Tutorial - Week 8

## Objectives:

* OUTER JOIN
* UNARY JOINS
* VIEWS
* RELATIONAL DIVIDES

1. Find the number of units sold of each item

**SELECT** item.Name, sum(saleitem.Quantity) as UnitsSold

**FROM** saleitem **NATURAL JOIN** item

**GROUP BY** item.Name

**ORDER BY** item.Name;

However, this query **does not** return the fact that the Horse Saddle has not been sold!

# OUTER JOINS

To retrieve all items even if they have not been sold you may need to use an OUTER JOIN.

MySQL Server supports LEFT OUTER JOIN and RIGHT OUTER JOIN. Syntactically while RIGHT JOIN and LEFT JOIN work it is best to use the OUTER word to indicate your intent with the SQL statement.

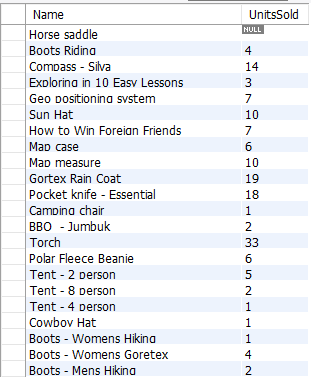
You use a LEFT OUTER JOIN or a RIGHT OUTER JOIN dependent on where the 'Null' column table resides in your query.

The following query provides a dummy column in the saleitem table.

**SELECT** item.Name, SUM(saleitem.quantity) as UnitsSold

**FROM** saleitem **RIGHT OUTER JOIN** item **ON** saleitem.itemID = item.itemID **GROUP BY** item.name

**ORDER BY** item.name;



1. Find any suppliers that deliver no more than two unique items. List the suppliers in alphabetical order



1. Find the names of suppliers that have never delivered a Compass



# Unary Joins

The query below is a self join to the employee table. You will notice that we have created an alias for the employee table as emp for employees and boss for their manager. The bossid in the employee table becomes the employeeid in the boss table. This is also known as an *UNARY* join

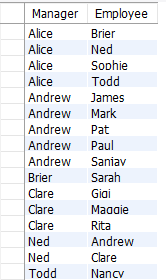
下面的查询是到employee表的一个自连接。您将注意到，我们为employee表创建了一个别名，为employees创建了emp，为manager创建了boss。employee表中的bossid变成boss表中的employeeid。这也称为UNARY连接

1. List the first names of each manager and their employees. Order the result by manager first name, then employee first name.

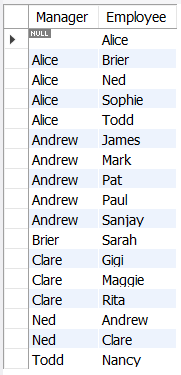
**SELECT** boss.FirstName AS Manager, emp.FirstName AS employee

**FROM** employee AS emp **INNER JOIN** employee AS boss ON emp.BossID = boss.employeeID

**ORDER BY** boss.FirstName, emp.FirstName;

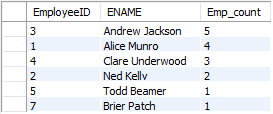


1. Now modify this query to use an outer join to list Alice as an employee



1. Type the query to count the number of direct employees of each manager, List the employeeID, Manager Name and number of employees.

Your result set should look similar to this:



# Views

Views are a table whose rows are not explicitly stored in the database but are returned as needed from a stored view definition.

Consider the following view

**CREATE VIEW** vdepartment\_Wages **AS**

**SELECT** departmentID, Name, SUM(Salary) as TotalWages

**FROM** department **NATURAL JOIN** employee

**GROUP BY** departmentID, Name

**ORDER BY** departmentID;

This creates a view called vdepartment\_wages. I can use this view like any table in my schema.

### SELECT \*

**FROM** vdepartment\_Wages

**WHERE** TotalWages > 150000;

A screenshot of a cell phone  Description automatically generated

However, what is really going on is the following query:

### SELECT \*

**FROM**

(**SELECT** departmentID, Name, SUM(Salary) as TotalWages

**FROM** department **NATURAL JOIN** employee

**GROUP BY** departmentID, Name

**ORDER BY** departmentID) as vdepartment\_Wages

**WHERE** TotalWages > 150000;

The SELECT statement for the view is being used in the FROM clause of SQL. This is here to explain how the view is used by retrieving the stored code from CREATE VIEW statement. This is still considered as a view and is known as an INLINE VIEW.

