[Object-]Relational Databases Management Systems - 2020

Claus Kaldeich©

```
--Part One: Relational Model, MySQL, MariaDB & related stuff
--Script for the DBMS course at ISLA-IPGT
--Review & exercises
--
create database db3 charset latin1 collate latin1_swedish_ci;
--
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 99.99,
PRIMARY KEY  (productID)
);
--Note: There is no engine declaration...
--
DESCRIBE products;
--
SHOW CREATE TABLE products;
--
--Better using  \G
Try:
SHOW CREATE TABLE products \G
--
--Single insert w/ value:
--Note, that the table was defined with the option 'auto_increment' for  productID.
--
INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);
--
--check table products:
select * from products;
--or
select * from products \G
--
INSERT INTO products VALUES
(NULL, 'PEN', 'Pen Blue',  8000, 1.25),
(NULL, 'PEN', 'Pen Black', 2000, 1.25);
--Chec the table
--Note: how the system applied the 'auto_increment'?
--
--Insert value to selected columns
INSERT INTO products (productCode, name, quantity, price)
VALUES
('PEC', 'Pencil 2B', 10000, 0.48),
('PEC', 'Pencil 2H', 8000, 0.49);
–
Check it again:
The missing value for the auto_increment column also results in max_value
+ 1.
--
--Missing values get default values; insert:
INSERT INTO products (productCode, name) VALUES ('PEC', 'Pencil HB');
--
```

```
--2nd attribute (productCode) is defined to be NOT NULL; check it:
INSERT INTO products values (NULL, NULL, NULL, NULL, NULL);
--

--Delete last inserted row from the table products
DELETE FROM products WHERE productID = 1006;
--Alternativley use SET to set the values:
INSERT INTO tableName SET column1=value1, column2=value2, ...
--The statement below is equivalent to the statement above:
INSERT INTO tableName (column1, column2, ...) VALUES (value1, value2, ...)
-- --

--Miscellaneous:

SELECT User, Host FROM mysql.user;
--
SELECT User, Host, Password, password_expired FROM mysql.user;
--
About 'select' without table:

SELECT NOW();
--
SELECT NOW() time;
--
SELECT NOW() time, sqrt(2) square_root;
--
Exercises:
Perform all commands above on your sql-terminal.
-- --

--Comparison operators for numbers (INT, DECIMAL, FLOAT):
'='            (equal to),
'<>' or '!='   (not equal to),
'>'            (greater than),
'<'            (less than),
'>='           (greater than or equal to),
'<='           (less than or equal to).

For example, price > 1.0, quantity <= 500.
--Check:
SELECT name, price FROM products WHERE price < 1.0;
--
SELECT name, quantity FROM products WHERE quantity <= 2000;
--
Do not compare FLOATs (real numbers) for equality ('=' or '<>'), as they are not
precise.
On the other hand, DECIMAL are precise.
--
For strings, in addition to full matching using operators like '=' and '<>', we can
perform pattern matching using operator LIKE (or NOT LIKE) with wildcard
characters.
```

```
The wildcards:
'_' matches any single character;
'%' matches any number of characters (including zero).

For example:
'abc%'    matches strings beginning with 'abc';
'%xyz'    matches strings ending with 'xyz';
'%aaa%'   matches strings containing 'aaa';
'___'     attaches strings containing exactly three characters ( 3 x _ );

'a_b%'    matches strings beginning with 'a', followed by any single
          character, followed by 'b', followed by zero or more
          characters.

For strings, you could also use '=', '<>', '>', '<', '>=', '<=' to compare two
strings (e.g., productCode = 'PEC').
--
Important note: The ordering of string depends on the so-called “collation” chosen
(the charset and collation (e.g., “collate latin1_swedish_ci”) are defined, when
the database is created, so give attention to this detail in order to get the
symbols and letters related to you language).
--
For example:
SELECT name, price FROM products WHERE productCode = 'PEN';
--Try also:
SELECT name, price FROM products WHERE productCode > 'PEN';
--
SELECT name, price FROM products WHERE productCode < 'PEN';
--
--"name" begins with 'PENCIL'
SELECT name, price FROM products WHERE name LIKE 'PENCIL%';
--
--"name" begins with 'P', followed by any two characters, followed by space,
followed by zero or more characters:
SELECT name, price FROM products WHERE name LIKE 'P__  %';
--
MariaDB/MySQL also support 'regular expressions' ('regexp') matching via the REGEXP
