INFO90002 Week 3 Lab

Objectives:

- Review the Case Study & your conceptual model
- Use MySQL modelling tool to build a logical model
-

Section 1. Data Modelling

1.1. Review the case study and your pen and paper conceptual data model from Lab

Case Study: The Department Store

This database is the central component of an information system used to manage a department store that specialises in camping and hiking equipment. The store has several departments. For each department we must record its name and unique department id, phone number, and which floor it is on. Each department has several employees working for it. Each department has a manager. A manager can manage one or more departments.

About each employee we record their first name, last name, a unique employee id, their annual salary, which department they work for and which other employee is their boss. The General Manager of the Department Store has no boss.

The items that the store sells each have a name and id, a type, a colour and the retail price. Whenever a department sells items to customers we record which item was sold, how many were sold, which department sold it. Each sale may contain one or more items Each sale is unique to each department within the store.

Items are delivered to the store by suppliers. Each delivery from a supplier contains one or more items delivered to one or more departments within the store and the wholesale price of each item. For each supplier we record a unique supplier id, name and contact phone number.

1.2 Use MySQL Workbench modelling tool to construct a logical data model identifying primary and foreign keys and the required attributes for each entity.

Section 2 SQL

Using MySQL Workbench, connect to your database MySQL Server

2.1 Show the metadata about the Department table

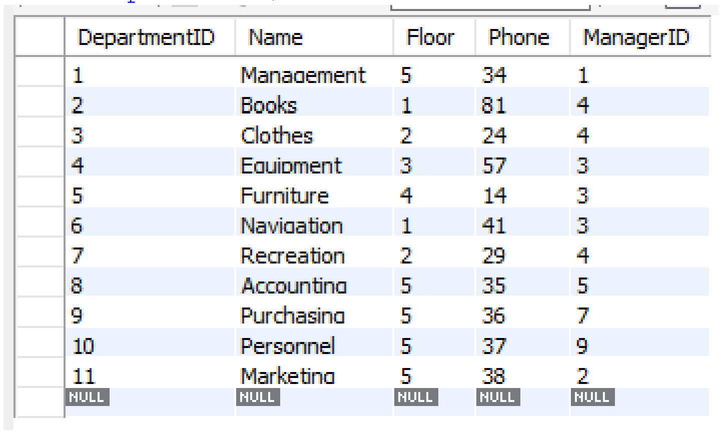
```
DESC Department;
```

The structure of a SELECT statement is below:

```
SELECT (select list)
FROM (from list)
WHERE (filtering list) -- optional
GROUP BY (grouping list) -- optional
HAVING (group qualifications) -- optional
```

To select all rows and all columns from a table we use the SQL shorthand "*" to select from the Department table we enter the following SQL

```
SELECT *
FROM Department;
```



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
2	Books	1	81	4
3	Clothes	2	24	4
4	Equipment	3	57	3
5	Furniture	4	14	3
6	Navigation	1	41	3
7	Recreation	2	29	4
8	Accounting	5	35	5
9	Purchasing	5	36	7
10	Personnel	5	37	9
11	Marketing	5	38	2
NULL	NULL	NULL	NULL	NULL

2.2 Type the SQL query to select all rows and columns from the Employee table in the HR database

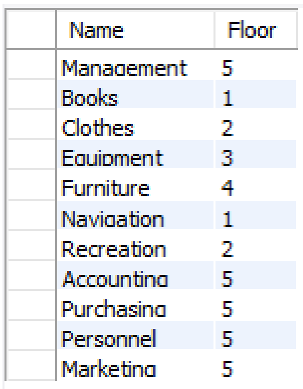
```
SELECT *
FROM Employee;
```

Filtering Results

To select only some columns or we need to specify the columns of the table in the query. To select columns we separate each column name with a “,”. If you describe the Department table we can see attributes Name and Floor.

To select only these two columns in the Department table, the SQL would look like this

```
SELECT Name, Floor
FROM Department;
```



Name	Floor
Management	5
Books	1
Clothes	2
Equipment	3
Furniture	4
Navigation	1
Recreation	2
Accounting	5
Purchasing	5
Personnel	5
Marketing	5

This is known as vertically filtering of the result set

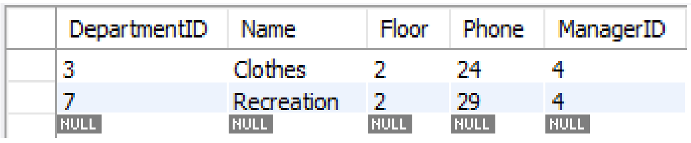
2.3 Type the SQL query to select the first name, last name and departmentid in the Employee table

```
SELECT Firstname, Lastname, departmentid
FROM Department;
```

Until now we have selected all rows in a table. Most times we don't want to retrieve all all rows from a table. This is known as horizontally filtering the result set.

