello')
HELLO

Figure 9: Upper Function

The Ucase() function is a synonym for the Upper() function.

Character Manipulation Functions

The seven important character manipulation functions are:

i) Concat: The concat function is used to join two parameters. This function displays the information from the two fields provided as parameters in one output column.

Example:

```
select concat(fname,lname) from employees;
```



concat(fname,lname)
JohnWalters
JackieMcKenzie
AsinMalik
AlisonMcNamara
JackieAbraham

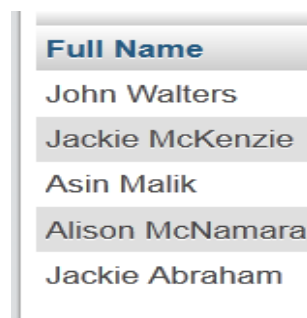
Figure 10: Concat Function

As you might have noticed that the full names of the employees are displayed, however there is no space between the first and last names. If you want to give space, then issue the following command:

```
select concat(fname,concat (' ',lname)) as "Full Name" from employees;
```

Note: The query with concat() function should work if you have created the employees table in the solved examples section of Chapter 1.

As you can see in Figure 11, the output has the full name displayed properly with an appropriate caption.



Full Name
John Walters
Jackie McKenzie
Asin Malik
Alison McNamara
Jackie Abraham

Figure 11: Nested Concat Function

ii) Substring: This function extracts a string of determined length. For example, if we want to display the first four characters from a string "Hello" as shown in Figure 12:

```
select substring("Hello",1,4)
```



Figure 12: Substr function

Note that the syntax of substring() function is: substr(<string>,<starting position>,<number of characters to be extracted>).

The substr() function is a synonym for the substring function.

The substr() function can be used to find any portion of the string. For example, if we want to display the last three characters of a string, the starting position can be represented through a negative number so that the starting position is calculated from the end of the string rather than the beginning of the string as shown in Figure 13.

```
select substring("Hello",-3)
```



Figure 13: Substring() function example

iii) Length: This function finds the length of the string. As shown in Figure 16, the query displays the first names of the employees and the length of their first names.

```
select fname, length(fname) from employees;
```

fname	length(fname)
John	4
Jackie	6
Asin	4
Alison	6
Jackie	6

Figure 14: Example of Length function.

iv) Instr: This function finds the numeric position of a named character.

Example

```
select fname, instr(fname,'E') from employees;
```

As you might have noticed from the output in Figure 15, the above query displays the name of the employee as well as the position of the character 'E' in the name.

fname	instr(fname,'E')
John	0
Jackie	6
Asin	0
Alison	0
Jackie	6

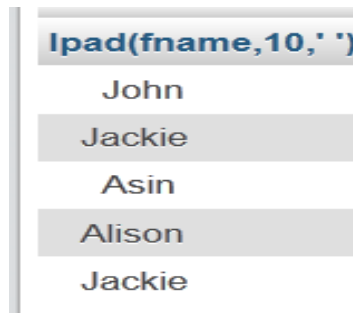
Figure 15: Example of Instr() function.

v) Lpad: To pad a string value with space or character towards left end of a string value.

Example:

```
Select lpad(fname,10,' ') from employees;
```

The above function aligns all the names of the employees to the right and pads up the spaces on the left with blank characters as shown in Figure 16.



The screenshot shows a MySQL query window with the following text:

```
lpad(fname,10,' ')
```

John
Jackie
Asin
Alison
Jackie

Figure 16: LPad() function example

Example:

```
Select lpad(fname,10,'*') from employees;
```

This query is similar to the previous query except that the spaces in the left are padded with the asterisk character as shown in Figure 17.



The screenshot shows a MySQL query window with the following text:

```
lpad(fname,10,'*')
```

*****John
*****Jackie
*****Asin
*****Alison
*****Jackie

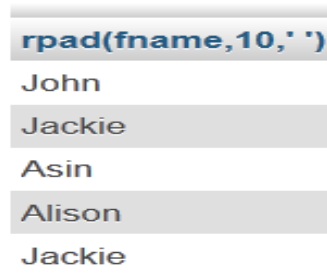
Figure 17: LPad() function example

vi) RPad: To pad string value or character towards right end of a string value.

