INFO20003 Week 7 Lab

Objectives:

- Practice further joins involving three and four tables
- Understand CASE statements and the UNION clause
- Develop complex SQL queries using derived tables and views
- Create and understand relational divides using EXISTS and NOT EXISTS

Section 1: More SQL

More about joins

Consider a query to list the item name (from the *item* table), quantity (from the *saleitem* table), sale id (from the *saleitem* and *sale* tables) and sale date (from the *sale* table) from the department store schema. (You can obtain an ER model of this schema from the Labs page on LMS.)

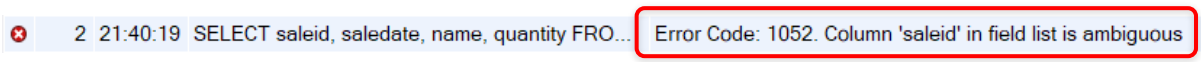
Try this first using NATURAL JOIN:

```
SELECT saleid, saledate, name, quantity
FROM item NATURAL JOIN saleitem NATURAL JOIN sale
ORDER BY saleid;
```

Now try it with INNER JOIN:

```
SELECT saleid, saledate, name, quantity
FROM item
  INNER JOIN saleitem ON item.itemid = saleitem.itemid
  INNER JOIN sale ON saleitem.saleid = sale.saleid
ORDER BY saleid;
```

The query using the inner join will not run. MySQL Server returns an error:



This is because MySQL Server does not know whether to use the sale.saleid or saleitem.saleid column in the SELECT and ORDER BY clauses. This can be fixed:

```
SELECT sale.saleid, saledate, name, quantity
FROM item
  INNER JOIN saleitem ON item.itemid = saleitem.itemid
  INNER JOIN sale ON saleitem.saleid = sale.saleid
ORDER BY sale.saleid;
```

Once we use the fully-qualified name for one table column, we might decide to use the fully-qualified name for them all in order to improve readability:

```
SELECT sale.saleid, sale.saledate, item.name, saleitem.quantity
FROM item
  INNER JOIN saleitem ON item.itemid = saleitem.itemid
  INNER JOIN sale ON saleitem.saleid = sale.saleid
ORDER BY sale.saleid;
```

◆ **Task 1.1** List the names, salaries and department names of the employees who manage more than two employees.

```
SELECT employee.FirstName, employee.LastName, employee.Salary,
  department.Name
FROM employee NATURAL JOIN department
WHERE employeeID IN
     (SELECT BossID
      FROM employee
      GROUP BY BossID
      HAVING COUNT(*) > 2);
```

	FirstName	LastName	Salary	Name
▶	Alice	Munro	125000.00	Management
	Andrew	Jackson	55000.00	Marketing
	Clare	Underwood	52000.00	Marketing

Remember that HAVING is used to apply a condition over an aggregate (COUNT, SUM, AVG, MAX, MIN) function.

◆ **Task 1.2** List the suppliers that have delivered at least 10 distinct items. List the supplier name and id.

```
SELECT supplier.SupplierID, supplier.Name
FROM deliveryitem
  INNER JOIN delivery ON deliveryitem.DeliveryID = delivery.DeliveryID
  INNER JOIN supplier ON delivery.SupplierID = supplier.SupplierID
GROUP BY supplier.SupplierID
HAVING COUNT(DISTINCT deliveryitem.ItemID) >= 10;
```

	SupplierID	Name
▶	102	Nepalese Corp.
	105	All Points_ Inc.

◆ **Task 1.3** List the item names that are delivered by ‘Nepalese Corp.’ and are sold in the Navigation department.

```
SELECT DISTINCT item.Name
FROM item
WHERE ItemID IN
    (SELECT ItemID
     FROM deliveryitem NATURAL JOIN delivery NATURAL JOIN supplier
     WHERE supplier.Name = 'Nepalese Corp.')
  AND ItemID IN
    (SELECT ItemID
     FROM saleitem NATURAL JOIN sale NATURAL JOIN department
     WHERE department.Name = 'Navigation');
```

	Name
▶	Compass - Silva
	Geo positioning system
	How to Win Foreign Friends
	Map case
	Map measure
	Gortex Rain Coat
	Pocket knife - Essential
	Torch

◆ **Task 1.4** Type a query that finds the item ids of the items sold on the second floor.

Your result should look like this:

	ItemID
	8
	12
	14
	17
	18
	19
	20
	21
	22
	23
	24
	25

◆ **Task 1.5** Find the id and name of suppliers that deliver both compasses and an item other than compasses.