In SQL we do this by using a *condition* on our query. If for example we wished to list all Departments that are listed on the second floor:

```
SELECT *
FROM Department
WHERE floor = 2;
```

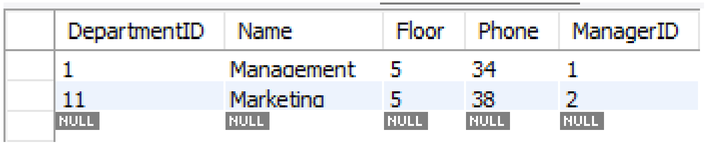


DepartmentID	Name	Floor	Phone	ManagerID
3	Clothes	2	24	4
7	Recreation	2	29	4
NULL	NULL	NULL	NULL	NULL

The SQL keyword 'WHERE' lists any horizontal filter we wish to put on the query.

How do we find out all department names that start with M? To do this we use the wildcard '%' this means any character or characters can be in the position of the %. Therefore we could write a query to display all departments that start with M. We need to use the SQL word LIKE:

```
SELECT *
FROM Department
WHERE Name LIKE 'M%';
```



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
11	Marketing	5	38	2
NULL	NULL	NULL	NULL	NULL

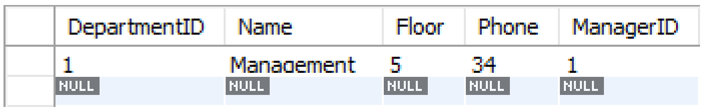
2.4 Type the SQL query to return the first and last names and department id of all employees who earn less than $55000.

```
SELECT firstname, lastname, departmentid
FROM Employee
WHERE Salary < 55000;
```

Sometimes we may need to filter the result set by having more than one condition met. For example we wish to list all the Departments that start with M and the manager ID is 1

```
SELECT *
FROM department
WHERE Name like 'M%'
AND ManagerID = 1;
```

In this case both conditions must be true and only 1 row is returned:

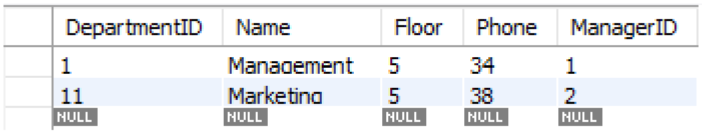


DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
NULL	NULL	NULL	NULL	NULL

However if we change the AND to OR the result set changes. Two rows are returned. When we use an OR condition both conditions are evaluated separately.

```
SELECT *
FROM Department
WHERE Name like 'M%'
OR ManagerID = 1;
```

The query lists all departments starting with M. Also the query lists all departments where the ManagerID is equal to 1.



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
11	Marketing	5	38	2
NULL	NULL	NULL	NULL	NULL

2.5 Type the SQL query that lists all employees who work in DepartmentID 11 AND who earn greater than 55000.

```
SELECT FirstName, LastName, Salary
FROM Employee
WHERE DepartmentID = 11
AND Salary > 55000;
```

Then change the AND to OR and note the difference in the result set. Notice Clare Underwood's salary.

```
SELECT FirstName, LastName, Salary
FROM Employee
WHERE DepartmentID = 11
OR Salary > 55000;
```

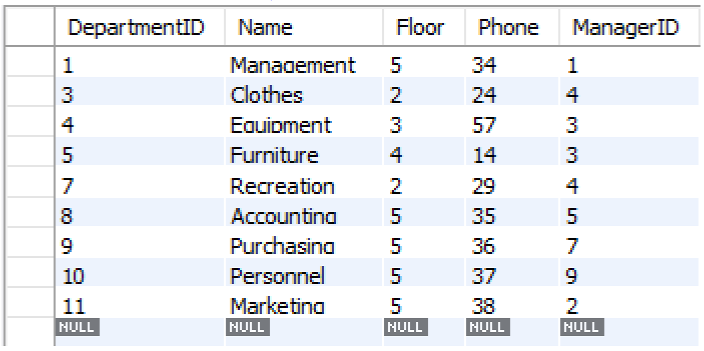
2.6 In MySQL Workbench show the metadata about the Department and Employee tables

```
DESC Department;
DESC Employee;
```