Example:

```
Select rpad(fname,10,' ') from employees;
```

The above function aligns all the names of the employees to the left and pads up the spaces on the right with blank characters as shown in Figure 18.



<code>rpad(fname,10,' ')</code>
John
Jackie
Asin
Alison
Jackie

Figure 18: RPad() function example

Example:

Select `rpad(fname,10,'*')` from employees;

This query is similar to the previous query except that the spaces in the right are padded with the asterisk character as shown in Figure 19.



<code>rpad(fname,10,'*')</code>
John*****
Jackie*****
Asin*****
Alison*****
Jackie*****

Figure 19: RPad() function example

vii) Trim: Trims leading or trailing characters (or both) from a character string

Number Functions

Number functions accept numeric input and return numeric values. There are several number functions, the most important of which are:

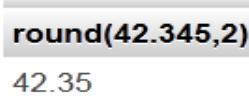
a) Round (column/expression, n)

This function rounds the column or expression to n decimal places. If the value of n is not given, then there are no decimal places. Similarly if the value of n is negative, then the numbers to the left of the decimal point are rounded.

Example:

```
Select round(42.345,2)
```

The above statement would return 42.35 because we are trying to round the number to two decimal places, and the value of the third decimal place is equal to 5. In such a case the second decimal is rounded to the next integer as shown in Figure 20. If the third decimal place was less than 5 then the digit at the second decimal place would not change.



```
round(42.345,2)  
42.35
```

Figure 20: Example of Round() Function

b) Mod(a,b)

This function returns the remainder of m divided by n.

Example:

```
select mod(12,5)
```

The above select statement would return 2, because when we divide 12 by 5, the remainder is 2.



```
mod(12,5)  
2
```

Figure 21: Example of Modulus Function

Date Functions

The default display and input format for any date is YYYY-MM-DD where YYYY is the year in a 4-digit format, MM is the month in a 2-digit format, and DD is the day in a 2-digit format.

a) Curdate() Function

Curdate() is a date function that returns the current database server date as shown in Figure 22.

Example:

```
select curdate()
```

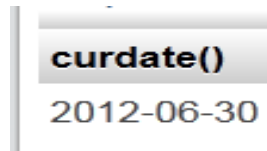


Figure 22: Curdate() function

b) Curtime() Function

The Curtime() function returns the current database server time as shown in Figure 23.

Example:

```
Select curtime()
```

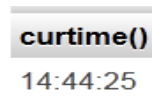


Figure 23: Curtime() function

c) Extract() function

The extract function is used to extract parts from date.

Example: The query below extracts the year from the date as shown in Figure 24.

```
SELECT EXTRACT(YEAR FROM '2012-06-22');
```

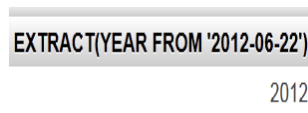


Figure 24: Example of Extract() function

Similarly, to extract the month from the current date as shown in Figure 25, we can give the following statement:

```
select extract(month from curdate())
```

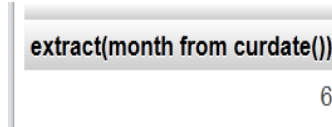


Figure 25: Example of Extract() function

Group Functions

Group functions operate on a group of rows and give one result per group.

The seven important group functions are:

- a) Avg (Average)
- b) Count
- c) Max (Maximum)
- d) Min (Minimum)
- e) Stddev (Standard Deviation)
- f) Sum
- g) Variance

Examples of these group functions are covered in the sample exercises section at the end of the chapter. These examples also have explanation about how these group functions work.

Group by Clause

This clause groups rows on a particular column.