Attempt 1:

```
SELECT DISTINCT delivery.supplierID, supplier.Name
FROM supplier
  INNER JOIN delivery ON supplier.supplierID = delivery.supplierID
  INNER JOIN deliveryitem ON delivery.deliveryID = deliveryitem.deliveryID
  INNER JOIN item ON deliveryitem.itemID = item.itemID
WHERE item.Name NOT LIKE 'Compass%'
AND delivery.supplierID IN
     (SELECT supplierID
      FROM delivery NATURAL JOIN item NATURAL JOIN deliveryitem
      WHERE item.Name LIKE 'Compass%')
ORDER BY delivery.supplierID;
```

Attempt 2:

```
SELECT DISTINCT delivery.SupplierID, supplier.Name
FROM supplier
  INNER JOIN delivery ON supplier.supplierID = delivery.supplierID
  INNER JOIN deliveryitem ON delivery.deliveryID = deliveryitem.deliveryID
  INNER JOIN item ON deliveryitem.itemID = item.itemID
WHERE delivery.SupplierID IN
     (SELECT SupplierID
      FROM delivery NATURAL JOIN deliveryitem NATURAL JOIN item
      WHERE item.Name LIKE 'Compass%')
GROUP BY delivery.SupplierID
HAVING COUNT(DISTINCT item.Name) > 1
ORDER BY delivery.SupplierID;
```

SupplierID	Name
101	Global Books & Maps
102	Nepalese Corp.
103	All Sports Manufacturing
105	All Points_ Inc.

Attempt 1 uses the approach to find those suppliers that supply things other than compasses (outer query) and also supply compasses (subquery).

Attempt 2 uses a more generalizable approach. The generalizable approach is better as it allows queries such as “Find suppliers that deliver two items other than compasses” – change the > 1 to > 2 in the HAVING clause in Attempt 2 to do this. (Attempt 2 uses DISTINCT to handle multiple deliveries of compasses from the same supplier.)

◆ **Task 1.6** Type a query that for each item, gives its type, the departments that sell the item, and the floor location of these departments.
Hint: You will need to join four tables (Item, SaleItem, Sale and Department) and make sure each ambiguous column in your SELECT clause is fully qualified (employee.EmployeeName instead of EmployeeName).

Your result should look like this (see next page):

Name	Type	DepartmentID	Floor
BBQ - Jumbuk	F	4	3
Boots - Mens Hiking	C	3	2
Boots - Womens Goretex	C	3	2
Boots - Womens Hiking	C	3	2
Boots Riding	C	2	1
Camping chair	F	4	3
Compass - Silva	N	2	1
Compass - Silva	N	4	3
Compass - Silva	N	6	1
Cowboy Hat	C	3	2
Exploring in 10 Easy Lessons	B	2	1
Geo positioning system	N	2	1
Geo positioning system	N	6	1
Gortex Rain Coat	C	2	1
Gortex Rain Coat	C	3	2
Gortex Rain Coat	C	4	3
Gortex Rain Coat	C	5	4
Gortex Rain Coat	C	6	1
How to Win Foreign Friends	B	2	1
How to Win Foreign Friends	B	6	1
Map case	E	6	1
Map measure	N	6	1
Pocket knife - Essential	E	2	1
Pocket knife - Essential	E	3	2
Pocket knife - Essential	E	4	3
Pocket knife - Essential	E	5	4
Pocket knife - Essential	E	6	1
Pocket knife - Essential	E	7	2
Polar Fleece Beanie	C	3	2
Sun Hat	C	3	2
Tent - 2 person	F	7	2
Tent - 4 person	F	7	2
Tent - 8 person	F	7	2
Torch	E	2	1
Torch	E	3	2
Torch	E	4	3
Torch	E	5	4
Torch	E	6	1

The following query uses NOT IN (rather than IN) to solve the problem. Each step is worked through and explained.

◆ **Task 1.7** Find the items that are not sold by departments on the second floor but are sold on other floors within the store.

When solving problems like this, work in steps:

1. Identify the items sold on the second floor,
2. Then find the items that are not in the result from part 1.

```
SELECT DISTINCT ItemID
FROM saleitem
WHERE ItemID NOT IN
   (SELECT DISTINCT ItemID
    FROM sale NATURAL JOIN saleitem NATURAL JOIN department
    WHERE department.Floor = 2)
ORDER BY ItemID;
```

ItemID	Floor
1	1
3	1
3	3
5	1
6	1
9	1
10	1
11	1
15	3
16	3

The inner query identifies the items that ARE sold on the second floor. The outer query then finds all items which have been sold (they are present in the sale table) but are not in the inner query.

