Unit III: Database Management

Database Concepts: Introduction to database concepts and its need.

CONTENTS -
Relational data model: Concept of domain, relation, tuple, attribute, degree, cardinality, key, primary key, candidate key, alternate key and foreign key;
Structured Query Language:
General Concepts: Advantages of using SQL, Data Definition Language and Data Manipulation Language;

Data Types: number / decimal, character / varchar , date; SQL commands: CREATE TABLE, DROP TABLE, ALTER TABLE, UPDATE ....SET...., INSERT, DELETE; SELECT, DISTINCT, FROM, WHERE, IN, BETWEEN, LIKE, NULL / IS NULL, ORDER BY,GROUP BY, HAVING;

SQL functions: SUM ( ), AVG ( ), COUNT ( ), MAX ( ) and MIN ( );
Joins: equi-join and natural join

Interface of Python with an SQL database - Connecting SQL with Python - Creating Database connectivity Applications - Performing Insert, Update, Delete queries - Display data by using fetchone(), fetchall(), rowcount

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is

MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter DOUBLE PRECISION(size, d) format.

DECIMAL(size, d) An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.

DEC(size, d) Equal to DECIMAL(size,d)

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and Time data types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIME	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

```
mysql> CREATE DATABASE   test;
Query OK, 1 row affected (0.03 sec)
mysql> DROP DATABASE test;
Query OK, 0 rows affected (0.11 sec)
Mysql> use test
```

1. Write a SQL statement to create a simple table countries including columns country_id, country_name and region_id.

Sample Solution:

```
CREATE TABLE countries
(
COUNTRY_ID varchar(2),
COUNTRY_NAME varchar(40),
REGION_ID decimal(10,0)
);
```

Let execute the above code in MySQL 5.6 command prompt Here is the structure of the table:

```
mysql> DESC countries;
+----------          +----------    +----  +---      +------  +-----+
| Field              | Type         | Null |   Key    | Default | Extra |
+----------          +----------    +----  +---       +------+-----+
| COUNTRY_ID         | varchar(2)   | YES  |          | NULL    |       |
| COUNTRY_NAME | varchar(40)  | YES  |          | NULL    |       |
| REGION_ID          | decimal(10,0) | YES |          | NULL    |       |
+----------          +----------    +---- +---       +------  +-----+
3 rows in set (0.01 sec)
```

```
mysql> CREATE TABLE IF NOT EXISTS products (
         productID    INT UNSIGNED  NOT NULL AUTO_INCREMENT,
         productCode  CHAR(3)       NOT NULL DEFAULT '',
         name         VARCHAR(30)   NOT NULL DEFAULT '',
         quantity     INT UNSIGNED  NOT NULL DEFAULT 0,
         price        DECIMAL(7,2)  NOT NULL DEFAULT 99999.99,
         PRIMARY KEY  (productID)
       );
Query OK, 0 rows affected (0.08 sec)
 -- Show all the tables to confirm that the "products" table has been created
mysql> SHOW TABLES;
+-------------+
| products    |
+-------------+

-- Describe the fields (columns) of the "products" table
mysql> DESCRIBE products;
+-------------+------------------+------+-----+----------+----------------+
| Field       | Type             | Null | Key | Default  | Extra          |
+-------------+------------------+------+-----+----------+----------------+
| productID   | int(10) unsigned | NO   | PRI | NULL     | auto_increment |
| productCode | char(3)          | NO   |     |          |                |
| name        | varchar(30)      | NO   |     |          |                |
| quantity    | int(10) unsigned | NO   |     | 0        |                |
| price       | decimal(7,2)     | NO   |     | 99999.99 |                |
```

2.Write a SQL statement to create a table countries set a constraint NULL.

Sample Solution:

```
CREATE TABLE IF NOT EXISTS countries
(
COUNTRY_ID varchar(2) NOT NULL,
COUNTRY_NAME varchar(40) NOT NULL,
REGION_ID decimal(10,0) NOT NULL
);
```

Let execute the above code in MySQL 5.6 command prompt

Here is the structure of the table:

```
mysql> desc countries;
+----------          +-----------    +----  +---   +-----   ---+-----+
| Field                 | Type              Null   | Key    | Default | Extra |
+----------          +-----------      +---- +---   +-------   +-----+
| COUNTRY_ID        | varchar(2)     | NO     |          | NULL    |        |
| COUNTRY_NAME | varchar(40)    | NO     |         | NULL    |        |
| REGION_ID          | decimal(10,0) | NO   |          | NULL    |        |
+----------          +-----------      +----+---  +------   +-----+
3 rows in set (0.00 sec)
```

3.Write a SQL statement to create a table named jobs including columns job_id, job_title, min_salary, max_salary and check whether the max_salary amount exceeding the upper limit 25000.