operator:
SELECT 'Pencil' REGEXP 'Pencil';
--
SELECT 'Pencil' REGEXP 'pencil';
--
SELECT BINARY 'Pencil' REGEXP 'pencil';
--
Note that the word being matched must match the whole pattern. Check it:
SELECT 'Pencil' REGEXP 'Penci';
--
SELECT 'Penci' REGEXP 'Pencil';
--
A match can be performed against more than one word with the '|' character (pipe).
For example:
SELECT 'Pencil' REGEXP 'Book|Pencil';
--
```

```
SELECT 'Pencil' REGEXP 'Book|Pen';
--
SELECT 'Pen' REGEXP 'Book|Pencil';
--Check it.
--
Special Characters:
The above examples introduce the syntax, but are not very useful.
It's the special characters that give regular expressions their power.
--
^ matches the beginning of a string (inside square brackets it can also mean NOT.
SELECT 'Pencil' REGEXP '^Pe';
--
$ matches the end of a string:
SELECT 'Pencil' REGEXP 'il$';
--
A dot '.' matches any single character:
SELECT 'Pencil' REGEXP 'Pen.il';
--
SELECT 'Pencil' REGEXP 'Pe..il';
--
x* matches zero or more of a character 'x'. In the examples below, it's the 'n'
character:
SELECT 'Pencil' REGEXP 'Pe*cil';
--
SELECT 'Pencil' REGEXP 'Pen*cil';
--
SELECT 'Pennnncil' REGEXP 'Pen*cil';
--
x? matches zero or one of a character 'x'. In the examples below, it's the 'n'
character.
SELECT 'Pencil' REGEXP 'Pen?cil';
+---------------------------+
| 'Pencil' REGEXP 'Pen?cil' |
+---------------------------+
|                         1 |
+---------------------------+

SELECT 'Pecil' REGEXP 'Pen?cil';
+--------------------------+
| 'Pecil' REGEXP 'Pen?cil' |
+--------------------------+
|                        1 |
+--------------------------+

SELECT 'Marrria' REGEXP 'Mar?ia';
+---------------------------+
| 'Marrria' REGEXP 'Mar?ia' |
+---------------------------+
|                         0 |
+---------------------------+
```

```
SELECT 'Pennncil' REGEXP 'Pen?cil';
+-----------------------------+
| 'Pennncil' REGEXP 'Pen?cil' |
+-----------------------------+
|                           0 |
+-----------------------------+
--
REGEX is a very large subject, check the literature to get more information about
'regular expressions'.
--
--Arithmetic Operators
You can perform arithmetic operations on numeric fields using arithmetic operators,
as tabulated below:
Operator              Description
+                     Addition
-                     Subtraction
*                     Multiplication
/                     Division
DIV                   Integer Division
%                     Modulus (Remainder)
--
Logical Operators     AND, OR, NOT, XOR (short: 'logop' or 'LOGOP')
--
It is possible to combine multiple conditions with boolean operators AND, OR, XOR.
You may also invert a condition using operator NOT.

For examples:
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 |
+-----------+-------------+----------+----------+-------+

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 |
+-----------+-------------+---------+----------+-------+

SELECT * FROM products WHERE quantity >= 5000 AND NOT price < 1.24;
+-----------+-------------+----------+----------+-------+
| productID | productCode | name     | quantity | price |
+-----------+-------------+----------+----------+-------+
|      1002 | PEN         | Pen Blue |     8000 |  1.25 |
+-----------+-------------+----------+----------+-------+
```

```
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 |
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+

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 |
+-----------+-------------+----------+----------+-------+

NOT:
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 |
+-----------+-------------+-----------+----------+-------+

Exercise:
SELECT * FROM products WHERE NOT (quantity >= 5000 AND name
LIKE '.......');
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
+-----------+-------------+-----------+----------+-------+

XOR:
SELECT * FROM products WHERE quantity >= 5000 XOR 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 |
+-----------+-------------+-----------+----------+-------+
Why? Explain the semantic of the query above.

--
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.
```

```
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 |
+-----------+-------------+-----------+----------+-------+
```

