INFO20003 Week 5 Lab

Objectives:

- Install the lab schemas, tables and data
- Learn SQL (Structured Query Language) SELECT syntax
- Practice writing SQL queries
- Join tables using natural and inner joins

Section 1: Confirm the schema install

In Lab 4, you were asked to install the "department store" schema named info20003labs-DepStore.sql.

◆ **Task 1.1** Confirm that you have installed the department store schema.

Log into the info20003db server (or your own MySQL Server) and look in the **Schemas** panel on the left.

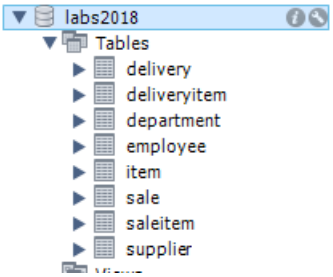

Figure 1: Tables that make up the lab schema

If you have the above tables, continue to Section 2. Otherwise, refer to the week 4 lab document for installation instructions.

Section 2: Beginning SQL

- Single-table SELECT statements
- Functions, operators and conditions
- Subqueries and joins

◆ **Task 2.1** Using MySQL Workbench, connect to the lab schema on MySQL Server.

◆ **Task 2.2** Show the metadata about the *department* table:

```
DESCRIBE department;
```

The DESCRIBE statement shows information about the columns that make up the table.

Field	Type	Null	Key	Default	Extra
departmentID	smallint(6)	NO	PRI	0	
Name	varchar(50)	NO		NULL	
Floor	tinyint(4)	NO		NULL	
Phone	char(10)	YES		NULL	
ManagerID	smallint(6)	YES	MUL	NULL	

For each column, MySQL returns the columns name, data type, whether NULL values are allowed in the column, whether the column is part of a primary key (PRI) or foreign key (MUL), and whether there is a default value for any new row.

The SELECT statement

SELECT statements retrieve and display data from the database. 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
ORDER BY (grouping list) -- optional
LIMIT (number of rows) OFFSET (number of rows) -- optional
```

To select all columns from a table we use the SQL shorthand `*`. To select all data from the *department* table, we enter the following SQL:

```
SELECT *
FROM department;
```

The result set is as follows:

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

◆ **Task 2.3** Type the SQL to select all data from the *employee* table.

Your result set should look like this:

EmployeeID	FirstName	LastName	Salary	DepartmentID	BossID	DateOfBirth
1	Alice	Munro	125000.00	1	NULL	1966-12-14
2	Ned	Kelly	85000.00	11	1	1970-07-16
3	Andrew	Jackson	55000.00	11	2	1958-04-01
4	Clare	Underwood	52000.00	11	2	1982-09-22
5	Todd	Beamer	68000.00	8	1	1965-05-24
6	Nancy	Cartwright	52000.00	8	5	1993-04-11
7	Brier	Patch	73000.00	9	1	1981-10-16
8	Sarah	Fergusson	86000.00	9	7	1978-11-15
9	Sophie	Monk	75000.00	10	1	1986-12-15
10	Sanjay	Patel	45000.00	6	3	1984-01-28
11	Rita	Skeeter	45000.00	2	4	1988-02-22
12	Gigi	Montez	46000.00	3	4	1992-03-20
13	Maggie	Smith	46000.00	3	4	1991-04-29
14	Paul	Innit	41000.00	4	3	1998-06-02
15	James	Mason	45000.00	4	3	1995-07-30
16	Pat	Clarkson	45000.00	5	3	1997-08-28
17	Mark	Zhang	45000.00	7	3	1996-10-01
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Filtering results

To select only some columns in a table, we need to specify the columns of the table in the SELECT statement separated with commas.

If you describe the Department table (`DESCRIBE department`), you can see columns called 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 *projection* (or *vertical filtering*) of the result set.

◆ **Task 2.4** Type an SQL query to select the first name, last name and department ID for every employee in the *employee* table.

Your result set should look like this:

firstname	lastname	departmentid
Alice	Munro	1
Ned	Kelly	11
Andrew	Jackson	11
Clare	Underwood	11
Todd	Beamer	8
Nancy	Cartwright	8
Brier	Patch	9
Sarah	Fergusson	9
Sophie	Monk	10
Sanjay	Patel	6
Rita	Skeeter	2
Gigi	Montez	3
Maggie	Smith	3
Paul	Innit	4
James	Mason	4
Pat	Clarkson	5
Mark	Zhang	7

Until now we have selected all rows in a table. Most of the time, we don't want to retrieve all rows from a table. Frequently we may wish to retrieve a subset of rows only.

In SQL we do this by using a *condition* on our query. This is known as *selection* (or *horizontal filtering*) of the result set. If, for example, we wish to list all departments that are located on the second floor, we would type:

```
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

How do we find out all department names that start with M? To do this, we use pattern matching using the LIKE operator. When used with LIKE, the *wildcard* character % means that 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 using the LIKE operator:

```
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

◆ **Task 2.5** Type a query to return the first and last name, salary and department ID of all employees who earn exactly $45,000.

Your result set should look like this:

	firstname	lastname	salary	departmentid
	Saniav	Patel	45000.00	6
	Rita	Skeeter	45000.00	2
	James	Mason	45000.00	4
	Pat	Clarkson	45000.00	5
	Mark	Zhang	45000.00	7

Multiple conditions: AND / OR

Sometimes we may need to filter the result set by more than one condition. For example, to list all the Departments that start with M for which the manager ID is 1, the SELECT query is as follows:

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

In this case, both conditions must be true, and only one 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.

```
SELECT *
FROM department
WHERE Name LIKE 'M%'
  OR ManagerID = 1;
```

The query lists all departments starting with M as well as 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

When we use an OR condition, both conditions are evaluated independently of each other and only one condition needs to be true for the row to be displayed.

Conditions

The table below summarises the conditional operators and their interpretation in SQL:

Condition	Meaning	Example	Explanation
>	Greater than	`salary > 55000`	value is above 55,000
<	Less than	`salary < 45000`	value is below 45,0
=	Equal to	`lastname = 'Underwood'`	must match exactly
>=	Greater than or equal to	`salary >= 45000`	value is 45,000 or lower
<=	Less than or equal to	`salary <= 45000`	value is 45,000 or higher
!= <>	Not equal to	`name != 'Torch'` `name <> 'Torch'`	can be any value except Torch

To select all departments located 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;
```

Alternatively:

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

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

Note that `!=` and `<>` are both functionally equivalent to “not equal to”.

Here is a query that lists all employees who work in department ID 11 **and** earn more than $55,000:

```
SELECT firstname, lastname, salary
FROM employee
```

```
WHERE departmentID = 11
  AND salary > 55000;
```

	firstname	lastname	salary
	Ned	Kelly	85000.00

◆ **Task 2.6** Run the above query using OR in place of AND.

The result set should look like this:

	firstname	lastname	salary
	Alice	Munro	125000.00
	Ned	Kelly	85000.00
	Andrew	Jackson	55000.00
	Clare	Underwood	52000.00
	Todd	Beamer	68000.00
	Brier	Patch	73000.00
	Sarah	Fergusson	86000.00
	Sophie	Monk	75000.00

Notice Clare Underwood's salary (52000.00), which is less than $55,000, but the department ID condition is satisfied.

Sorting: ORDER BY

Result sets can be ordered by any column (or a set of columns). Because SQL does not guarantee that the result set will be in the same order every time a query is executed, you can enforce a particular order by using the ORDER BY keyword:

```
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 ascending, that is from the smallest value to the largest (1 → 10 or A → Z). You can explicitly state this by typing ASC (short for Ascending order). To sort from the 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 a different order for each column. In the following example, the departments are ordered from top floor to bottom floor, then departments on the same floor are ordered from smallest to largest departmentID:

```
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

◆ **Task 2.7** Type a query that returns the first and last name and salary of all employees who earn more than $45,000. Order the result from the highest earner to the lowest.