Sample Solution:

```
CREATE TABLE jobs (
JOB_ID varchar(10) NOT NULL ,
JOB_TITLE varchar(35) NOT NULL,
MIN_SALARY decimal(6,0),
MAX_SALARY decimal(6,0)
CHECK(MAX_SALARY<=25000)
);
```

4.Write a SQL statement to create a table named job_history including columns employee_id, start_date, end_date, job_id and department_id and make sure that the value against column end_date will be entered at the time of insertion to the format like '--/–/—'.

Sample Solution:

```
CREATE TABLE  job_history (
EMPLOYEE_ID decimal(6,0) NOT NULL,
START_DATE date NOT NULL,
END_DATE date NOT NULL
CHECK (END_DATE LIKE '--/–/—'),
JOB_ID varchar(10) NOT NULL,
DEPARTMENT_ID decimal(4,0) NOT NULL
);
```

5.Write a SQL statement to create a table named countries including columns country_id,country_name and region_id and make sure that no duplicate data against column country_id will be allowed at the time of insertion.

Sample Solution:

```
CREATE TABLE  countries
(
COUNTRY_ID varchar(2) NOT NULL,
COUNTRY_NAME varchar(40) NOT NULL,
REGION_ID decimal(10,0) NOT NULL,
UNIQUE(COUNTRY_ID)
);
```

6.Write a SQL statement to create a table named jobs including columns job_id, job_title, min_salary and max_salary, and make sure that, the default value for job_title is blank and min_salary is 8000 and max_salary is NULL will be entered automatically at the time of insertion if no value assigned for the specified columns.

Sample Solution:

```
CREATE TABLE I jobs
(
JOB_ID varchar(10) NOT NULL UNIQUE,
JOB_TITLE varchar(35) NOT NULL DEFAULT ' ',
MIN_SALARY decimal(6,0) DEFAULT 8000,
MAX_SALARY decimal(6,0) DEFAULT NULL
);
```

7. Write a SQL statement to create a table countries including columns country_id, country_name and region_id and make sure that the column country_id will be unique and store an auto incremented value.

```
CREATE TABLE countries
(
COUNTRY_ID integer NOT NULL UNIQUE AUTO_INCREMENT PRIMARY KEY,
COUNTRY_NAME varchar(40) NOT NULL,
REGION_ID decimal(10,0) NOT NULL
);
```

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

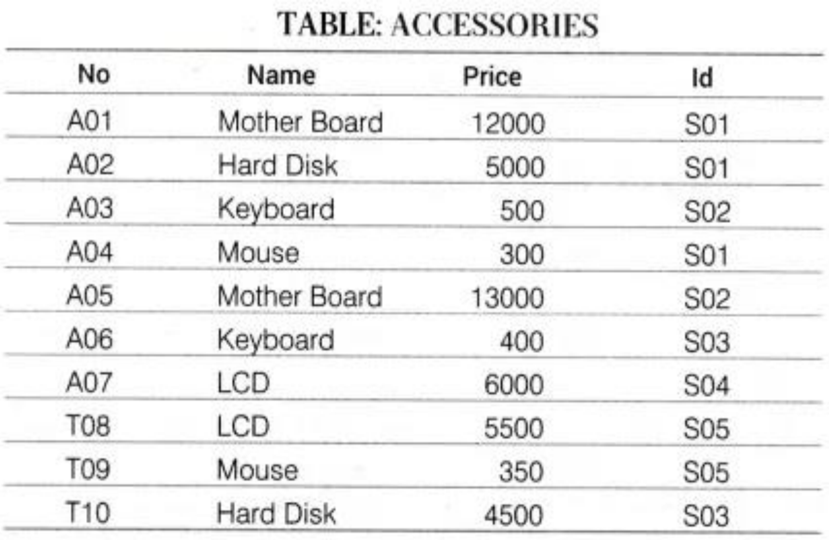


TABLE: ACCESSORIES

No	Name	Price	Id
A01	Mother Board	12000	S01
A02	Hard Disk	5000	S01
A03	Keyboard	500	S02
A04	Mouse	300	S01
A05	Mother Board	13000	S02
A06	Keyboard	400	S03
A07	LCD	6000	S04
T08	LCD	5500	S05
T09	Mouse	350	S05
T10	Hard Disk	4500	S03