Exercise. Try by yourself: Include the NOT.

--

BETWEEN, NOT BETWEEN

To check if the value is within a range, use the BETWEEN ... AND ... operator.
Note: This is easier and clearer than the equivalent AND-OR expression.

```
SELECT * FROM products WHERE (price BETWEEN 1.0 AND 2.0);
+-----------+-------------+-----------+----------+-------+
| 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 |
+-----------+-------------+-----------+----------+-------+

SELECT * FROM products WHERE NOT (price BETWEEN 1.0 AND 2.0);
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+

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 |
+-----------+-------------+-----------+----------+-------+
```

Exercise:
Try it by yourself: Introduce the logpop NOT and check the results.

--

IS NULL, IS NOT NULL

NULL is a special value, which represent "no value", "missing value" or "unknown value".
It is possible to check, if a column contains NULL by IS NULL or IS NOT NULL.

For example:

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

--
Note: Using comparison operators (such as = or <>) to check for NULL is a mistake - a very common mistake:
NULL cannot be compared.

--
ORDER BY Clause

You can order the rows selected using ORDER BY clause, with the following syntax:
SELECT ... FROM tableName
WHERE criteria
ORDER BY columnA ASC|DESC, columnB ASC|DESC, ...

Note: If several rows have the same value in columnA, it will be ordered according to columnB, and so on.
For strings, the ordering could be case-sensitive or case-insensitive, depending on the so-called character
collating sequence used.
--
For examples:
Order the results by price in descending order:
SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;

productID	productCode	name	quantity	price
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1001	PEN	Pen Red	5000	1.23

Order by price in descending order, followed by quantity in ascending (default) order:
SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;

productID	productCode	name	quantity	price
1003	PEN	Pen Black	2000	1.25
1002	PEN	Pen Blue	8000	1.25
1001	PEN	Pen Red	5000	1.23

Exercises.
Check the following statements:
SELECT productID, productCode, quantity FROM products ORDER BY ... . , ...;

```
+-----------+-------------+----------+
| productID | productCode | quantity |
+-----------+-------------+----------+
|      1003 | PEN         |     2000 |
|      1001 | PEN         |     5000 |
|      1002 | PEN         |     8000 |
|      1005 | PEC         |     8000 |
|      1004 | PEC         |    10000 |
+-----------+-------------+----------+
--
SELECT productID,name,quantity,price,(.... .. ....) FROM products ORDER BY ... ;
+-----------+-----------+----------+-------+--------------------+
| productID | name      | quantity | price | (                ) |
+-----------+-----------+----------+-------+--------------------+
|      1003 | Pen Black |     2000 |  1.25 |            2500.00 |
|      1005 | Pencil 2H |     8000 |  0.49 |            3920.00 |
|      1004 | Pencil 2B |    10000 |  0.48 |            4800.00 |
|      1001 | Pen Red   |     5000 |  1.23 |            6150.00 |
|      1002 | Pen Blue  |     8000 |  1.25 |           10000.00 |
+-----------+-----------+----------+-------+--------------------+

--
LIMIT Clause

A SELECT query on a large database may produce many rows. It is possible to use the
LIMIT clause to limit the number of rows displayed.

Examples.
Display the first two rows of a select statement:
SELECT * FROM products ORDER BY price LIMIT 2;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+
--
To continue to the following records, specify the number of rows to be skipped,
followed by the number of rows to be displayed in the LIMIT clause.

Example.
Skip the first two rows and display the next 1 row:
SELECT * FROM products ORDER BY price LIMIT 2, 1;


+-----------+-------------+---------+----------+-------+
| productID | productCode | name    | quantity | price |
+-----------+-------------+---------+----------+-------+
|      1001 | PEN         | Pen Red |     5000 |  1.23 |
+-----------+-------------+---------+----------+-------+

How?
```

Step 1:
```
SELECT * FROM products ORDER BY price;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |Step 2
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |Step 2
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |Step 3
|      1002 | PEN         | Pen Blue  |     8000 |  1.25 |
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
+-----------+-------------+-----------+----------+-------+
```

Step 2:
```
SELECT * FROM products ORDER BY price LIMIT 2;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+
```

Step 3:
```
SELECT * FROM products ORDER BY price LIMIT 2,1;
+-----------+-------------+---------+----------+-------+
| productID | productCode | name    | quantity | price |
+-----------+-------------+---------+----------+-------+
|      1001 | PEN         | Pen Red |     5000 |  1.23 |
+-----------+-------------+---------+----------+-------+
```

Exercises:
```
SELECT * FROM products ORDER BY price LIMIT ... , ... ;
+-----------+-------------+----------+----------+-------+
| productID | productCode | name     | quantity | price |
+-----------+-------------+----------+----------+-------+
|      1001 | PEN         | Pen Red  |     5000 |  1.23 |
|      1002 | PEN         | Pen Blue |     8000 |  1.25 |
+-----------+-------------+----------+----------+-------+

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

```
SELECT * FROM products .......  ....  ......... LIMIT ... ;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |
+-----------+-------------+-----------+----------+-------+
2 rows in set (0.00 sec)

SELECT * FROM products .......  ....  ......... LIMIT ... , ... ;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+

--
AS - Alias

The keyword 'AS' to define an alias, which is 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:
SELECT productID AS ID, productCode AS Code,name AS Description, price AS 'Unit
Price'
FROM products
ORDER BY ID;
+------+------+-------------+------------+
| 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 |
+------+------+-------------+------------+
Note: The identifier (also called "alias") "Unit Price" contains a blank and must
be (back-) quoted.