Before starting working on group functions, create the "Emp" table using the statement below:

```
CREATE TABLE EMP
(EMPNO NUMERIC(4) NOT NULL, ENAME VARCHAR(10),
JOB VARCHAR(9), MGR NUMERIC(4), HIREDATE DATETIME,
SAL NUMERIC(7, 2), COMM NUMERIC(7, 2),
DEPTNO NUMERIC(2))
```

Also insert the following 14 records so that we can test the group functions:

```
INSERT INTO EMP VALUES
(7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, NULL, 20);
INSERT INTO EMP VALUES
(7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, NULL, 30);
INSERT INTO EMP VALUES
(7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
INSERT INTO EMP VALUES
(7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
INSERT INTO EMP VALUES
(7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20);
INSERT INTO EMP VALUES
```

```

(7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
INSERT INTO EMP VALUES
(7698, 'BLAKE', 'MANAGER', 7839, '1981-05-1', 2850, NULL, 30);
INSERT INTO EMP VALUES
(7782, 'CLARK', 'MANAGER', 7839, '1981-06-19', 2450, NULL, 10);
INSERT INTO EMP VALUES
(7788, 'SCOTT', 'ANALYST', 7566, '1982-12-09', 3000, NULL, 20);
INSERT INTO EMP VALUES
(7839, 'KING', 'PRESIDENT', NULL, '1981-09-17', 5000, NULL, 10);
INSERT INTO EMP VALUES
(7844, 'TURNER', 'SALESMAN', 7698, '1981-09-18', 1500, 0, 30);
INSERT INTO EMP VALUES
(7876, 'ADAMS', 'CLERK', 7788, '1983-01-12', 1100, NULL, 20);
INSERT INTO EMP VALUES
(7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, NULL, 20);
INSERT INTO EMP VALUES
(7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);

```

After inserting the 14 records check the table, you should see the records as shown in Figure 26.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980-12-17 00:00:00	800.00	NULL	20
7900	JAMES	CLERK	7698	1981-12-03 00:00:00	950.00	NULL	30
7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981-02-22 00:00:00	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981-04-02 00:00:00	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-06-19 00:00:00	2450.00	NULL	10
7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00	3000.00	NULL	20
7839	KING	PRESIDENT	NULL	1981-09-17 00:00:00	5000.00	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-18 00:00:00	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1983-01-12 00:00:00	1100.00	NULL	20
7902	FORD	ANALYST	7566	1981-12-03 00:00:00	3000.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-23 00:00:00	1300.00	NULL	10

Figure 26: Records of the Emp Table

Now, we can test the group functions.

Example:

First we will write a query to display the department number and the maximum salary in each department. For this we can use the group max() to find the maximum salary. But this would only display the maximum salary for all departments, if we want to group according to department number as shown in Figure 27 then we have to use the group by clause.

```
select deptno,max(sal) from emp group by deptno;
```

deptno	max(sal)
10	5000.00
20	3000.00
30	2850.00

Figure 27: Example of Max() function

The next example query displays the total salary for each department grouped by the job position as shown in Figure 28. In this case we can use the sum() function.

```
select deptno,job,sum(sal) from emp group by deptno,job;
```

deptno	job	sum(sal)
10	CLERK	1300.00
10	MANAGER	2450.00
10	PRESIDENT	5000.00
20	ANALYST	6000.00
20	CLERK	1900.00
20	MANAGER	2975.00
30	CLERK	950.00
30	MANAGER	2850.00
30	SALESMAN	5600.00

Figure 28: Example of sum() function

Having Clause

This clause is used to specify certain conditions in rows, retrieved by using group by clause. This clause should be preceded by using group by clause.

In the query below the department number is displayed along with the total salary for each department for those employees working in department numbers 10 and 20. The sample output is shown in Figure 29.

Example:

```
select deptno,sum(sal) from emp group by deptno having deptno in (10,20);
```

deptno	sum(sal)
10	8750.00
20	10875.00

Figure 29: Example of Sum() function

When you have the "WHERE", "GROUP BY", and "HAVING" clauses together in a "SELECT" statement, MySQL[®] processes the "WHERE" clause first. The rows that are returned after the WHERE clause executes are then grouped based on the GROUP BY clause. Finally, any conditions on the group functions are applied to the grouped rows before the final output is returned.