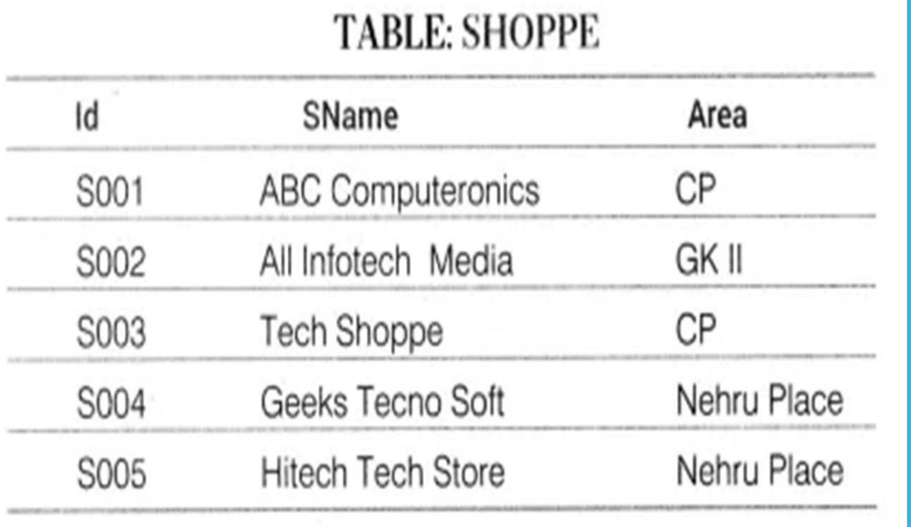


TABLE: SHOPPE

Id	SName	Area
S001	ABC Computeronics	CP
S002	All Infotech Media	GK II
S003	Tech Shoppe	CP
S004	Geeks Tecno Soft	Nehru Place
S005	Hitech Tech Store	Nehru Place

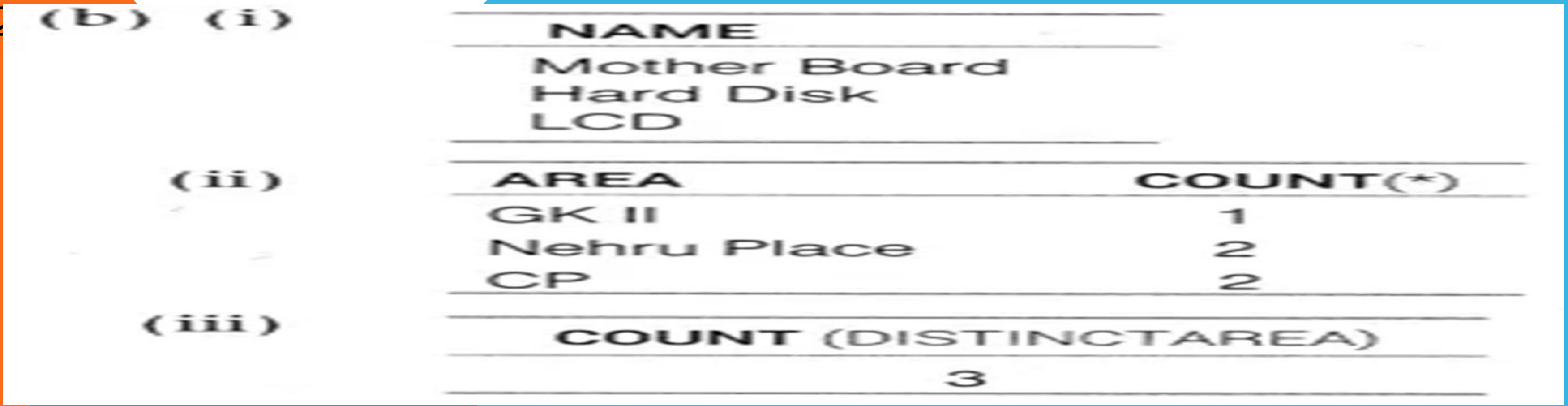
(b) Write the output of the following SQL commands:

(i) SELECT DISTINCT NAME FROM ACCESSORIES WHERE PRICE>=5000;

(ii) SELECT AREA, COUNT(*) FROM SHOPPE GROUP BY AREA;

(iii) SELECT COUNT (DI ST INCT AREA) FROM SHOPPE:

(iv) SELECT NAME, PRICE*0.05 DISCOUNT FROM ACCESSORIES WHERE SNO IN ('S02



(b) (i)

NAME
Mother Board
Hard Disk
LCD

(ii)

AREA	COUNT(*)
GK II	1
Nehru Place	2
CP	2

(iii)

COUNT (DISTINCTAREA)
3

iv) The given query will result in an error as there is no column named SNo in ACCESSORIES table.

a) Write the SQL queries:

(i) To display Name and Price of all the Accessories in ascending order of their Price.
(ii) To display Id and SName of all Shoppe located in Nehru Place.
(iii) To display Minimum and Maximum Price of each Name of Accessories.
(iv) To display Name, Price of all Accessories and their respective SName, where they are available.

a) (i) SELECT Name, Price FROM ACCESSORIES ORDER BY Price:
(ii) SELECT Id, SName FROM SHOPPE WHERE Area ='Nehru Place';
(iii) SELECT MIN(Price) "Minimum Price", MAX(Price)"Maximum Price", Name FROM ACCESSORIES GROUP BY Name:
(iv) SELECT Name, Price, SName FROM ACCESSORIES A, SHOPPE S WHERE A.Id = S.Id;

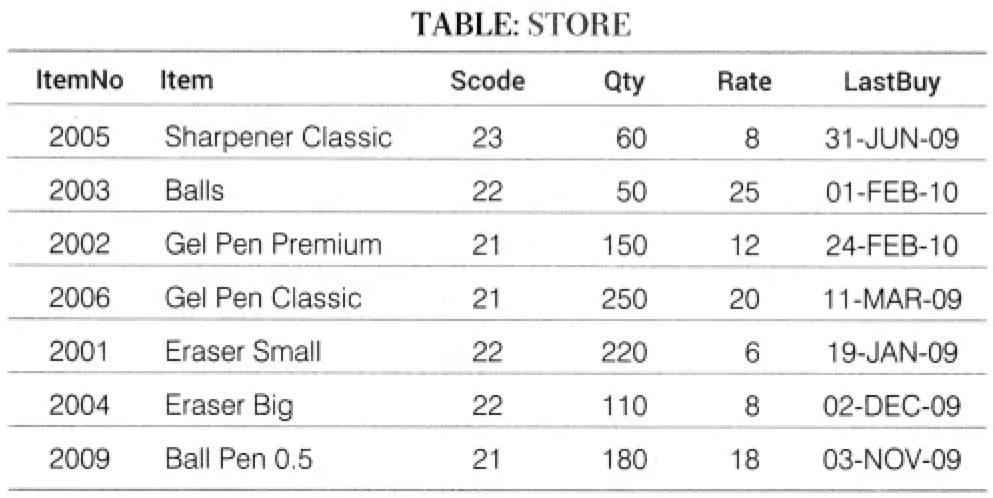


TABLE: STORE

ItemNo	Item	Scode	Qty	Rate	LastBuy
2005	Sharpener Classic	23	60	8	31-JUN-09
2003	Balls	22	50	25	01-FEB-10
2002	Gel Pen Premium	21	150	12	24-FEB-10
2006	Gel Pen Classic	21	250	20	11-MAR-09
2001	Eraser Small	22	220	6	19-JAN-09
2004	Eraser Big	22	110	8	02-DEC-09
2009	Ball Pen 0.5	21	180	18	03-NOV-09