Exercises.
SELECT productID,name,quantity,price, (..... .. .....) AS ..... FROM products ORDER
BY ... ;
+-----------+-----------+----------+-------+----------+
| productID | name      | quantity | price |          |
+-----------+-----------+----------+-------+----------+
|      1003 | Pen Black |     2000 |  1.25 |  2500.00 |
|      1005 | Pencil 2H |     8000 |  0.49 |  3920.00 |
|      1004 | Pencil 2B |    10000 |  0.48 |  4800.00 |
|      1001 | Pen Red   |     5000 |  1.23 |  6150.00 |
|      1002 | Pen Blue  |     8000 |  1.25 | 10000.00 |
+-----------+-----------+----------+-------+----------+
```

CONCAT()

To concatenate a few columns as one (e.g., joining the last name and first name) by using function CONCAT().

For example:
```
SELECT CONCAT(productCode,' - ',name) AS 'Product Description', price 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 |
+---------------------+-------+
```

See below is 2 ways to sort the output by 'Product Description' (columns 1+2) or by 'Price'.

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

or,
```
SELECT CONCAT(productCode,' - ',name) AS 'Product Description', price Price FROM products
ORDER BY 'Product Description';
+---------------------+-------+
| 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 |
+---------------------+-------+
```

Note: It is possible to use an ordinal to refers the attribute or column.
That means, the ordinal refers to the position of the attribute after the SELECT.

--

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

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

or,
SELECT CONCAT(productCode,' - ',name) AS 'Product Description', price Price FROM
products order by 1;
+---------------------+-------+
| Product Description | Price |
+---------------------+-------+
| PEC - Pencil 2B     |  0.48 |
| PEC - Pencil 2H     |  0.49 |
| PEN - Pen Black     |  1.25 |
| PEN - Pen Blue      |  1.25 |
| PEN - Pen Red       |  1.23 |
+---------------------+-------+

Exercises.
Create one-column list of products with the name and ten-pack price with 10%
discount.
SELECT CONCAT(name,'     ',((..... * ....) .. .....),'$') AS 'Product 10-pack 10%
discount' FROM products;
+------------------------------+
| Product 10-pack 10% discount |
+------------------------------+
| Pen Red        11.07$        |
| Pen Blue        11.25$       |
| Pen Black        11.25$      |
| Pencil 2B        4.32$       |
| Pencil 2H        4.41$       |
+------------------------------+
```

DISTINCT

A column may have duplicate values, so use keyword DISTINCT to select only distinct values.
Apply also apply DISTINCT to several columns to select distinct combinations of these columns. For example:

```
SELECT price FROM products;
+-------+
| price |
+-------+
|  1.23 |
|  1.25 |
|  1.25 |
|  0.48 |
|  0.49 |
+-------+

SELECT DISTINCT price FROM products;
+-------+
| price |
+-------+
|  1.23 |
|  1.25 |
|  1.25 |
|  0.48 |
|  0.49 |
+-------+
```

Remove duplicates:

```
SELECT DISTINCT price AS 'Distinct Prices' FROM products;
+-----------------+
| Distinct Prices |
+-----------------+
|            1.23 |
|            1.25 |
|            0.48 |
|            0.49 |
+-----------------+
```

But...

```
SELECT DISTINCT price, name FROM products;
+-------+-----------+
| price | name      |
+-------+-----------+
|  1.23 | Pen Red   |
|  1.25 | Pen Blue  |
|  1.25 | Pen Black |
|  0.48 | Pencil 2B |
|  0.49 | Pencil 2H |
+-------+-----------+
```

and

```
SELECT  price, name FROM products;
+-------+-----------+
| price | name      |
+-------+-----------+
|  1.23 | Pen Red   |
|  1.25 | Pen Blue  |
|  1.25 | Pen Black |
|  0.48 | Pencil 2B |
|  0.49 | Pencil 2H |
+-------+-----------+
```

Why?
Exerxise: Explain the reason why.

--
GROUP BY Clause

The GROUP BY clause allows you to collapse multiple records with a common value into groups.

For example:

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

SELECT * FROM products GROUP BY productCode; -- Only first record's name in each group is shown.
+-----------+-------------+----------------------+----------+-------+
| productID | productCode | name                 | quantity | price |
+-----------+-------------+----------------------+----------+-------+
|      1006 | NTB         | Notebook A4L90p      |     9000 |  3.30 |
|      1010 | PAP         | Paper A4 white 80/500 |     1100 |  5.00 |
|      1004 | PEC         | Pencil 2B            |    10000 |  0.48 |
|      1001 | PEN         | Pen Red              |     5000 |  1.23 |
|      1012 | RUB         | Rubber white Soft    |     1200 |  1.99 |
+-----------+-------------+----------------------+----------+-------+