At any time the SQL that makes up the view definition can be queried from the Data Dictionary:

**SELECT** table\_name, view\_definition

**FROM** Information\_schema.views

-- **WHERE** Table\_SCHEMA= 'labs2018' – remove comment for BYOD devices

;

1. List the employees in the Accounting department and the difference between their salaries and the average salary of the department

First create a view of the all department Names and average Salary called VdepartmentSalary Now use the view vdepartmentSalary in the query to answer the question

A screenshot of a cell phone  Description automatically generated

1. List each employee’s salary, the average salary within that person’s department, and the difference between the employees’ salaries and the average salary of the department

*HINT: Use the vdepartmentSalary view …*



1. How many supplier – department pairs exist in which the supplier delivers at least one item of type E to the department?

First create the view:

**CREATE VIEW** vSupplierdepartment **AS**

(**SELECT** DISTINCT SupplierID, departmentID

**FROM** delivery **NATURAL JOIN** deliveryitem **NATURAL JOIN** item

**WHERE** item.Type = 'E' );

Then count the rows in the view:

**SELECT** count(\*)

**FROM** vSupplierdepartment;



# Using Views

1. Create a VIEW of department names and total number of sales for each department.
2. Use the view created in Task 10 to identify department names with more than 5 sales. List the department and number of sales.
3. Create a view to list the department id, department name, maximum salary, average salary, minimum salary, total salary and number of staff in each department.
4. Use the view created in Task 12 to find the lowest salary in the department with the highest headcount.

# Relational Divides

## Relational Divides - How they work

1. List the departments that have at least one sale of all the items delivered to them Attempt 1 uses NOT EXISTS to find the departments that have sold all itemids that have been delivered.

列出至少有一次销售所有交付给他们的物品的部门，尝试1使用NOT EXISTS来查找已经销售所有交付的物品id的部门。

**SELECT** DISTINCT departmentID **FROM** deliveryitem del1 **WHERE NOT EXISTS**

(**SELECT** \*

**FROM** deliveryitem del2

**WHERE** del2.departmentID = del1.departmentID AND **NOT EXISTS**

(**SELECT** \*

**FROM** saleitem **NATURAL JOIN** sale

**WHERE** del2.itemID = saleitem.itemID

**AND** del1.departmentID = sale.departmentID));

Firstly NOT EXISTS means if there are no rows in the result set that evaluates to TRUE, if there are rows it evaluates to FALSE. Therefore if the departmentid from deliveritem del1 matches a row in deliveritem del2 a value is in the set and there not exists evaluates to FALSE.

It helps to look at the result pairs side by side. First the deliveryitem deapartmentids and itemids:

首先，NOT EXISTS表示如果结果集中没有计算为TRUE的行，如果有计算为FALSE的行。因此，如果来自deliveritem del1的departmentid与deliveritem del2中的一行匹配，则值在集合中且不存在，则计算为FALSE。并排查看结果会有所帮助。首先是交付项目部门和项目id:

**SELECT** distinct(departmentid), itemid

**FROM** deliveryitem

**ORDER BY** departmentid, itemid;

The result set is (departmentid, itemid)

{(2,3), (2,5), (2,6), (2,9), (2,12), (2,14), (2,17),

(3,1), (3,8), (3,12), (3,14),(3,17), (3,18), (3,22),(3,23),(3,24),(3,25),

(4,2), (4,3), (4,12), (4,14), (4,15), (4,16), (4,17),

(5,12), (5,14), (5,17),

(6,3), (6,5), (6,6), (6,9), (6,10), (6,11), (6,12), (6,13), (6,14), (6,17),

(7,5), (7,9), (7,14), (7,19), (7,19), (7,20), (7,21) }

This automatically tells us that departmentids 1,8,9,10 & 11 will not be in our result set because they have not received a delivery

We then look at the departments that have sold items:

**SELECT** distinct(departmentid), saleitem.itemid

**FROM** saleitem **INNER JOIN** sale **on** sale.saleid = saleitem.saleid **ORDER BY** departmentid, itemid;

This result set is (departmentid, itemid):

{(2,1), (2,3), (2,5), (2,6), (2,9), (2,12), (2,14), (2,17),

(3,8), (3,12), (3,14), (3,18),(3,22), (3,23), (3,24), (3,25),

(4,3), (4,12), (4,14), (4,15), (4,16), (4,17),

(5, 12), (5,14), (5,17) ,

(6,3), (6,6), (6,9), (6,10), (6,11), (6,12), (6, 14), (6,17),

(7,14), (7,19), (7,20), (7,21)}

Consider the result sets side by side - each row is the set for the departmentid, itemid in deliveryitem and sale/saleitem tables:

|  |  |  |
| --- | --- | --- |
| deliveryitem (departmnetid, itemid) |  | sale/saleitem (departmentid, itemid) |
| (2,3), (2,5), (2,6), (2,9), (2,12), (2,14), (2,17) |  | (2,1), (**2,3**), (**2,5**), (**2,6**), (**2,9**), (**2,12**), (**2,14**), (**2,17**), |
| (**3,1**), (3,8), (3,12), (3,14),(3,17), (3,18),  (3,22),(3,23),(3,24),(3,25) |  | (3,8), (3,12), (3,14), (3,18),(3,22), (3,23), (3,24),  (3,25), |
| (**4,2**), (4,3), (4,12), (4,14), (4,15), (4,16), (4,17) |  | (4,3), (4,12), (4,14), (4,15), (4,16), (4,17), |
| (5,12), (5,14), (5,17) |  | (**5, 12**), (**5,14**), (**5,17**) |
| (6,3), (6,5), (6,6), (6,9), (6,10), (6,11), (6,12),  (**6,13**), (6,14), (6,17) |  | (6,3), (6,6), (6,9), (6,10), (6,11), (6,12), (6, 14),  (6,17) |
| (**7,5**), (**7,9**), (7,14), (7,19), (7,19), (7,20), (7,21) |  | (7,14), (7,19), (7,20), (7,21) |

*Table 1: The result set in deliveryitem must be found for the department result set for sale/saleitem.This is true for departments 2 & 5 only (note the deliveryitem itemID is a subset of the sale/saleitem result set for department id 2, as item id 1 was in stock and sold but has not been delivered)*

Consider the department 3 (row 3) result sets.

The SELECT clause is selecting department 3 from the deliveryitem (del1) table it then joins to the deliveryitem (del2) in the first subquery and finds departmentID 3, itemID 1 the result set (3,1). As the record is found the NOT EXISTS condition is evaluated to FALSE as a record exists.

Now we need to find a FALSE record for the sale, however result set (3,1) does NOT EXIST in the sale, saleitem subquery - and evaluates to TRUE. Because it is an AND condition both subqueries must be true TRUE != FALSE the result set is not returned.

This process repeats for every result set returned by the queries. Only when FALSE = FALSE (rows DO EXIST) will a result set be returned. This is because of the join to the table deliveryitem (aliased as del1) in both subqueries del1.departmentid=del2.departmentid in subquery 1 and del1.departmentid=sale.departmentid in subquery 2.

1. Find the items (itemID) sold by ALL departments located on the second floor

**SELECT** saleitem.itemID

**FROM** saleitem NATURAL JOIN sale NATURAL JOIN department

**WHERE** department.Floor = 2

**GROUP BY** saleitem.itemID

**HAVING** count(DISTINCT department.departmentID) = (**SELECT** count(DISTINCT departmentID) **FROM** department

**WHERE** department.Floor = 2

)

**ORDER BY** saleitem.itemID;

And using a different method

**SELECT** DISTINCT itemID

**FROM** item

### WHERE NOT EXISTS

(**SELECT** \*

**FROM** department

**WHERE** department.Floor = 2 AND **NOT EXISTS**

(**SELECT** \*

**FROM** saleitem **NATURAL JOIN** sale

**WHERE** saleitem.itemID = item.itemID

**AND** sale.departmentID = department.departmentID

)

)

A screenshot of a cell phone  Description automatically generated**ORDER BY** itemID;

1. List the department names that have not recorded a sale for all the items of type N

**SELECT** department.Name

**FROM** department

**WHERE** departmentID **NOT IN** (**SELECT** departmentID **FROM** department **WHERE NOT EXISTS**

(**SELECT** \*

**FROM** item

**WHERE** item.Type = 'N' AND **NOT EXISTS**

(**SELECT** \*

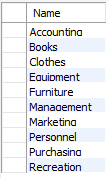
**FROM** sale **NATURAL JOIN** saleitem

**WHERE** sale.departmentID = department.departmentID

**AND** saleitem.itemID = item.itemID)

)

)

**ORDER BY** department.Name;

1. Type a relational divide query that lists the suppliers that delivery only items sold by the Books department

A screenshot of a cell phone  Description automatically generated

**SELECT** supplier.Name

**FROM** supplier

**WHERE** SupplierID IN

(**SELECT** SupplierID **FROM** delivery) AND **NOT EXISTS**

(**SELECT** \*

**FROM** deliveryitem **NATURAL JOIN** delivery

**WHERE** delivery.SupplierID = supplier.SupplierID

**AND** itemID **NOT IN**

(**SELECT** itemID

**FROM** saleitem **NATURAL JOIN** sale **NATURAL JOIN** department

**WHERE** department.Name = 'Books'));

As you will see there are many different queries that can achieve the same result set.

**End of Week 9 Lab**

# Diagram Description automatically generatedAppendix: The New department Store ER Physical Model