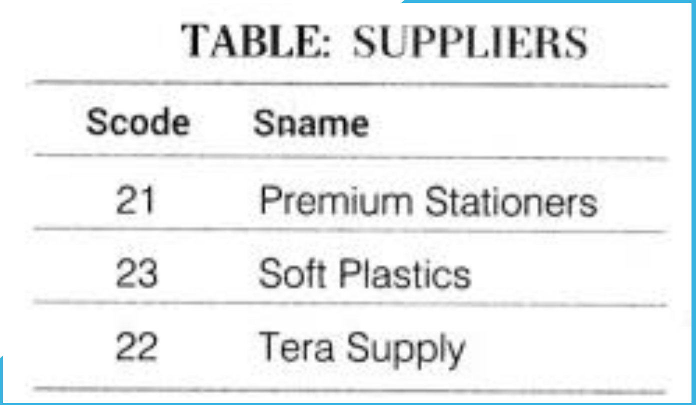


TABLE: SUPPLIERS

Scode	Sname
21	Premium Stationers
23	Soft Plastics
22	Tera Supply

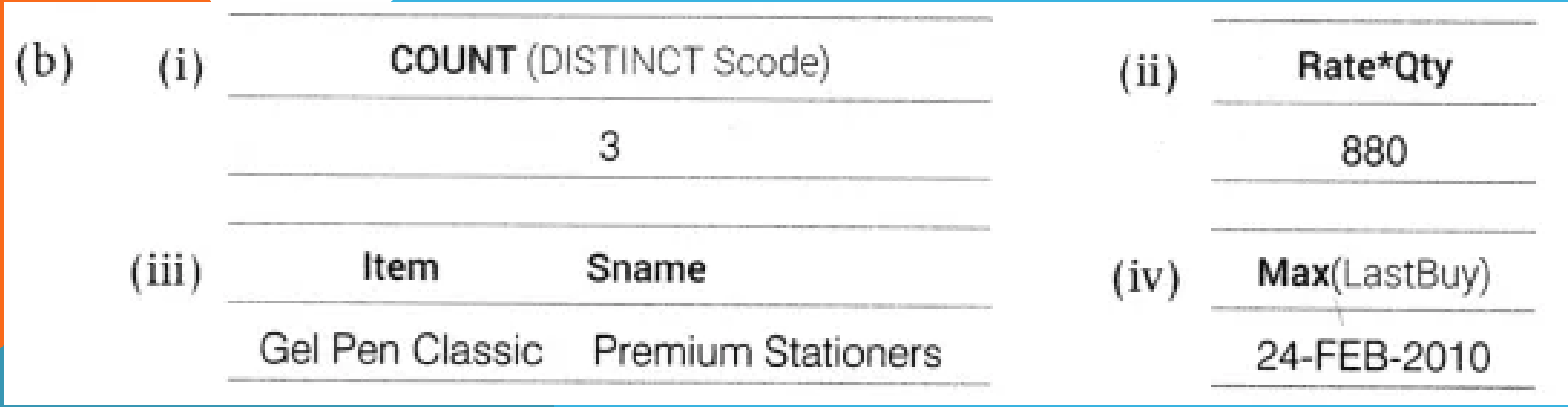
(b) Give the output of the following SQL queries:

(i) SELECT COUNT(DISTINCT Scode) FROM STORE;

(ii) SELECT Rate * Qty FROM STORE WHERE ItemNo = 2004;

(iii) SELECT Item, Sname FROM STORE S, SUPPLIERS P
WHERE S.Scode=P.Scode AND ItemNo=2006;

(iv) SELECT MAX(LastBuy) FROM STORE;



(b) (i)

COUNT (DISTINCT Scode)
3

(ii)

Rate*Qty
880

(iii)

Item	Sname
Gel Pen Classic	Premium Stationers

(iv)

Max(LastBuy)
24-FEB-2010

(a) Write SQL commands for the following statements:
(i) To display details of all the items in the STORE table in ascending order of LastBuy.
(ii) To display ItemNo and Item name of those items from STORE table, whose Rate is more than Rs. 15.
(Hi) To display the details of those items whose Supplier code (Scode) is 22 or Quantity in Store (Qty) is more than 110 from the table STORE.
(iv) To display minimum rate of items for each Supplier individually as per Scode from the table STORE.

(a) (i) SELECT * FROM STORE ORDER BY LastBuy;
(ii) SELECT ItemNo, Item FROM STORE WHERE Rate>15;
(iii) SELECT * FROM STORE WHERE Scode = 22 OR Qty>110;
(iv) SELECT MIN(Rate) FROM STORE GROUP BY Scode;

SQL ALTER TABLE Statement
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

SQL ALTER TABLE Example

Look at the "Persons" table:

ID	LastName	FirstName	Address	City
1	Hansen Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove Borgvn 23		Sandnes
3	Pettersen	Kari Storgt 20		Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

The "Persons" table will now look like this:

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen Ola	Timoteivn 10	Sandnes		
2	Svendson	Tove Borgvn 23		Sandnes	
3	Pettersen	Kari Storgt 20		Stavanger	

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two- or four-digit format.