SELECT * FROM products GROUP BY productCode ORDER BY 1; -- '1' is an ordinal and refers to the first column.
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
+-----------+-------------+-----------+----------+-------+
```

GROUP BY Aggregate Functions: COUNT, MAX, MIN, AVG, SUM, STD, GROUP_CONCAT:
GROUP BY used together with GROUP BY aggregate functions (such as COUNT(), AVG(), SUM()) to produce group summary.

The function COUNT(*) returns the rows selected; and
COUNT(columnName) counts only the non-NULL values of the given column.
An easy example:

```
SELECT COUNT(*) AS 'Nr. of codes' FROM products;
+--------------+
| Nr. of codes |
+--------------+
|            5 |
+--------------+
--
More inserts:
INSERT INTO products (productCode, name, quantity, price) VALUES
('NTB', 'Notebook A4L90p', 9000, 3.30),
('NTB', 'Notebook B5L 60p', 8000, 2.80),
('NTB', 'Notebook A5L 50p', 70000, 2.20),
('NTB', 'Notebook B6L 40p', 70000, 2.00);
--
Check number of products by productCode:
SELECT productCode AS Code,COUNT(*) AS 'Items by Code' FROM products WHERE
productCode='NTB';

+------+---------------+
| Code | Items by Code |
+------+---------------+
| NTB  |             4 |
+------+---------------+
--
Setting and using runtime variables:
SET @productCode='NTB';
SELECT productCode AS Code, COUNT(*) AS 'Items by Code' FROM products WHERE
productCode=@productCode;
+------+---------------+
| Code | Items by Code |
+------+---------------+
| NTB  |             4 |
+------+---------------+
--
More inserts:
INSERT INTO products (productCode, name, quantity, price) VALUES
('RUB', 'Rubber white Soft', 1200, 1.99),
('RUB', 'Rubber color Medium', 1500, 0.99);
--
Check the table, run:
select * from products;
--
SELECT productCode AS Item,COUNT(*) AS 'Items by Code' FROM products GROUP BY
productCode;
```

```
+------+---------------+
| Item | Items by Code |
+------+---------------+
| NTB  |             4 |
| PAP  |             2 |
| PEC  |             2 |
| PEN  |             3 |
| RUB  |             2 |
+------+---------------+
```

Or

```
SELECT productCode AS Item,COUNT(*) AS 'Items by Code' FROM products GROUP BY
productCode ORDER BY 2;
+------+---------------+
| Item | Items by Code |
+------+---------------+
| PEC  |             2 |
| PAP  |             2 |
| RUB  |             2 |
| PEN  |             3 |
| NTB  |             4 |
+------+---------------+
```

GROUP BY aggregate functions such as AVG(), MAX(), MIN() and SUM().

For example: (Without GROUP BY - All rows)

```
SELECT MAX(price), MIN(price), AVG(price), STD(price), SUM(quantity) FROM products;
+------------+------------+------------+------------+---------------+
| MAX(price) | MIN(price) | AVG(price) | STD(price) | SUM(quantity) |
+------------+------------+------------+------------+---------------+
|       5.00 |       0.48 |   2.136923 |   1.422211 |        195000 |
+------------+------------+------------+------------+---------------+
```

The numbers should be formated as xxx.xx (will be introduced below).

```
SELECT productCode AS Code, MAX(price) AS 'Highest Price', MIN(price)
AS 'Lowest Price'
FROM products
GROUP BY productCode;
+------+---------------+--------------+
| Code | Highest Price | Lowest Price |
+------+---------------+--------------+
| NTB  |          3.30 |         2.00 |
| PAP  |          5.00 |         4.80 |
| PEC  |          0.49 |         0.48 |
| PEN  |          1.25 |         1.23 |
| RUB  |          1.99 |         0.99 |
+------+---------------+--------------+
```

--

CAST(... AS ...) function

The CAST function is used for converting a value from one datatype to another specific datatype, or to format floating-point numbers.

Examples:
Set the variables:

```
set @a=5, @b=3;
```

Then, work it out:

```
select @a+@b;
+-------+
| @a+@b |
+-------+
|     8 |
+-------+

select @a+@b "a+b";
+-------+
| a+b |
+-----+
|   8 |
+-----+

select @a a, @b b, @a+@b "a+b";
+------+------+------+
| a    | b    | a+b  |
+------+------+------+
|    5 |    3 |    8 |
+------+------+------+

select @a a, @b b, @a*@b "a*b";
+------+------+------+
| a    | b    | a*b  |
+------+------+------+
|    5 |    3 |   15 |
+------+------+------+


select @a a, @b b, sqrt(@a)*sqrt(@b) "sqrt(a)*sqrt(b)";
+------+------+-------------------+
| a    | b    | sqrt(a)*sqrt(b)   |
+------+------+-------------------+
|    5 |    3 | 3.872983346207417 |
+------+------+-------------------+