Compare that to this query:

```
SELECT DISTINCT ItemID, department.Floor
FROM sale NATURAL JOIN saleitem NATURAL JOIN department
WHERE department.Floor <> 2
ORDER BY ItemID;
```

This query finds items sold on floors other than the second floor – but this includes items which have been sold on the second floor as well!

ItemID	Floor
1	1
3	3
3	1
5	1
6	1
9	1
10	1
11	1
12	3
12	4
12	1
14	3
14	4
14	1
15	3
16	3
17	3
17	4
17	1

The additional items are 12, 14 and 17. A final query will confirm which floors these items are sold on:

```
SELECT DISTINCT ItemID, department.Floor
FROM sale NATURAL JOIN saleitem NATURAL JOIN department
WHERE saleitem.ItemID IN (12,14,17)
ORDER BY Itemid, department.Floor;
```

ItemID	Floor
12	1
12	2
12	3
12	4
14	1
14	2
14	3
14	4
17	1
17	2
17	3
17	4

It is always good to check your logic with some *ad hoc* queries on the dataset. The more familiar you are with the dataset, the clearer your thinking will be about how to approach solving the task. (Of course, in a timed assessment scenario, such as an exam, the dataset is not given – so you also need to practise writing queries without the dataset. The Golf case study SQL practice questions on LMS can help you prepare for this.)

◆ **Task 1.8** What is the average quantity of each item of type 'N' delivered by each company that delivers it? Be sure to list the item name in your answer.

```
SELECT delivery.SupplierID, supplier.Name, item.Name,
  FORMAT(AVG(deliveryitem.Quantity), 2) AS AvgDelQty
FROM supplier
  INNER JOIN delivery ON supplier.supplierID = delivery.supplierID
  INNER JOIN deliveryitem ON delivery.deliveryID = deliveryitem.deliveryID
  INNER JOIN item ON deliveryitem.itemID = item.itemID
WHERE item.Type = 'N'
GROUP BY delivery.SupplierID, supplier.Name, item.Name;
```

Your result should look like this:

SupplierID	Name	Name	AvgDelQty
101	Global Books & Maps	Compass - Silva	4.67
101	Global Books & Maps	Geo positioning system	3.00
101	Global Books & Maps	Map measure	10.00
102	Nepalese Corp.	Compass - Silva	3.00
102	Nepalese Corp.	Geo positioning system	4.00
102	Nepalese Corp.	Map measure	10.00
103	All Sports Manufacturing	Compass - Silva	8.00
103	All Sports Manufacturing	Geo positioning system	1.50
103	All Sports Manufacturing	Map measure	10.00
105	All Points_ Inc.	Compass - Silva	1.00

◆ **Task 1.9** List suppliers that have delivered more than 40 items of types 'C' and 'N'.

Approach the problem in steps:

1. First, find the items that are of type C and N:

```
SELECT item.Name, item.Type
FROM item
WHERE item.Type IN ('C','N')
ORDER BY item.Name;
```

2. Then, find out how many of each of those items have been delivered:

```
SELECT item.Name, SUM(deliveryitem.Quantity)
FROM deliveryitem INNER JOIN item
  ON deliveryitem.ItemID = item.ItemID
WHERE item.Type IN ('C','N')
GROUP BY item.Name;
```

3. Ensure that the quantity delivered is greater than 40:

```
SELECT item.Name, SUM(deliveryitem.Quantity)
FROM deliveryitem INNER JOIN item
  ON deliveryitem.ItemID = item.ItemID
WHERE item.Type IN ('C','N')
GROUP BY item.Name
HAVING SUM(deliveryitem.Quantity) > 40;
```

4. Display the supplier names and ids, changing the GROUP BY clause to match:

```
SELECT supplier.SupplierID, supplier.Name
FROM supplier
  INNER JOIN delivery ON supplier.supplierID = delivery.supplierID
  INNER JOIN deliveryitem ON delivery.deliveryID = deliveryitem.deliveryID
  INNER JOIN item ON deliveryitem.itemID = item.itemID
WHERE item.Type IN ('C','N')
GROUP BY supplier.SupplierID
HAVING SUM(deliveryitem.Quantity) > 40;
```