As we've seen SQL supports the use of mathematical operators and functions

To select all departments that are above the first floor we would type

```
SELECT *
FROM Department
WHERE Floor > 1;
```



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
3	Clothes	2	24	4
4	Equipment	3	57	3
5	Furniture	4	14	3
7	Recreation	2	29	4
8	Accounting	5	35	5
9	Purchasing	5	36	7
10	Personnel	5	37	9
11	Marketing	5	38	2
NULL	NULL	NULL	NULL	NULL

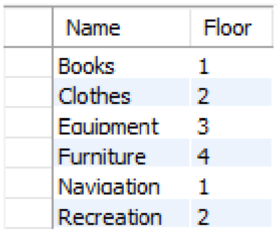
Or find out which departments are not on the fifth floor

```
SELECT Name, Floor
FROM Department
WHERE Floor != 5;
```

OR

```
SELECT Name, Floor
FROM Department
WHERE Floor <> 5;
```

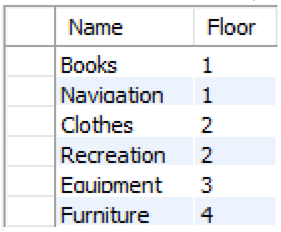
Note that != and <> are both functionally equivalent to 'Not Equal To'



Name	Floor
Books	1
Clothes	2
Equipment	3
Furniture	4
Navigation	1
Recreation	2

We can order the result set by any column (it does not have to be in the SELECT clause). Because SQL does not have to guarantee the result set order to be the same every time, you can force the result set to be ordered identically by using ORDER BY SQL keywords. The ORDER BY forces the result set to be ordered the same way every time the query is executed.

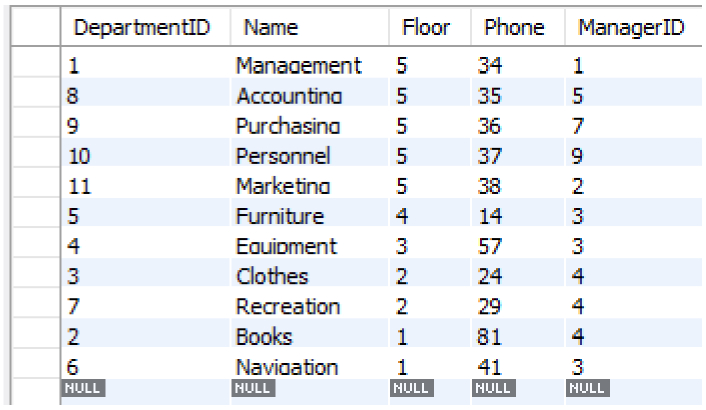
```
SELECT Name, Floor
FROM Department
WHERE Floor != 5
ORDER BY Floor;
```



Name	Floor
Books	1
Navigation	1
Clothes	2
Recreation	2
Equipment	3
Furniture	4

The default sort order is from the smallest value to largest (1-10 or A-Z). You can explicitly state this by typing ASC (short for Ascending order). To sort from largest value to smallest you would enter DESC (short for Descending order).

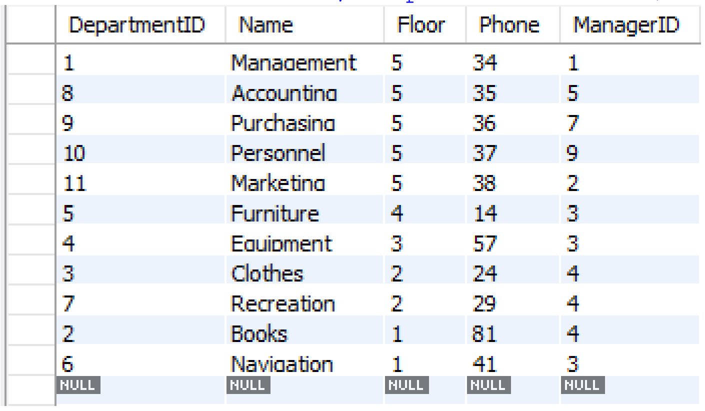
```
SELECT *
FROM Department
ORDER BY Floor DESC;
```