Finally:
SELECT @a a, @b b, CAST(sqrt(@a)*sqrt(@b) AS DECIMAL(7,2)) "sqrt(a)*sqrt(b)";
+------+------+-----------------+
| a    | b    | sqrt(a)*sqrt(b) |
+------+------+-----------------+
|    5 |    3 |            3.87 |
+------+------+-----------------+
-
```

Now, to the table 'products':

```
SELECT productCode,MAX(price),MIN(price),AVG(price),STD(price),SUM(quantity)
FROM products
GROUP BY 1
ORDER BY 6 DESC;
```

productCode	MAX(price)	MIN(price)	AVG(price)	STD(price)	SUM(quantity)
NTB	3.47	2.10	2.705000	0.539096	152500
PEC	0.51	0.50	0.505000	0.005000	18000
PEN	1.31	1.23	1.283333	0.037712	14950
RUB	2.09	1.04	1.565000	0.525000	2700
PAP	5.25	5.04	5.145000	0.105000	2300
DIG	449.99	449.99	449.990000	0.000000	4

--

Exercise. Use CAST(..) to format the floating-point numbers above and to get the results as below:

Note: Pay attention to the order.

productCode	MAX(price)	MIN(price)	Average	Std. Dev	SUM(quantity)
NTB	3.30	2.00	2.58	0.51	157000
PEC	0.49	0.48	0.49	0.01	18000
PEN	1.25	1.23	1.24	0.01	15000
RUB	1.99	0.99	1.49	0.50	2700
PAP	5.00	4.80	4.90	0.10	2300

--

HAVING clause

HAVING is similar to WHERE, but it can operate on the GROUP BY aggregate functions; whereas WHERE operates only on columns.

```
SELECT productCode AS 'Product Code', COUNT(*) AS 'Count',
CAST(AVG(price) AS DECIMAL(7,2)) AS 'Average'
FROM products
GROUP BY productCode
HAVING Count >=3;
```

Product Code	Count	Average
NTB	4	2.58
PEN	3	1.24

Cannot use WHERE count >= 3.

--

Exercise. Use another criteria than 'count' to get the output below:

```
SELECT productCode AS 'Product Code', COUNT(*) AS 'Count', CAST(AVG(price) AS
DECIMAL(7,2)) AS 'Average'
FROM products  GROUP BY productCode HAVING ........ >= ... ;
```

```
+--------------+-------+---------+
| Product Code | Count | Average |
+--------------+-------+---------+
| PAP          |     2 |    4.90 |
+--------------+-------+---------+
--

WITH ROLLUP

The WITH ROLLUP clause shows the summary of group values.

For example:
SELECT productCode, MAX(price), MIN(price),
CAST(AVG(price) AS DECIMAL(7,2)) AS 'Average',
SUM(quantity)
FROM products
GROUP BY 1
WITH ROLLUP;
+-------------+------------+------------+---------+---------------+
| productCode | MAX(price) | MIN(price) | Average | SUM(quantity) |
+-------------+------------+------------+---------+---------------+
| NTB         |       3.30 |       2.00 |    2.58 |        157000 |
| PAP         |       5.00 |       4.80 |    4.90 |          2300 |
| PEC         |       0.49 |       0.48 |    0.49 |         18000 |
| PEN         |       1.25 |       1.23 |    1.24 |         15000 |
| RUB         |       1.99 |       0.99 |    1.49 |          2700 |
| NULL        |       5.00 |       0.48 |    2.14 |        195000 |
+-------------+------------+------------+---------+---------------+
The aggregate functions were applied to all groups.
--
The line starting with 'NULL' is inappropriate?
Then, use COALESCE.

--
COALESCE(...., ...)

The SQL Coalesce (and ISNULL) function(s) is (are) used to handle NULL values.
During the expression evaluation process the NULL values are replaced with the
user-defined value.
COALESCE evaluates the arguments in order and always returns first non-null value
from the defined argument list.
Note: In SQL NULL is a STATE, not a value.

Exmaple:
SELECT COALESCE(NULL, NULL, 'Frist non-null argument', NULL, 'Second non-null
argument');
Check it on your terminal.
```

```
Back to the table 'products':
SELECT COALESCE(productCode, 'Total') Code, MAX(price), MIN(price),
CAST(AVG(price) AS DECIMAL(7,2)) AS 'Average',
SUM(quantity)
FROM products
GROUP BY productCode
WITH ROLLUP;
+-------+------------+------------+---------+---------------+
| Code  | MAX(price) | MIN(price) | Average | SUM(quantity) |
+-------+------------+------------+---------+---------------+
| NTB   |       3.30 |       2.00 |    2.58 |        157000 |
| PAP   |       5.00 |       4.80 |    4.90 |          2300 |
| PEC   |       0.49 |       0.48 |    0.49 |         18000 |
| PEN   |       1.25 |       1.23 |    1.24 |         15000 |
| RUB   |       1.99 |       0.99 |    1.49 |          2700 |
| Total |       5.00 |       0.48 |    2.14 |        195000 |
+-------+------------+------------+---------+---------------+
Important note:
DON'T use ordinals in the GROUP BY attribute, when using COALESCE on that
attribute.
--