This statement uses an IN condition, but it could also be written with an OR clause:

```
SELECT supplier.SupplierID, supplier.Name
FROM supplier
  INNER JOIN delivery ON supplier.supplierID = delivery.supplierID
  INNER JOIN deliveryitem ON delivery.deliveryID = deliveryitem.deliveryID
  INNER JOIN item ON deliveryitem.itemID = item.itemID
WHERE item.Type = 'C' OR item.Type = 'N'
GROUP BY supplier.SupplierID
HAVING SUM(deliveryitem.Quantity) > 40;
```

The result is the same either way:

	SupplierID	Name
▶	101	Global Books & Maps
	105	All Points_ Inc.

◆ **Task 1.10** Find the ids of items sold by at least two departments on the second floor.

```
SELECT DISTINCT ItemID
FROM sale NATURAL JOIN saleitem NATURAL JOIN department
WHERE department.Floor = 2
GROUP BY ItemID
HAVING COUNT(DISTINCT DepartmentID) >= 2;
```

Alternatively:

```
SELECT DISTINCT ItemID
FROM sale NATURAL JOIN saleitem
WHERE DepartmentID IN
    (SELECT DepartmentID
     FROM department
     WHERE department.Floor = 2)
GROUP BY ItemID
HAVING COUNT(DISTINCT DepartmentID) >= 2;
```

There are often multiple ways of achieving the same result set.

	ItemID
▶	14

◆ **Task 1.11** Name the items which have been delivered by exactly one supplier.

```
SELECT item.Name, COUNT(DISTINCT SupplierID) AS SupplierCount
FROM deliveryitem NATURAL JOIN delivery NATURAL JOIN item
GROUP BY item.Name
HAVING COUNT(DISTINCT SupplierID) = 1
ORDER BY item.Name;
```

	Name	SupplierCount
▶	BBQ - Jumbuk	1
	Boots - Mens Hiking	1
	Boots - Womens Goretex	1
	Boots - Womens Hiking	1
	Boots Riding	1
	Camping chair	1
	Cowboy Hat	1
	Horse saddle	1
	Polar Fleece Beanie	1
	Sun Hat	1
	Tent - 2 person	1
	Tent - 4 person	1
	Tent - 8 person	1

UNION

The SQL standard includes the set operators from relational algebra – union (UNION), intersection (INTERSECT) and set difference (EXCEPT). Of these three, MySQL only supports UNION.

Consider the following task:

◆ **Task 1.12** Name the items that are delivered by 'Nepalese Corp.' **or** sold in the Navigation department.

This query can be written as two subqueries:

```
SELECT item.Name
FROM item
WHERE ItemID IN
     (SELECT ItemID
      FROM deliveryitem NATURAL JOIN delivery NATURAL JOIN supplier
      WHERE supplier.Name = 'Nepalese Corp.')
  OR ItemID IN
     (SELECT ItemID
      FROM saleitem NATURAL JOIN sale NATURAL JOIN department
      WHERE department.Name = 'Navigation');
```

The same query can be written using a UNION:

```
SELECT item.Name
FROM deliveryitem NATURAL JOIN delivery NATURAL JOIN supplier
  INNER JOIN item ON deliveryitem.ItemID = item.ItemID
WHERE supplier.Name = 'Nepalese Corp.'
UNION
SELECT item.Name
FROM saleitem NATURAL JOIN sale NATURAL JOIN department
  INNER JOIN item ON saleitem.ItemID = item.ItemID
WHERE department.Name = 'Navigation';
```

Note that the SELECT statements have been simplified when UNION is used. The SELECT uses the item name directly, not the item id.

The result set of both sides of the UNION must have the same number of columns in the same order with the same data types. In this case we have only one column: the *name* column of the item table.

	Name
▶	Compass - Silva
	Exploring in 10 Easy Lessons
	Geo positioning system
	How to Win Foreign Friends
	Map case
	Map measure
	Gortex Rain Coat
	Pocket knife - Steadfast
	Pocket knife - Essential
	Camping chair
	BBQ - Jumbuk
	Torch
	Tent - 2 person
	Tent - 8 person
	Tent - 4 person

UNION ALL may be used as an alternative to UNION when duplicate rows are to be preserved.

CASE

The CASE statement, comparable to a series of *if … elseif … elseif … else* statements in a programming language like Python or C, returns the value associated with the first condition that evaluates to true.