Your result set should look like this:

FirstName	LastName	Salary
Alice	Munro	125000.00
Sarah	Fergusson	86000.00
Ned	Kelly	85000.00
Sophie	Monk	75000.00
Brier	Patch	73000.00
Todd	Beamer	68000.00
Andrew	Jackson	55000.00
Nancy	Cartwright	52000.00
Clare	Underwood	52000.00
Gigi	Montez	46000.00
Maggie	Smith	46000.00

LIMIT

We can select only the first *x* rows of the result set by using the word LIMIT and specifying an integer *x*:

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

	Name
	Accounting
	Management

- **Task 2.8** Type the same query and note the two rows returned. Change the ORDER BY from ASC to DESC and rerun the query. Is the result set different? Why or why not?

- **Task 2.9** Write a query to list the first and last names of the five highest-paid employees.

The result set should look like this:

	FirstName	LastName
▶	Alice	Munro
	Sarah	Fergusson
	Ned	Kelly
	Sophie	Monk
	Brier	Patch

About functions in SQL

Functions are mathematical and scientific calculations that are performed automatically by the database engine. There are two classes of functions in SQL:

- There are ordinary, *non-aggregate* functions that take one or more inputs and return an output value, such as SQRT and CONCAT. These behave in a similar way to functions in Python, C, Java, etc.
- *Aggregate* functions are functions that deliver information about multiple rows in a result set. For example, the COUNT function counts the number of rows in the result set. Other key aggregate functions are MAX, MIN, SUM and AVG.

A full list of functions you can use in MySQL is found in the MySQL reference manual.

A non-aggregate function: CONCAT

CONCAT is a concatenation function over string values that joins columns together.

To join the *firstname* and *lastname* columns of the *employee* table together, we can use CONCAT:

```
SELECT CONCAT(firstname, lastname)
FROM employee;
```

concat(firstname, lastname)
AliceMunro
NedKelly
AndrewJackson
ClareUnderwood
ToddBeamer
NancyCartwright
BrierPatch
SarahFergusson
SophieMonk
SanjayPatel
RitaSkeeter
GigiMontez
MaggieSmith
PaulInnit
JamesMason
PatClarkson
MarkZhang

However, this doesn't look so great, as there is no space between the first and last names. We can insert the space as an additional parameter to CONCAT as shown below:

```
SELECT CONCAT(firstname, ' ', lastname)
FROM employee;
```

CONCAT(firstname, ' ', lastname)
Alice Munro
Ned Kelly
Andrew Jackson
Clare Underwood
Todd Beamer
Nancy Cartwright
Brier Patch
Sarah Fergusson
Sophie Monk
Sanjay Patel
Rita Skeeter
Gigi Montez
Maggie Smith
Paul Innit
James Mason
Pat Clarkson
Mark Zhang

This can be very useful for generating strings:

```
SELECT CONCAT(FirstName, ' ', LastName, ' works in the ',
  department.Name, ' department') AS info
FROM employee NATURAL JOIN department;
```

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

Note: This query uses a **natural join** between two tables, Employee and Department. More information about joins will be provided later in this lab.

An aggregate function: COUNT

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

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

COUNT(*)
11

Unlike CONCAT which operates row-by-row, COUNT returns a single result value by looking at all the rows in the table at once.

The COUNT(*) function returns the number of rows that satisfy the WHERE clause of a SELECT statement. In the above query, we have not provided a WHERE clause, so the query returns the total number of rows of department table.

The following query returns the number of rows where the name of the department starts with M:

```
SELECT COUNT(*)
FROM department
WHERE Name LIKE 'M%';
```

COUNT(*)
2

COUNT (DISTINCT…) only counts each distinct entry one time. If there were two departments with the same name (starting with ‘M’), using Count(Distinct …) in the above query would return 1 less than Count(…) did as used above (eg would return just 1). Try and think of some

examples where COUNT (DISTINCT …) is more useful (eg consider if a table has a composite key….), ask your tutor (or a quick google search) if you want some more explanation!