2.3 Common Pitfalls

a) Use of like operator and wild cards

The common mistake made by students while working with wild card characters (%) and (_) is the use of "=" operator instead of "like" operator for comparisons.

Example:

```
Select ename from emp where ename='A%';
```

The result of the above query would not be an error, as the syntax is proper, the error is more logical than syntactical. No records would be displayed for the above query even though several employees have 'A' as the first character of their name. Instead issue the following query:

```
Select ename from emp where ename like 'A%';
```

The above query would display the first names of all employees whose first name starts with 'A' as shown in Figure 30.



ename
ALLEN
ADAMS

Figure 30: Example of Like Operator

b) "IS" operator and null values

Another common mistake made is while working with null values. Null values do not have any defined value; hence the use of equality or comparison operators is not valid. Hence while using null values we have to use either "is" or "is not" operators. For example, if you want to display all job_id from employees table where commission percentage is not given, the following query would NOT display the desired result:

```
select distinct job from emp where comm = null;
```

Instead use the following query:

```
select distinct job from emp where comm is null;
```

So, only use the "is" operator (with "not" operator if required), while checking for null values.

2.4 Sample Exercises 1 with Solutions

Note: Solve all the following questions by using the Emp table created in the examples in this chapter.

1. Write a query to display the highest and least salary of each department in the company.
2. Write a query to count the number of employees with job position "CLERK" and "MANAGER".
3. Write a query to display the employee names along with the date of hiring for all employees who were hired in 1982.
4. Write a query to display the name of all employees who do not have a manager.
5. Write a query to display the names of all employees who have both an "a" and an "e" in their last name.
6. Write a query to display the employee number, name, salary and new salary (with salary increased by 15%) for each employee. Name the new columns as New Salary.
7. Write a query that displays the name in lowercase and the length of the name for all employees whose name starts with the letters "A", "B" or "C". Give each column an appropriate label. Sort the results by the employees' names.
8. Write a query to display the names of all employees who are working as clerks (with job position CLERK). Also display the length of service i.e. the number of months the employees have worked in the company.
9. Write a query to display the first name and salary for all employees. Format the salary to be 12 characters long, left-padded with the \$ symbol. Label the column as SALARY.
10. Write a query to display the total number of employees in each job

2.5 Solutions to Sample Exercises 1

1. select deptno, max(sal) as "Maximum Salary", min(sal) as "Minimum Salary" from emp group by deptno;

Output:

deptno	Maximum Salary	Minimum Salary
10	5000.00	1300.00
20	3000.00	800.00
30	2850.00	950.00

Figure 31: Output for query 1

Explanation:

This query makes use of the group by clause, apart from the group functions max() and min(), to find the maximum and minimum salaries. The use of the group by clause allows the display of maximum and minimum salary for each employee.

2. select JOB, count(*) as "Total Number of Employees" from emp group by job having job in ('CLERK','MANAGER');

Output:

JOB	Total Number of Employees
CLERK	4
MANAGER	3

Figure 32: Output for query 2

Explanation:

This query makes use of the "having" clause along with the "in" operator. The "having" clause is always used in conjunction with the group by clause and could be used to compare the column in the group by clause with a certain set of values. The count (*) function would count the number of employees; in this case we are interested only in two departments, so this function would count the total number of employees in each of these departments.

3. `select ename, hiredate from emp where hiredate like '1982%';`

Output:

ename	hiredate
SCOTT	1982-12-09 00:00:00
MILLER	1982-01-23 00:00:00

Figure 33: Output for query 3

Explanation:

This query uses the like operator. Note that we have to use the like character whenever we make use of the wild characters for matching patterns.

