**Lab 3: JOIN Operations, Group by & Having Clause and Sub queries**

**Inner Join**

Suppose you want to get:

* The productCodeand productName from the products table.
* The textDescription of product lines from the productlines table*.*

To do this, you need to select data from both tables by matching rows based on values in the productline column using the INNER JOIN clause as follows:

SELECT

    productCode,

    productName,

    textDescription

FROM

    products t1

INNER JOIN productlines t2

    ON t1.productline = t2.productline;

Because the joined columns of both tables have the same name  productline, you can use the USING syntax:

SELECT

    productCode,

    productName,

    textDescription

FROM

    products

INNER JOIN productlines USING (productline);

The query returns the same result set. However, the USING syntax is much shorter and cleaner.

MySQL INNER JOIN with GROUP BY clause example

SELECT

    t1.orderNumber,

    t1.status,

    SUM(quantityOrdered \* priceEach) total

FROM

    orders t1

INNER JOIN orderdetails t2

    ON t1.orderNumber = t2.orderNumber

GROUP BY orderNumber;

This query returns order number, order status and total sales from the  orders  and  orderdetails tables using the INNER JOIN clause with the GROUP BY clause.

Similarly, the following query uses the INNER JOIN with the USING syntax:

SELECT

    orderNumber,

    status,

    SUM(quantityOrdered \* priceEach) total

FROM

    orders

INNER JOIN orderdetails USING (orderNumber)

GROUP BY orderNumber;

This query uses two INNER JOIN clauses to join three tables: orders, orderdetails, and products:

SELECT

    orderNumber,

    orderDate,

    orderLineNumber,

    productName,

    quantityOrdered,

    priceEach

FROM

    orders

INNER JOIN

    orderdetails USING (orderNumber)

INNER JOIN

    products USING (productCode)

ORDER BY

    orderNumber,

    orderLineNumber;

This example uses three INNER JOIN clauses to query data from the four tables as;

SELECT

    orderNumber,

    orderDate,

    customerName,

    orderLineNumber,

    productName,

    quantityOrdered,

    priceEach

FROM

    orders

INNER JOIN orderdetails

    USING (orderNumber)

INNER JOIN products

    USING (productCode)

INNER JOIN customers

    USING (customerNumber)

ORDER BY

    orderNumber,

    orderLineNumber;

## MySQL INNER JOIN using other operators

So far, you have seen that the join condition used the equal operator (=) for matching rows.

In addition to the equal operator (=), you can use other operators such as greater than ( >), less than ( <), and not-equal ( <>) operator to form the join condition.

The following query uses a less-than ( <) join to find sales price of the product whose code is S10\_1678 that is less than the manufacturer’s suggested retail price (MSRP) for that product.

SELECT

    orderNumber,

    productName,

    msrp,

    priceEach

FROM

    products p

INNER JOIN orderdetails o

   ON p.productcode = o.productcode

      AND p.msrp > o.priceEach

WHERE

    p.productcode = 'S10\_1678';

## MySQL LEFT JOIN

Using MySQL LEFT JOIN clause to join two tables

This query uses the LEFT JOIN clause to find all customers and their orders:

SELECT

    customers.customerNumber,

    customerName,

    orderNumber,

    status

FROM

    customers

LEFT JOIN orders ON

    orders.customerNumber = customers.customerNumber;

**Alternatively**

SELECT

    c.customerNumber,

    customerName,

    orderNumber,

    status

FROM

    customers c

LEFT JOIN orders o

    ON c.customerNumber = o.customerNumber;

In this example:

* The customers is the left table and orders is the right table.
* The LEFT JOIN clause returns all customers including the customers who have no order. If a customer has no order, the values in the column orderNumber and status are NULL.

Because both table customers and orders have the same column name ( customerNumber) in the join condition with the equal operator, you can use the USING syntax as follows:

SELECT

    customerNumber,

    customerName,

    orderNumber,

    status

FROM

    customers

LEFT JOIN orders USING (customerNumber);

Using MySQL LEFT JOIN clause to find unmatched rows

The LEFT JOIN clause is very useful when you want to find rows in a table that doesn’t have a matching row from another table.

The following example uses the LEFT JOIN to find customers who have no order:

SELECT

    c.customerNumber,

    c.customerName,

    o.orderNumber,

    o.status

FROM

    customers c

LEFT JOIN orders o

    ON c.customerNumber = o.customerNumber

WHERE

    orderNumber IS NULL;

Using MySQL LEFT JOIN to join three tables

This example uses two LEFT JOIN clauses to join the three tables: employees, customers, and payments

SELECT

    lastName,

    firstName,

    customerName,

    checkNumber,

    amount

FROM

    employees

LEFT JOIN customers ON

    employeeNumber = salesRepEmployeeNumber

LEFT JOIN payments ON

    payments.customerNumber = customers.customerNumber

ORDER BY

    customerName,

    checkNumber;

* The first LEFT JOIN returns all employees and customers who represented each employee or NULL if the employee does not in charge of any customer.
* The second LEFT JOIN returns payments of each customer represented by an employee or NULL if the customer has no payment.