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
8	Accounting	5	35	5
9	Purchasing	5	36	7
10	Personnel	5	37	9
11	Marketing	5	38	2
5	Furniture	4	14	3
4	Equipment	3	57	3
3	Clothes	2	24	4
7	Recreation	2	29	4
2	Books	1	81	4
6	Navigation	1	41	3
NULL	NULL	NULL	NULL	NULL

You can order by more than one column and in different order for each column:

```
SELECT *
FROM Department
ORDER BY Floor DESC, DepartmentID ASC;
```



DepartmentID	Name	Floor	Phone	ManagerID
1	Management	5	34	1
8	Accounting	5	35	5
9	Purchasing	5	36	7
10	Personnel	5	37	9
11	Marketing	5	38	2
5	Furniture	4	14	3
4	Equipment	3	57	3
3	Clothes	2	24	4
7	Recreation	2	29	4
2	Books	1	81	4
6	Navigation	1	41	3
NULL	NULL	NULL	NULL	NULL

2.7 Type the SQL query that returns all employees who earn less than 45,000

```
SELECT firstname, lastname, salary, departmentid
FROM Employee
WHERE Salary < 45000;
```

2.8 Type the SQL query that returns all employees who earn 45,000 or more. Order the query from highest earner to lowest

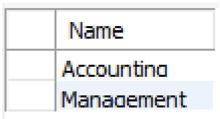
```
SELECT firstname, lastname, salary, departmentid
FROM Employee
WHERE Salary >= 45000
ORDER BY Salary DESC;
```

2.9 Type the SQL query that returns all rows and columns in the employee table. Order the result set by departmentid then sort the employees lastname alphabetically

```
SELECT *
FROM Employee
ORDER BY DepartmentID, LastName;
```

We can limit the number of rows in the result set by using the word LIMIT and specifying an integer after the LIMIT word.

```
SELECT Name
FROM Department
WHERE Floor = 5
ORDER BY Name ASC
LIMIT 2;
```



	Name
	Accounting
	Management

2.10 Type the above query and note the two rows returned. Change the ORDER BY from ASC to DESC and rerun the query. Is the result set different? If so, why are they different?

```
SELECT Name
FROM Department
WHERE Floor = 5
ORDER BY Name DESC
LIMIT 2;
```

Yes, the order is different – we have ordered from Z to A rather than A-Z and only displayed the first two rows.

2.11 Type the SQL query that returns the first name, last name and salary of the five highest salary earners across the Department store.

```
SELECT FirstName, LastName, Salary
FROM Employee
ORDER BY Salary DESC
LIMIT 5;
```

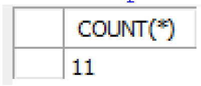
OPERATORS and FUNCTIONS

Functions are mathematical and scientific calculations that are performed automatically by the database engine. There are several function types across all database data types. The most common functions we use are COUNT, MAX, MIN. The full list of Functions you can use in MYSQL are found here in the MySQL reference manual

To find out how many departments there are we can use the COUNT() function. Functions must be given something to act on which can be a column, or all columns using the wild card *

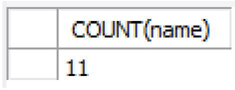
E.G.

```
SELECT COUNT(*)
FROM Department;
```