4. `select ename from emp where mgr is null;`

Explanation:

The manager_id field contains the employee_id of the manager of an employee. Hence, if this field is empty, it means that the employee has no manager. Note: Do not use the "=" operator instead of "is" operator. In this case only one employee does not have a manager.

5. `select ename FROM emp WHERE ename LIKE '%a%' AND ename LIKE '%e%';`

Output:

ename
JAMES
ALLEN
BLAKE

Figure 34: Output for query 5

Explanation:

This query makes use of the wildcard character '%'. This wild card basically means "any number of characters". Hence, '%a%' means that there can be any number of characters before and after the character 'a'. The testing of two conditions means that the characters 'a' and 'e' should be there at some position within the name. The use

of 'and' condition ensures that both 'a' and 'e' are part of the last name of the employee.

6. `select empno, ename, sal, ROUND(sal * 1.15, 0) "New Salary" FROM emp;`

Output:

7900	JAMES	950.00	1093
7499	ALLEN	1600.00	1840
7521	WARD	1250.00	1438
7566	JONES	2975.00	3421
7654	MARTIN	1250.00	1438
7698	BLAKE	2850.00	3278
7782	CLARK	2450.00	2818
7788	SCOTT	3000.00	3450
7839	KING	5000.00	5750
7844	TURNER	1500.00	1725
7876	ADAMS	1100.00	1265
7902	FORD	3000.00	3450
7934	MILLER	1300.00	1495

Figure 35: Output for query 6

Explanation:

The above query calculates the new salary which is 15% more than the existing salary. This means:

$\text{new salary} = \text{salary} + (\text{salary} * 0.15) = \text{salary} * (1 + 0.15) = 1.15 * \text{salary}$
 Since the caption new salary consists of two words, we have to encapsulate these two words within double quotes.

7. `select lower(ename) "Name", length(ename) "Length" FROM emp where ename like 'A%' or ename like 'B%' or ename like 'C%' order by ename;`

Output:

Name	Length
adams	5
allen	5
blake	5
clark	5

Figure 36: Output for query 7

Explanation:

The above query makes use of the like operator to check for patterns. The use of the initcap function ensures that the first character of each name is capitalized. The length() function finds the length of each name.

8. `select ename, round(datediff(curdate(), hiredate)) MONTHS_WORKED from emp where job='CLERK' order by months_worked;`

Output:

ename	MONTHS_WORKED
ADAMS	359
MILLER	371
JAMES	372
SMITH	384

Figure 37: Output for query 8

Explanation:

The above query makes use of the date function months_between() to calculate the number of months between the hiredate and the current date. For more details regarding the months_between function, refer to the notes in the chapter. Note that the "order by" clause always comes at the end of the select query.

9. `select ename, lpad(sal, 15, '$') SALARY FROM emp`

Output:

ename	SALARY
SMITH	\$\$\$\$\$\$\$\$\$800.00
JAMES	\$\$\$\$\$\$\$\$\$950.00
ALLEN	\$\$\$\$\$\$\$\$\$1600.00
WARD	\$\$\$\$\$\$\$\$\$1250.00
JONES	\$\$\$\$\$\$\$\$\$2975.00
MARTIN	\$\$\$\$\$\$\$\$\$1250.00
BLAKE	\$\$\$\$\$\$\$\$\$2850.00
CLARK	\$\$\$\$\$\$\$\$\$2450.00
SCOTT	\$\$\$\$\$\$\$\$\$3000.00
KING	\$\$\$\$\$\$\$\$\$5000.00
TURNER	\$\$\$\$\$\$\$\$\$1500.00
ADAMS	\$\$\$\$\$\$\$\$\$1100.00
FORD	\$\$\$\$\$\$\$\$\$3000.00
MILLER	\$\$\$\$\$\$\$\$\$1300.00

Figure 38: Output for query 9

Explanation:

Since the salary has to be padded with the dollar sign towards the left, we can use the `lpad()` function. The `lpad()` function takes three arguments: the string, the total width and the character to be padded. So 15 characters are reserved for the salary, if the salary takes 4 digits then the remaining 11 digits are padded with the dollar sign. This is quite useful tactic to ensure that no digits are added to the left of the salary.