◆ **Task 1.13** Employees who earn more than $80,000 a year are classified as pay grade A. If they earn less than or equal to $45,000 they are classified as pay grade C. All other employees are in pay grade B. List each employee's name and pay grade. Order the results by pay grade, then last name.

```
SELECT FirstName, LastName,
  CASE WHEN Salary > 80000 THEN 'A'
       WHEN Salary <= 45000 THEN 'C'
       ELSE 'B'
  END AS PayGrade
FROM employee
ORDER BY PayGrade, LastName;
```

FirstName	LastName	PayGrade
Sarah	Fergusson	A
Ned	Kelly	A
Alice	Munro	A
Todd	Beamer	B
Nancy	Cartwright	B
Andrew	Jackson	B
Sophie	Monk	B
Gigi	Montez	B
Brier	Patch	B
Maggie	Smith	B
Clare	Underwood	B
Pat	Clarkson	C
Paul	Innit	C
James	Mason	C
Sanjay	Patel	C
Rita	Skeeter	C
Mark	Zhang	C

◆ **Task 1.14** The company's building is going to be renovated. The renovations will include all departments on floors 3 and 4 of the building plus the 'Recreation' department. If additional funding is approved, the rest of floor 2 may be included in the renovation as well. For all employees, print their name and a text value 'Yes', 'Maybe' or 'No' that states whether they will be affected by the renovations. Order by last name.

Your result set should look like this (see next page):

	FirstName	LastName	Affected
▶	Todd	Beamer	No
	Nancy	Cartwright	No
	Pat	Clarkson	Yes
	Sarah	Fergusson	No
	Paul	Innit	Yes
	Andrew	Jackson	No
	Ned	Kelly	No
	James	Mason	Yes
	Sophie	Monk	No
	Gigi	Montez	Maybe
	Alice	Munro	No
	Brier	Patch	No
	Sanjay	Patel	No
	Rita	Skeeter	No
	Maggie	Smith	Maybe
	Clare	Underwood	No
	Mark	Zhang	Yes

Section 2: Derived tables and views

Derived tables

When writing more complex queries, we might wish to query the result set of an SQL statement as if it was its own table. To accomplish this, we can place subqueries in the FROM clause of our query – these subqueries are known as *derived tables*.

Consider the following task:

◆ **Task 2.1** For each department, find the average quantity of items sold in each sale.

We need to add up the quantity of items in each sale using the SUM function, then take the average of those sums using AVG. You might begin to write a query like this:

```
SELECT departmentID, AVG(SUM(Quantity))
FROM ...
```

However, it is not possible to nest aggregate functions in this way. Instead, we need to split the problem into two queries.

The first query finds the quantity of items sold in each sale, also returning the department in which the sale occurred:

```
SELECT departmentID, SUM(Quantity) AS total_qty
FROM sale NATURAL JOIN saleitem
GROUP BY SaleID;
```

We then need to query the output of this query! We make it into a derived table and group its rows by department id:

```
SELECT departmentID, AVG(total_qty)
FROM (SELECT departmentID, SUM(Quantity) AS total_qty
      FROM sale NATURAL JOIN saleitem
      GROUP BY SaleID) AS SaleQuantities
GROUP BY departmentID;
```

departmentID	AVG(total_qty)
2	3.8333
3	4.8750
4	3.1667
5	1.5000
6	4.3846
7	2.0000

Notice that we gave the derived table an alias, *SaleQuantities*. It is compulsory to give an alias to every derived table.

Views

A view can be thought of as a virtual table whose rows are not explicitly stored in the database but are returned as needed. A view stores a SELECT query which is executed every time the view is queried. In this way, a view behaves like a derived table that is "bookmarked" for repeated use.

Views are commonly used in large enterprises for access control. Suppose that the company has decided that employee salary amounts are confidential, and moreover, employees in the Management department should have all their details hidden from other staff.

In this situation, you would use the database's access control features to hide the *employee* table from all but a set of privileged users. You would then create a view into the *employee* table that leaves out the *salary* column as well as all rows for employees in Management:

```
CREATE VIEW employee_public AS
SELECT employeeID, FirstName, LastName, departmentID,
  bossID, DateOfBirth
FROM employee
WHERE departmentID <> (SELECT departmentID
                       FROM department
                       WHERE Name = 'Management');
```

Unprivileged users would be forced to use this view to gain information about employees. The *employee_public* view can be queried just like a normal table:

```
SELECT *
FROM employee_public
WHERE LastName LIKE 'M%';
```

employeeID	FirstName	LastName	departmentID	bossID	DateOfBirth
9	Sophie	Monk	10	1	1986-12-15
12	Gigi	Montez	3	4	1992-03-20
15	James	Mason	4	3	1995-07-30

Notice that the *salary* column is not present, and Alice Munro is not included in the results.