UPDATE - Modifying Data

To modify existing data, use UPDATE ... SET command, with the following syntax:
UPDATE tableName SET columnName = {value|NULL|DEFAULT}, ... WHERE criteria

For example, increase the  prices by 5% for all products
UPDATE products SET price = price * 1.05;
--
Check the changes:
select * from products;
+-----------+-------------+----------------------+----------+-------+
| productID | productCode | name                 | quantity | price |
+-----------+-------------+----------------------+----------+-------+
|      1001 | PEN         | Pen Red              |     5000 |  1.29 |
|      1002 | PEN         | Pen Blue             |     8000 |  1.31 |
|      1003 | PEN         | Pen Black            |     2000 |  1.31 |
|      1004 | PEC         | Pencil 2B            |    10000 |  0.50 |
|      1005 | PEC         | Pencil 2H            |     8000 |  0.51 |
|      1006 | NTB         | Notebook A4L90p      |     9000 |  3.47 |
|      1007 | NTB         | Notebook B5L 60p     |     8000 |  2.94 |
|      1008 | NTB         | Notebook A5L 50p     |    70000 |  2.31 |
|      1009 | NTB         | Notebook B6L 40p     |    70000 |  2.10 |
|      1010 | PAP         | Paper A4 white 80/500 |     1100 |  5.25 |
|      1011 | PAP         | Paper A4 eco 80/500  |     1200 |  5.04 |
|      1012 | RUB         | Rubber white Soft    |     1200 |  2.09 |
|      1013 | RUB         | Rubber color Medium  |     1500 |  1.04 |
+-----------+-------------+----------------------+----------+-------+
```

Exercise.
Insert a new record into products:
1. The productID should be the set by the system.
2. The productCode is 'DIG' for 'digital'.
3. The name is 'iPad mini-4-Wi-Fi-128'.
4. Quantity in stock: 4.
5. Unit price: 449.00

Note: Don't forget, that the first attribute productID is been insert by the system.
It's necessary to declare the attribute name for the values to be inserted (2-5).
--
Result:
```
select * from products;
+-----------+-------------+----------------------+----------+--------+
| productID | productCode | name                 | quantity | price  |
+-----------+-------------+----------------------+----------+--------+
|      1001 | PEN         | Pen Red              |     5000 |   1.29 |
..
|      1013 | RUB         | Rubber color Medium  |     1500 |   1.04 |
|      1014 | DIG         | iPad mini-4-Wi-Fi-128 |        4 | 449.99 |
+-----------+-------------+----------------------+----------+--------+
```
--
```
UPDATE products SET quantity = quantity - 100 WHERE name = 'Pen Red';
```
--
Check the update:
```
select * from products where name = 'Pen Red';
+-----------+-------------+---------+----------+-------+
| productID | productCode | name    | quantity | price |
+-----------+-------------+---------+----------+-------+
|      1001 | PEN         | Pen Red |     4900 |  1.29 |
+-----------+-------------+---------+----------+-------+
```
--
Update more then one attribute value:
```
UPDATE products SET quantity = quantity + 50, price = 1.23 WHERE name = 'Pen Red';
select * from products where name = 'Pen Red';
+-----------+-------------+---------+----------+-------+
| productID | productCode | name    | quantity | price |
+-----------+-------------+---------+----------+-------+
|      1001 | PEN         | Pen Red |     4950 |  1.23 |
+-----------+-------------+---------+----------+-------+
```
--
Exercise: Sales.
1. 100 A4-paper reams were sold: update the quantity (based on productID).
2. 1500 notebooks each, of the sizes A5L and B6L, were sold to schools and highschools.
   Update quantities based on name and the sizes mentioned above.
Notes: Use only 1 SQL statement for each update; 1 ream = 500 sheets.
Proceed with the above mentioned updates.

Results:
select * from products;

productID	productCode	name	quantity	price
...				
1006	NTB	Notebook A4L90p	7500	3.47
1007	NTB	Notebook B5L 60p	8000	2.94
1008	NTB	Notebook A5L 50p	68500	2.31
1009	NTB	Notebook B6L 40p	68500	2.10
...				

--
Check the example below:
SELECT productID, name, price * quantity AS Balance FROM products ORDER BY productCode, 3 DESC;

productID	name	Balance
1014	iPad mini-4-Wi-Fi-128	1799.96
1008	Notebook A5L 50p	158235.00
1009	Notebook B6L 40p	143850.00
1006	Notebook A4L90p	26025.00
1007	Notebook B5L 60p	23520.00
1011	Paper A4 eco 80/500	6048.00
1010	Paper A4 white 80/500	5775.00
1004	Pencil 2B	5000.00
1005	Pencil 2H	4080.00
1002	Pen Blue	10480.00
1001	Pen Red	6088.50
1003	Pen Black	2620.00
1012	Rubber white Soft	2508.00
1013	Rubber color Medium	1560.00

--
Exercise.
After all those updates, check the balance of the stocks for the accounting...
Calculate the value of all yours items grouped on 'productCode'.