	COUNT(*)
	11

```
SELECT COUNT(Name)
FROM Department;
```

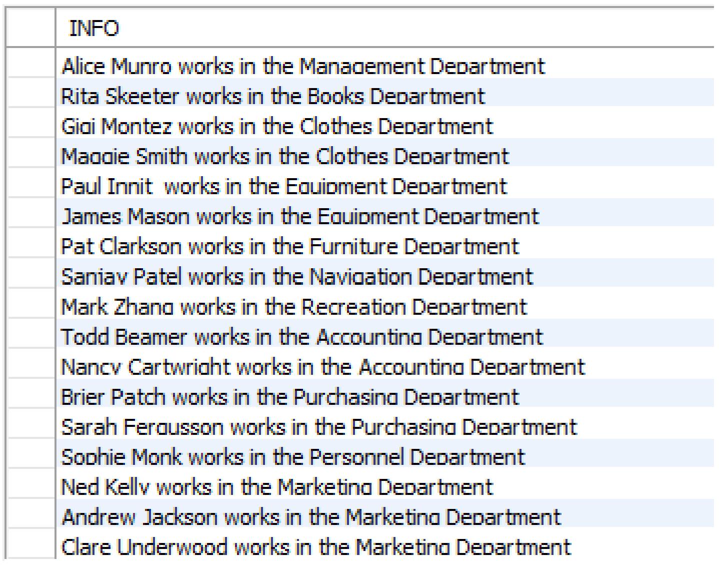


	COUNT(name)
	11

```
SELECT CONCAT(FirstName,' ',LastName , ' works in the ' ,
Department.Name, ' Department') AS INFO
FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

Note we did a ***join*** *between two tables Employee and Department*

More about joining tables in the next lab.



	INFO
	Alice Munro works in the Management Department
	Rita Skeeter works in the Books Department
	Gigi Montez works in the Clothes Department
	Maggie Smith works in the Clothes Department
	Paul Innit works in the Equipment Department
	James Mason works in the Equipment Department
	Pat Clarkson works in the Furniture Department
	Sanjay Patel works in the Navigation Department
	Mark Zhang works in the Recreation Department
	Todd Beamer works in the Accounting Department
	Nancy Cartwright works in the Accounting Department
	Brier Patch works in the Purchasing Department
	Sarah Fergusson works in the Purchasing Department
	Sophie Monk works in the Personnel Department
	Ned Kelly works in the Marketing Department
	Andrew Jackson works in the Marketing Department
	Clare Underwood works in the Marketing Department

2.12 Type the SQL query to find the total number of employees in the employee table

```
SELECT COUNT(*)
```

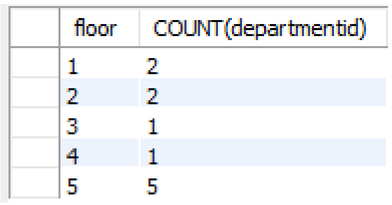
```
FROM Employee;
```

Alternatively:

```
Select count(lastname)
FROM Employee;
```

Sometimes we want to group the function by a particular attribute. For example to find out the number of each departments on each floor of the department store we would type:

```
SELECT floor, count(departmentid)
FROM DEPARTMENT
GROUP BY floor;
```



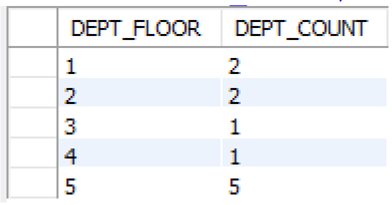
floor	COUNT(departmentid)
1	2
2	2
3	1
4	1
5	5

We use the GROUP BY keyword when aggregate functions are with a column that does not aggregate the rows. We must group by the non aggregated column or columns to ensure the full result set is returned. Thus in the above example we GROUP BY *floor*.

Try this: Remove the GROUP BY keyword and notice the difference in the query output

We can also alias the columns to make the output make more sense to the reader. Then use that alias within the query

```
SELECT floor as DEPT_FLOOR, COUNT(departmentid) AS
DEPT_COUNT
FROM DEPARTMENT
GROUP BY DEPT_FLOOR
ORDER BY DEPT_FLOOR;
```



DEPT_FLOOR	DEPT_COUNT
1	2
2	2
3	1
4	1
5	5

2.13 Type the SQL query to find how many employees work in each department

```
SELECT DepartmentID, Count(EmployeeID)
FROM Employee
GROUP BY DepartmentID;
```

2.14 Type the SQL query to find each department’s average salary?

```
SELECT DepartmentID, AVG(Salary)
FROM Employee
```

```
    GROUP BY DepartmentID;
```

2.15 Type the SQL query that finds what department has the highest salary?

```
    SELECT DepartmentID, MAX(Salary)
    FROM Employee
    Group By DepartmentID
    ORDER BY MAX(Salary) DESC
    LIMIT 1;
```

2.16 Type the SQL query that finds the department with the lowest average salary?

```
    SELECT DepartmentID, MIN(Salary)
    FROM Employee
    Group By DepartmentID
    ORDER BY MIN(Salary)
    LIMIT 1;
```

FORMAT(X,D) and ROUND(X,D) are functions you can use to improve the readability of a query result. Round will round the argument – what is in the parenthesis "X" to D decimal places. Format will format the argument to D decimal places and include commas.