Please do **not** use views when writing SQL queries for assessment in this subject.

Section 3: Relational divides using EXISTS and NOT EXISTS

Sometimes, we need to find rows in one table that are associated with **all** rows in the other table as part of a many-to-many relationship. This can be done using GROUP BY and HAVING, checking whether the count of rows is equal to the maximum number of available rows. However, *relational divides* are another approach that can be used.

◆ **Task 3.1** List the names of departments that have recorded a sale for *all* the items of type 'N'.

Using a relational divide:

```
SELECT name
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)
    );
```

	name
▶	Navigation

Using GROUP BY and HAVING:

```
SELECT department.name
FROM department NATURAL JOIN sale NATURAL JOIN saleitem
  INNER JOIN item ON item.itemID = saleitem.itemID
WHERE item.Type = 'N'
GROUP BY department.name
HAVING COUNT(DISTINCT item.itemID) = (SELECT COUNT(*)
                                     FROM item
                                     WHERE Type = 'N');
```

	name
▶	Navigation

The solution with GROUP BY and HAVING is just as correct as the relational divide. However, you need to ensure that the subquery in the HAVING clause matches the conditions in the query itself.

◆ **Task 3.2** Who are the suppliers (give their ids and names) that deliver all the items of type ‘N’?

```
SELECT supplierID, supplier.Name
FROM supplier
WHERE NOT EXISTS
   (SELECT *
    FROM item
    WHERE item.Type = 'N'
      AND NOT EXISTS
         (SELECT *
          FROM delivery NATURAL JOIN deliveryitem
          WHERE delivery.supplierID = supplier.supplierID
            AND deliveryitem.itemID = item.itemID)
    );
```

	supplierID	Name
▶	101	Global Books & Maps
	102	Nepalese Corp.
	103	All Sports Manufacturing

◆ **Task 3.3** List the departments that have recorded at least one sale for all the items delivered to them.

Attempt 1 (relational divide):

```
SELECT DISTINCT departmentID
FROM deliveryitem AS del1
WHERE NOT EXISTS
   (SELECT *
    FROM deliveryitem AS 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)
    );
```

Attempt 2:

```
SELECT DISTINCT departmentID
FROM deliveryitem AS del1
WHERE NOT EXISTS
   (SELECT *
    FROM deliveryitem AS del2
    WHERE del2.departmentID = del1.departmentID
      AND itemID NOT IN
         (SELECT itemID
          FROM sale NATURAL JOIN saleitem
          WHERE sale.departmentID = del1.departmentID)
    );
```

Attempt 3 (using GROUP BY and HAVING):

```
SELECT del1.departmentid
FROM deliveryitem AS del1
  INNER JOIN saleitem ON del1.itemid = saleitem.itemid
  INNER JOIN sale ON saleitem.saleid = sale.saleid
    AND del1.departmentid = sale.departmentid
GROUP BY del1.departmentid
HAVING COUNT(DISTINCT saleitem.itemid) =
    (SELECT COUNT(DISTINCT itemid)
     FROM deliveryitem AS del2
     WHERE del2.departmentid = del1.departmentid);
```

Attempt 4 (using outer joins):

```
SELECT deliveryitem.departmentID
FROM deliveryitem
  LEFT JOIN (saleitem NATURAL JOIN sale)
    ON deliveryitem.itemID = saleitem.itemID
    AND deliveryitem.departmentID = sale.departmentID
GROUP BY deliveryitem.departmentID
HAVING COUNT(*) = COUNT(saleitem.itemID);
```

	departmentID
▶	2
	5

Attempt 4 uses the fact that COUNT(*) gives the number of rows for each departmentID, while COUNT(saleitem.itemID) counts non-NULL values. If there are any rows in deliveryitem for which the LEFT JOIN cannot find a match (i.e. there is no sale of that item in that department), the two counts will be different, and that department will be excluded.

◆ **Task 3.4** Type a query that lists the suppliers that deliver only items sold by the Books department (in other words, suppliers for which every delivery is of an item or items sold by the Books department).

	Name
▶	Sweatshops Unlimited
	Sao Paulo Manufacturing

End of Week 7 Lab