10. `select job, count(*) as "Total Employees" FROM emp GROUP BY job;`

Output:

job	Total Employees
ANALYST	2
CLERK	4
MANAGER	3
PRESIDENT	1
SALESMAN	4

Figure 39: Output for query 10

Explanation:

Since we need to find the total number of employees working in different job codes, we need to group them on the basis of the job id. We can then use the group function `count()` to count the total number of employees in each department.

2.6 Sample Exercises 2 with Solutions

1. Write a query to find the difference between the highest and lowest salaries. Label the column DIFFERENCE.
2. Write a query to display the last names of all employees who have a last name starting with 'A', 'L' or 'M'.
3. Write a query to count the number of employees whose last name ends with the letter 'n'.
4. Write a query to display all the job titles (job ids) in the departments 10 and 20.
5. Write a query to display the highest, lowest, sum, and average salary of all employees. Label the columns as Maximum, Minimum, Sum, and Average, respectively. Finally, round your results to the nearest whole number.
6. Write a query to find the number of managers without listing them. Label the column as Number of Managers. Hint: Use the MANAGER_ID column to determine the number of managers.
7. Write a query to display the names of all employees whose first name consists of exactly five characters.

2.7 Solutions to Sample Exercises 2

1. `select max(sal) - min(sal) Difference from emp;`
2. `select ename from emp where substr(ename,1,1) in ('A','L','M');`
3. `select count(*) from emp where ename like '%n';`
4. `select distinct job from emp where deptno in (10,20);`
5. `select round(max(sal),0) "Maximum", round(min(sal),0) "Minimum",
round(sum(sal),0) "Sum", round(avg(sal),0) "Average" FROM emp;`
6. `select count(distinct mgr) "Number of Managers" FROM emp;`
7. `select distinct ename from emp where length(ename)=5;`

Glossary

Database - Database is a collection of related data, in the context of MySQL[®] a database can be considered as a collection of database objects including tables, views, and other objects.

Database Management System - A database management system is a software package that helps administrators and users manage a database.

Field - A field in the context of a relational database is a column in a table. A field or column typically expresses a characteristic for example in the student table fields could be name of the student and the GPA of the student. A collection of tables forms a table.

Function - A function can be predefined or user-defined and is used to accomplish a particular task. The advantage of function is that the user does not have to define commonly used tasks rather the user creates a function or uses a predefined function whenever the task has to be accomplished.

Record - A record in the context of a relational database is a collection of fields. A record is also referred to as a row or a tuple.

Table - A table in the context of a relational database is a collection of records. A product table is a collection of several product records.

References

DuBois, P. 2007. *MySQL[®] Cookbook* (2nd ed.). New York, NY: O'Reilly Media.

Kofler, M. 2005. *The Definitive Guide to MySQL[®] 5 (Definitive Guides)* (2nd ed.). New Jersey, NJ: Apress.

Tahaghoghi, S.M.M., & Williams, H. 2006. *Learning MySQL[®]*. New York, NY: O'Reilly Media.

Additional Resources

Dyer, R. 2005. *MySQL® in a Nutshell*. New York, NY: O'Reilly Media.

MySQL® 5.6 Reference Manual. 2012.
<http://dev.mysql.com/doc/refman/5.6/en/index.html>

About the Author

Rao V.N. has completed Masters in Computer Applications (MCA) from Osmania University in India as well as Masters in Business Administration (MBA) from IUKB, Switzerland. He holds a certificate on Effective Online Tutoring from Department of Continuing Education, Oxford University. He holds several professional certifications including Oracle Certified SQL Expert, Oracle Certified Associate (OCA), Microsoft Certified Trainer (MCT), Microsoft Office Specialist (MOS), Sun Certified Java Professional (SCJP) and Cisco Certified Network Associate (CCNA). He has over 10 years of experience in teaching programming and database related courses at both graduate and under-graduate level.