```
    SELECT AVG(Salary) AS AVG_SAL
    FROM Employee;
```

```
60529.411765
```

```
SELECT FORMAT(AVG(SALARY),2) AS AVG_SAL

FROM Employee;
```

```
60,529.41
```

```
SELECT ROUND(AVG(SALARY),2) AS AVG_SAL

FROM Employee;
```

```
60529.41
```

Format and Round while producing the same output are subtly different. Format converts the output into a STRING (hence the 60<comma>529), whereas round keeps the result as a NUMBER

Sub Queries

Sometimes we need to find a value and then use it. For example if we wanted to find out the departmentid of the department which has the lowest salary.

First we would find the lowest salary across all of the department store.

You might be tempted to write a query like this

```
SELECT min(Salary), DepartmentID
FROM Employee;
```

But if we check this result set we will see that this result is wrong.

```
SELECT min(salary)
FROM Employee
WHERE DepartmentID =1;
```

OR

```
SELECT min(Salary), DepartmentID
FROM Employee
GROUP BY DepartmentID;
```

Department 1 does not have the lowest salary! While we can use Limit keyword – it is not always wise to do so.

So we break the problem into two components. Find the lowest salary, then find the departmentid rows where the salary matches the value of the first query.

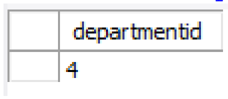
```
SELECT MIN(Salary)
FROM Employee;
```

The result set is one value {41000} we then use that value in our next query

```
SELECT DepartmentID
FROM Employee
WHERE Salary = 41000;
```

This can be done in the one statement:

```
SELECT departmentid
FROM Employee
WHERE SALARY =
    (SELECT MIN(Salary)
     FROM Employee);
```



	departmentid
	4

The query in parenthesis is known as the "inner query" and the other query is known as the "outer query".

This is doing exactly the same as two SQL statements above it but in one statement. The query in parenthesis is run first and the result returned (41000) then each row in the outer query matches the salary value with the value returned in the inner query. The only match is where Department ID is 4.

> *Important: if the subquery returns more than one result "=" will not work you must use "IN"*

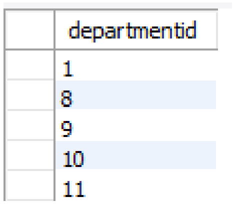
How many employees work in departments are on the fifth floor?

To approach this question, we break it into separate components.

1. What departmentid's are on the fifth floor
2. Count the number of employees whose departmentid matches the results returned in the inner query who are in the same department as the first query

Query 1 The "inner" query

```
SELECT departmentid
FROM Department
WHERE Floor = 5;
```



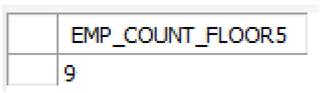
	departmentid
	1
	8
	9
	10
	11

Returns the result set {1,8,9,10,11} that is five rows.

Now we need to count each row in the Employee table where the departmentID matches 1 or 8 or 9 or 10 or 11.

As the result set has more than one row we need to use the keyword IN

```
SELECT count(employeeid)
FROM Employee
WHERE departmentid IN
    (SELECT departmentID
     FROM Department
     WHERE Floor = 5);
```



	EMP_COUNT_FLOOR5
	9

TRY: If you have time replace the keyword IN with" =" and observe the error in the query window

Error Code: 1242. Subquery returns more than 1 row

End of Week 3 Lab

SQL Homework

If you have time you can attempt the following SQL questions or do them at home against the department store schema you have installed on your computer.

Most of this week's homework requires you to read the manual. That is the functions section of the MySQL reference manual https://dev.mysql.com/doc/refman/5.7/en/functions.html

H1 How many deliveries have there been in the month of July?

Hint: the only information you have been given is the month name

```
SELECT COUNT(DeliveryID)
FROM Delivery
WHERE Monthname(deliverydate) = 'July';
```

H2 List the names of the tents available for sale

```
SELECT name
FROM Item
WHERE name like '%Tent%';
```

H3 What month has had the highest number of sales?

```
SELECT Count(SaleID), Monthname(SaleDate)
FROM SAle
Group By Monthname(SaleDate)
Order by Count(SaleID) DESC
LIMIT 1;
```

H4 List the salary total and employee count for each departmentid. Order by the smallest salary total to largest.