**Using Where Clause**

SELECT

    o.orderNumber,

    customerNumber,

    productCode

FROM

    orders o

LEFT JOIN orderDetails

    USING (orderNumber)

WHERE

    orderNumber = 10123;

## MySQL RIGHT JOIN clause

MySQL RIGHT JOIN is similar to [LEFT JOIN](http://www.mysqltutorial.org/mysql-left-join.aspx), except that the treatment of the joined tables is reversed.

SELECT

    employeeNumber,

    customerNumber

FROM

    customers

RIGHT JOIN employees

    ON salesRepEmployeeNumber = employeeNumber

ORDER BY

    employeeNumber;

* The RIGHT JOIN returns all rows from the table employees whether rows in the table employees have matching values in the column salesRepEmployeeNumber of the table customers.
* If a row from the table employees has no matching row from the table customers , the RIGHT JOIN uses NULL for the customerNumber column.

Using MySQL RIGHT JOIN to find unmatching rows

The following statement uses the RIGHT JOIN clause to find employees who do not in charge of any customers:

SELECT

    employeeNumber,

    customerNumber

FROM

    customers

RIGHT JOIN employees ON

    salesRepEmployeeNumber = employeeNumber

WHERE customerNumber is NULL

ORDER BY employeeNumber;

## MySQL self join examples

MySQL self join using INNER JOIN clause

To get the whole organization structure, you can join the employees table to itself using the employeeNumber and reportsTo columns. The table employees has two roles: one is the *Manager* and the other is *Direct Reports.*

SELECT

    CONCAT(m.lastName, ', ', m.firstName) AS Manager,

    CONCAT(e.lastName, ', ', e.firstName) AS 'Direct report'

FROM

    employees e

INNER JOIN employees m ON

    m.employeeNumber = e.reportsTo

ORDER BY

    Manager;

MySQL self join using LEFT JOIN clause

The President is the employee who does not have any manager or the value in the reportsTo column is NULL .

The following statement uses the LEFT JOIN clause instead of INNER JOIN to include the President:

SELECT

    IFNULL(CONCAT(m.lastname, ', ', m.firstname),

            'Top Manager') AS 'Manager',

    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'

FROM

    employees e

LEFT JOIN employees m ON

    m.employeeNumber = e.reportsto

ORDER BY

    manager DESC;

Using MySQL self join to compare successive rows

By using the MySQL self join, you can display a list of customers who locate in the same city by joining the customers table to itself.

SELECT

    c1.city,

    c1.customerName,

    c2.customerName

FROM

    customers c1

INNER JOIN customers c2 ON

    c1.city = c2.city

    AND c1.customername > c2.customerName

ORDER BY

    c1.city;

In this example, the table customers is joined to itself using the following join conditions:

* c1.city = c2.city  makes sure that both customers have the same city.
* c.customerName > c2.customerName ensures that no same customer is included.

CREATE TABLE stores (

    id INT PRIMARY KEY AUTO\_INCREMENT,

    store\_name VARCHAR(100)

);

INSERT INTO stores(store\_name)

VALUES('North'),

      ('South');

[CREATE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/create-table.html) [TABLE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/create-table.html) sales ( product\_id [varchar](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/string-types.html)(20), store\_id [INT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/numeric-types.html), quantity [DECIMAL](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/numeric-types.html)(13 , 2 ) [NOT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/logical-operators.html#operator_not) NULL, sales\_date [DATE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/date-and-time-types.html) [NOT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/logical-operators.html#operator_not) NULL, PRIMARY KEY (product\_id , store\_id), FOREIGN KEY (product\_id) REFERENCES products (productCode) ON [DELETE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/delete.html) CASCADE ON [UPDATE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/update.html) CASCADE, FOREIGN KEY (store\_id) REFERENCES stores (id) ON [DELETE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/delete.html) CASCADE ON [UPDATE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/update.html) CASCADE )

[INSERT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/insert.html) INTO sales(store\_id,product\_id,quantity,sales\_date) [VALUES](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/miscellaneous-functions.html#function_values)(1,'S10\_1678',20,'2017-01-02'), (1,'S10\_1949',15,'2017-01-05'), (1,'S10\_2016',25,'2017-01-05'), (2,'S10\_1678',30,'2017-01-02'), (2,'S10\_1949',35,'2017-01-05')

MySQL CROSS JOIN example

This statement returns total sales for each store and product, you calculate the sales and [group](http://www.mysqltutorial.org/mysql-group-by.aspx)them by store and product as follows:

[SELECT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) store\_name, productName, [SUM](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/group-by-functions.html#function_sum)(quantity \* buyPrice) AS revenue FROM sales INNER JOIN products ON products.productCode = sales.product\_id INNER JOIN stores ON stores.id = sales.store\_id GROUP BY store\_name , productName;

If you want to know also which store had no sales of a specific product, use the CROSS JOIN clause to get the combination of all stores and products:

[SELECT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) store\_name, productName FROM stores AS a CROSS JOIN products AS b;

MySQL GROUP BY

SELECT

    status

FROM

    orders

GROUP BY status;

Using MySQL GROUP BY with aggregate functions

The [aggregate functions](http://www.mysqltutorial.org/mysql-aggregate-functions.aspx) allow you to perform the calculation of a set of rows and return a single value. The GROUP BY clause is often used with an aggregate function to perform calculation and return a single value for each subgroup.