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

The "Persons" table will now look like this:

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

```
SELECT  5 * 8 FROM DUAL;
40
SELECT 45/5 FROM DUAL;
9
```

```
mysql> SELECT * FROM products;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |
|      1002 | PEN         | Pen Blue  |     8000 |  1.25 |
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+
```

Logical Operators - AND, OR, NOT, XOR
You can combine multiple conditions with boolean operators AND, OR, XOR. You can also invert a condition using operator NOT. For examples,

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %';
+-----------+-------------+----------+----------+-------+
| productID | productCode | name     | quantity | price |
+-----------+-------------+----------+----------+-------+
|      1001 | PEN         | Pen Red  |     5000 |  1.23 |
|      1002 | PEN         | Pen Blue |     8000 |  1.25 |
+-----------+-------------+----------+----------+-------+
mysql> SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name
LIKE 'Pen %';
+-----------+-------------+---------+----------+-------+
| productID | productCode | name    | quantity | price |
+-----------+-------------+---------+----------+-------+
|      1001 | PEN         | Pen Red |     5000 |  1.23 |
+-----------+-------------+---------+----------+-------+

mysql> SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %');
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+
```

IN, NOT IN

You can select from members of a set with IN (or NOT IN) operator. This is easier and clearer than the equivalent AND-OR expression.

```
mysql> SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black');
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
+-----------+-------------+-----------+----------+-------+
```

BETWEEN, NOT BETWEEN

To check if the value is within a range, you could use BETWEEN ... AND ... operator. Again, this is easier and clearer than the equivalent AND-OR expression.

```
mysql> SELECT * FROM products
       WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000);
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
+-----------+-------------+-----------+----------+-------+
```

IS NULL, IS NOT NULL

NULL is a special value, which represent "no value", "missing value" or "unknown value". You can checking if a column contains NULL by IS NULL or IS NOT NULL. For example,

```
mysql> SELECT * FROM products WHERE productCode IS NULL;
Empty set (0.00 sec)
```

Using comparison operator (such as = or <>) to check for NULL is a mistake - a very common mistake. For example,

```
SELECT * FROM products WHERE productCode = NULL;
-- This is a common mistake. NULL cannot be compared.
```

```
CREATE TABLE shop (
   article INT UNSIGNED  DEFAULT '0000' NOT NULL,
   dealer  CHAR(20)      DEFAULT ''     NOT NULL,
   price   DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
   PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
   (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
   (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

AS - Alias

You could use the keyword AS to define an alias for an identifier (such as column name, table name). The alias will be used in displaying the name. It can also be used as reference. For example,

```
mysql> SELECT productID AS ID, productCode AS Code,   name AS Description, price AS `Unit
           Price`        FROM products           -- Define aliases to be used as display names
             ORDER BY ID;                        -- Use alias ID as reference
+------+------+-------------+------------+
| ID   | Code | Description | Unit Price |
+------+------+-------------+------------+
| 1001 | PEN  | Pen Red     |       1.23 |
| 1002 | PEN  | Pen Blue    |       1.25 |
| 1003 | PEN  | Pen Black   |       1.25 |
| 1004 | PEC  | Pencil 2B   |       0.48 |
| 1005 | PEC  | Pencil 2H   |       0.49 |
+------+------+-------------+------------+
```

Take note that the identifier "Unit Price" contains a blank and must be back-quoted.

Function CONCAT()

You can also concatenate a few columns as one (e.g., joining the last name and first name) using function CONCAT(). For example,

```
+---------------+------+
mysql> SELECT CONCAT(productCode, ' - ', name) AS `Product Description`, price FROM products;
+--------------------+-------+
| Product Description | price |
+--------------------+-------+
| PEN - Pen Red       |  1.23 |
| PEN - Pen Blue      |  1.25 |
| PEN - Pen Black     |  1.25 |
| PEC - Pencil 2B     |  0.48 |
| PEC - Pencil 2H     |  0.49 |
```

****