```
SELECT DepartmentID, COUNT(employeeid), SUM(Salary)
FROM Employee
GROUP BY DepartmentID
ORDER BY Sum(Salary);
```

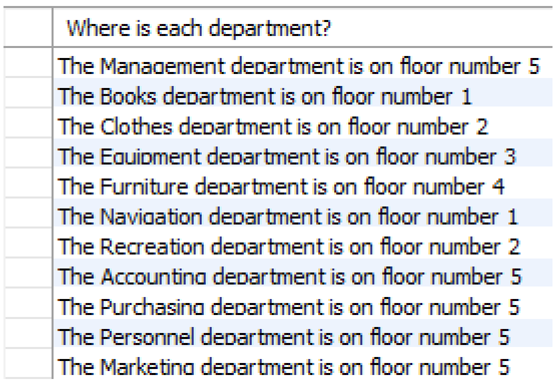
H5 How many sales have been on a Sunday?

```
SELECT count(saleID)
FROM sale
Where DayName(Saledate) = 'Sunday';
```

H6 How many days have elapsed between the first delivery date and most recent delivery date for each supplier?

```
SELECT SupplierID,
Datediff(max(deliverydate),min(deliverydate)),
count(distinct(deliverydate))
FROM Delivery
Group by SupplierID;
```

H7 Produce the following output by writing a SQL statement



Where is each department?
The Management department is on floor number 5
The Books department is on floor number 1
The Clothes department is on floor number 2
The Equipment department is on floor number 3
The Furniture department is on floor number 4
The Navigation department is on floor number 1
The Recreation department is on floor number 2
The Accounting department is on floor number 5
The Purchasing department is on floor number 5
The Personnel department is on floor number 5
The Marketing department is on floor number 5

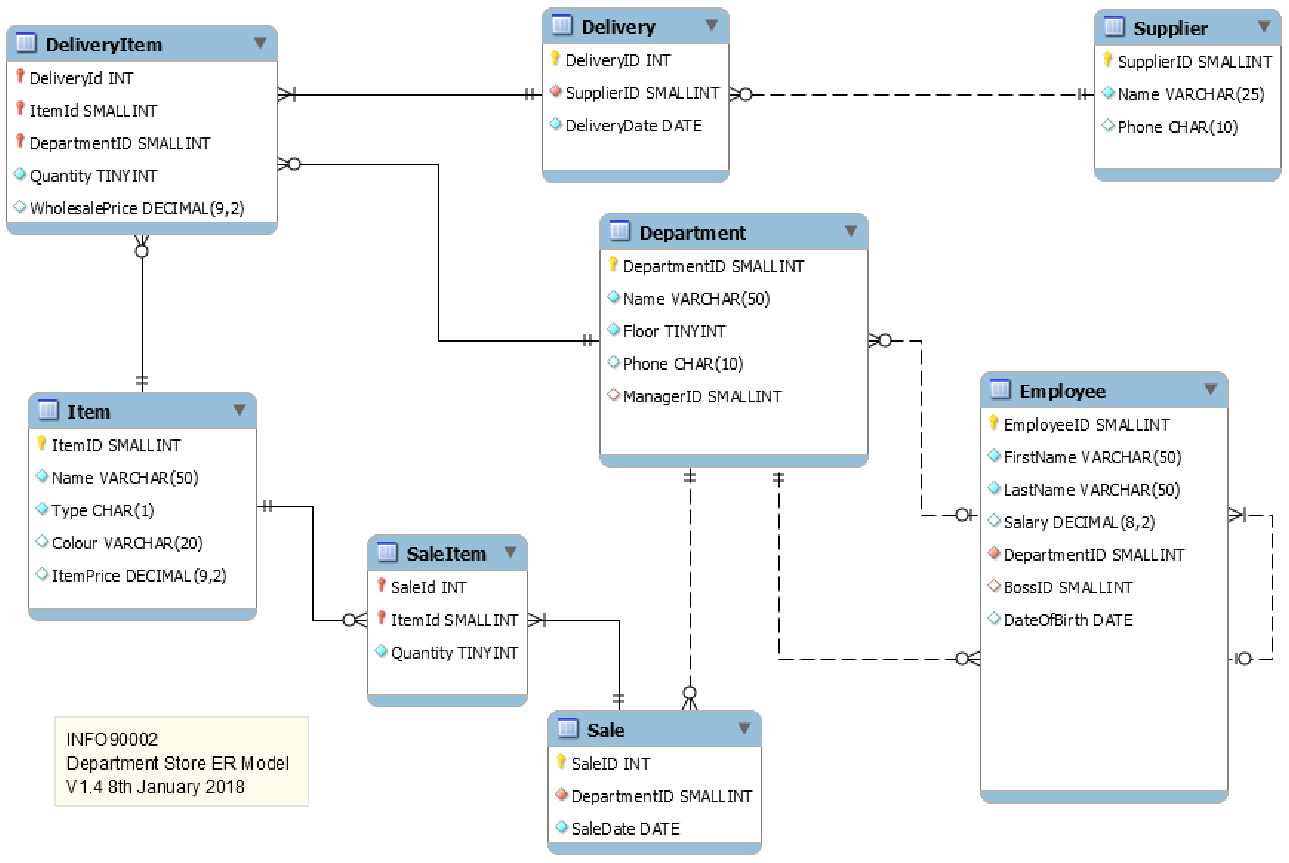
Note there is no _ in the title it is "Where is each department?" not Where_is_each_department?