For example, if you want to know the number of orders in each status, you can use the COUNT function with the GROUP BY clause as follows:

SELECT

    status, COUNT(\*)

FROM

    orders

GROUP BY status;

To get the total amount of all orders by status, you [join](http://www.mysqltutorial.org/mysql-inner-join.aspx)the orders table with the orderdetails table and use the SUM function to calculate the total amount. See the following query:

SELECT

    status, SUM(quantityOrdered \* priceEach) AS amount

FROM

    orders

        INNER JOIN

    orderdetails USING (orderNumber)

GROUP BY status;

Similarly, the following query returns the order numbers and the total amount of each order.

SELECT

    orderNumber,

    SUM(quantityOrdered \* priceEach) AS total

FROM

    orderdetails

GROUP BY orderNumber;

Using MySQL GROUP BY with HAVING clause example

To filter the groups returned by GROUP BY clause, you use a  [HAVING](http://www.mysqltutorial.org/mysql-having.aspx) clause. The following query uses the HAVING clause to select the total sales of the years after 2003.

SELECT

    YEAR(orderDate) AS year,

    SUM(quantityOrdered \* priceEach) AS total

FROM

    orders

        INNER JOIN

    orderdetails USING (orderNumber)

WHERE

    status = 'Shipped'

GROUP BY year

HAVING year > 2003;

You can use GROUP BY clause to get order numbers, the number of items sold per order, and total sales for each:

SELECT

    ordernumber,

    SUM(quantityOrdered) AS itemsCount,

    SUM(priceeach\*quantityOrdered) AS total

FROM

    orderdetails

GROUP BY ordernumber;

Now, you can find which order has total sales greater than 1000 by using the HAVING clause as follows:

SELECT

    ordernumber,

    SUM(quantityOrdered) AS itemsCount,

    SUM(priceeach\*quantityOrdered) AS total

FROM

    orderdetails

GROUP BY ordernumber

HAVING total > 1000;

You can construct a complex condition in the HAVING clause using logical operators such as [OR](http://www.mysqltutorial.org/mysql-or/) and [AND](http://www.mysqltutorial.org/mysql-and/). Suppose you want to find which orders have total sales greater than 1000 and contain more than 600 items, you can use the following query:

SELECT

    ordernumber,

    SUM(quantityOrdered) AS itemsCount,

    SUM(priceeach\*quantityOrdered) AS total

FROM

    orderdetails

GROUP BY ordernumber

HAVING total > 1000 AND itemsCount > 600;

# MySQL Subquery

A MySQL subquery is a query nested within another query such as [SELECT](http://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [INSERT](http://www.mysqltutorial.org/mysql-insert-statement.aspx), [UPDATE](http://www.mysqltutorial.org/mysql-update-data.aspx)or [DELETE](http://www.mysqltutorial.org/mysql-delete-statement.aspx). In addition, a subquery can be nested inside another subquery.

A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

The following query returns employees who work in offices located in the USA.

SELECT

    lastName, firstName

FROM

    employees

WHERE

    officeCode IN (SELECT

            officeCode

        FROM

            offices

        WHERE

            country = 'USA');

MySQL subquery with comparison operators

You can use comparison operators e.g., =, >, < to compare a single value returned by the subquery with the expression in the [WHERE](http://www.mysqltutorial.org/mysql-where/) clause.

For example, the following query returns the customer who has the maximum payment.

SELECT

    customerNumber,

    checkNumber,

    amount

FROM

    payments

WHERE

    amount = (SELECT MAX(amount) FROM payments);

In addition to the equality operator, you can use other comparison operators such as greater than ( >), less than( <).

For example, you can find customers whose payments are greater than the average payment using a subquery:

SELECT

    customerNumber,

    checkNumber,

    amount

FROM

    payments

WHERE

    amount > (SELECT

            AVG(amount)

        FROM

            payments);

You can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows:

SELECT

    customerName

FROM

    customers

WHERE

    customerNumber NOT IN (SELECT DISTINCT

            customerNumber

        FROM

            orders);

**Subquery in the From Clause**

You can use subquery in the from clause as well.

select lastName, firstName

from ( select \* from employees where reportsTo is NULL) as temp\_table

The above query will select lastname and firstname of employee from employees table who reports to no one. Here the subquery in from clause create a temporary table temp\_table from where lastname and firstname are selected.