```
SELECT COALESCE(productCode, .........) Code, .....(....... * ........)
AS Balance
FROM products
GROUP BY productCode
WITH ROLLUP;
```

```
+-------+-----------+
| Code  | Balance   |
+-------+-----------+
| DIG   |   1799.96 |
| NTB   | 351630.00 |
| PAP   |  11823.00 |
| PEC   |   9080.00 |
| PEN   |  19188.50 |
| RUB   |   4068.00 |
| Total | 397589.46 |
+-------+-----------+
Note: (Again..) In this case use 'GROUP BY productCode', the attribute name,
instead 'GROUP BY 1', an ordinal.
--

Exporting a database data to a text file.

The 'mysqldump' console utility is used to export databases to SQL text files.
For Linux (use the 'root' account):
$ sudo su
[sudo] password for ck:
# mysqldump -u root -p db3 > db3-dump.sql
Enter password:
# ls db3-dump.sql
db3-dump.sql
#
Done!

Windows (use forward-slash (instead of back-slash) as directory separator):
SELECT * FROM products INTO OUTFILE 'd:/myProject/db3-dmup.csv'
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\r\n';
-
OS X/Linux (in this case no test was performed):
SELECT * FROM products INTO OUTFILE '/home/ck/db3-dump.csv' COLUMNS TERMINATED BY
',';
--

Importing a database (or single tables) from a text file.

For Linux.
Importing the whole databse:
Create the bew databse:
MariaDB [..]> create database db4 charset latin1 collate latin1_swedish_ci;
-
Now use the terminal (command-line):
Check the directory to get the right file, namelly 'db3-dump.sql'.
Use /home/<your-user> (e.g. '~')
Look for the dump file 'db3-dump.sql':
~$ ll db3-dump.sql
-rw-r--r-- 1 ck ck 2683 May  3 13:21 db3-dump.sql
~$ sudo [/usr/bin/]mysql -u root -p  db4 '/home/ck/db3-dump.sql'
--
```

```
Import comma-separated-values-file ('.csv') into a table.

Insert/import tuples (in this case just 1 tuple) from a csv-file into the table
'products':
File: db3-hp-calc-input.csv
\N,DIG,HP Prime Calculator,20,159.00
--
Example: Import a tuple into 'products'.
LOAD DATA LOCAL INFILE '/home/ck/db3-hp-calc-input.csv' INTO TABLE products COLUMNS
TERMINATED BY ',';
--
Result:

select * from products;
+-----------+-------------+----------------------+----------+--------+
| productID | productCode | name                 | quantity | price  |
+-----------+-------------+----------------------+----------+--------+
|      1001 | PEN         | Pen Red              |     4950 |   1.23 |
..
|      1014 | DIG         | iPad mini-4-Wi-Fi-128 |        4 | 449.99 |
|      1015 | DIG         | HP Prime Calculator  |       20 | 159.00 |
+-----------+-------------+----------------------+----------+--------+
--
Exercise:
Create a tab-separeted-value-file (db3-input.tsv) with following lines (no empty-
lines are allowed):
\N   PEC   Pencil  3B   500   0.52
\N   PEC   Pencil  4B   200   0.62
\N   PEC   Pencil  5B   100   0.73
\N   PEC   Pencil  6B   500   0.47

Important note:
'\N' stands for the entry of the 'productID' done by the system, due to the
AUTO_INCREMENT option.

Example using tsv-file:
LOAD DATA LOCAL INFILE '/home/ck/db3-input.tsv' INTO TABLE products COLUMNS
TERMINATED BY '\t';
Result:
select * from products;
+-----------+-------------+----------------------+----------+--------+
| productID | productCode | name                 | quantity | price  |
+-----------+-------------+----------------------+----------+--------+
|      1001 | PEN         | Pen Red              |     4950 |   1.23 |
...
|      1015 | DIG         | HP Prime Calculator  |       20 | 159.00 |
|      1016 | PEC         | Pencil 3B            |      500 |   0.52 |
|      1017 | PEC         | Pencil 4B            |      200 |   0.62 |
|      1018 | PEC         | Pencil 5B            |      100 |   0.73 |
+-----------+-------------+----------------------+----------+--------+

--------------------------------------------------------------------------------
```