```
SELECT concat('The ', name, ' department is on floor
number ',floor) as "Where is each department?"
FROM Department;
```

H8 Find the minimum, maximum, average and standard deviation for salaries in each department

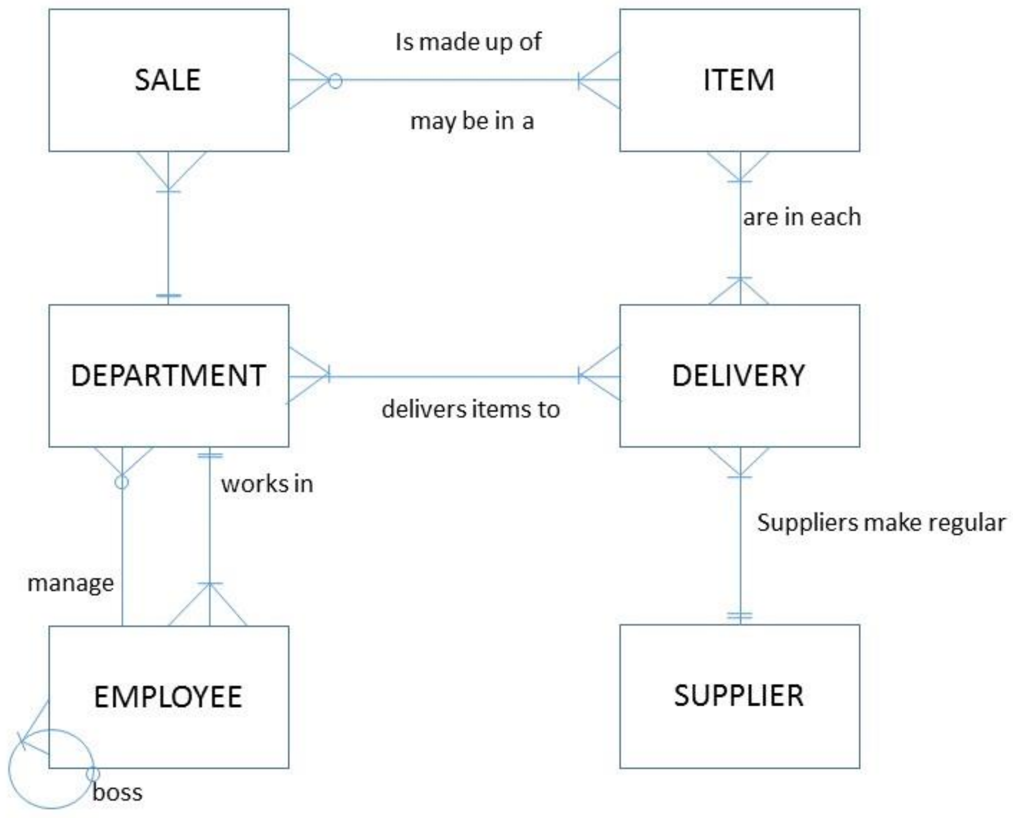
```
SELECT DepartmentID, MIN(Salary), Max(Salary),
STDDEV(Salary)
FROM Employee
GROUP BY DepartmentID;
```

Appendix – Department Store ER Model



Appendix – Department Store conceptual design

Department Store – Conceptual Model (suggested solution)


SALE
Is made up of
may be in a
ITEM
are in each
DEPARTMENT
delivers items to
DELIVERY
works in
manage
Suppliers make regular
EMPLOYEE
SUPPLIER
boss