GROUP BY

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

```
SELECT floor, COUNT(*)
FROM department
GROUP BY floor;
```

The COUNT function no longer counts all the rows in the result set. Instead, the GROUP BY clause combines all the rows for each floor into one, and the COUNT function counts how many original rows were combined:

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

In other words, grouping by the *floor* column counts the number of departments for each distinct floor. There are 5 dictinct floors {1,2,3,4,5} and the COUNT(*) is for each distinct floor.

We must group by the column or columns that are not part of the aggregate function (in this case, *floor*) to ensure the full result set is returned. If you tried removing the GROUP BY keyword, the count is correct (there are indeed 11 departments in total) but which floor number should be displayed?

```
SELECT floor, COUNT(departmentID)
FROM department;
```

floor	count(departmentid)
??	11

Not all the departments are on the same floor. The above query no longer works on modern MySQL versions (previously, MySQL would pick any floor number).

Column aliases

To help make the query and its result set easier to understand, we can alias the columns. To alias a column use AS *youraliasname*:

```
SELECT floor AS dept_floor, COUNT(*) 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

You can refer to a column by its alias in the GROUP BY and ORDER BY clauses, but **not** in the WHERE clause.

◆ **Task 2.10** Type a query to find how many employees work in each department. Alias the two columns of the result set to *dept* and *staff_count* respectively.

Your result set should look something like this:

	DEPT	STAFF_COUNT
	1	1
	2	1
	3	2
	4	2
	5	1
	6	1
	7	1
	8	2
	9	2
	10	1
	11	3

Suppose we are asked, “What is the average salary of each department?”

```
SELECT departmentID, AVG(salary)
FROM employee
GROUP BY departmentID;
```

	departmentid	AVG(salary)
	1	125000.000000
	2	45000.000000
	3	46000.000000
	4	43000.000000
	5	45000.000000
	6	45000.000000
	7	45000.000000
	8	60000.000000
	9	79500.000000
	10	75000.000000
	11	64000.000000

Note: We will deal with the excessive decimal places of the AVG() function later in this lab.

◆ **Task 2.11** What is the maximum salary for each department? Use the MAX() aggregate function.

departmentid	MAXSAL
1	125000.00
9	86000.00
11	85000.00
10	75000.00
8	68000.00
3	46000.00
2	45000.00
4	45000.00
5	45000.00
6	45000.00
7	45000.00

◆ **Task 2.12** Find the department which has the highest **average** salary.
Hint: Start with the query just above Task 2.11. Order the result set by salary, then use the LIMIT keyword to display only a single row.

departmentID	AvgSalary
1	125000.00

HAVING

The HAVING keyword is used when you wish to put a condition on an aggregate function. It is always used with GROUP BY.

To list the departments having an average salary over $55,000, this query could be used:

```
SELECT DepartmentID, AVG(Salary)
FROM employee
GROUP BY DepartmentID
HAVING AVG(Salary) > 55000;
```

In this case we want only the (grouped) rows with AVG(salary) greater than $55,000.

DepartmentID	AVG(Salary)
1	125000.000000
8	60000.000000
9	79500.000000
10	75000.000000
11	64000.000000

◆ **Task 2.13** Find the department IDs of departments with only one employee.
Hint: Use the previous query as a model, replacing AVG(Salary) with the COUNT function.

DepartmentID
1
2
5
6
7
10

Remember that HAVING is the only way to use a condition over an aggregate function (e.g. AVG, MAX, SUM, COUNT, etc) in a query which has a GROUP BY clause.

Formatting the output

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 X to D decimal places. FORMAT will format the argument X to D decimal places and include commas as thousand separators.

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

AVG_SAL
60529.411765

```
SELECT FORMAT(AVG(SALARY), 2) AS AVG_SAL
FROM employee;
```

AVG_SAL
60,529.41

```
SELECT ROUND(AVG(SALARY), 2) AS AVG_SAL
FROM employee;
```

AVG_SAL
60529.41

Note that FORMAT and ROUND are subtly different. FORMAT converts the output into a string (hence the use of the comma at the thousand position), whereas ROUND keeps the result as a number.

Subqueries

Sometimes we want to perform an operation over a set of values. For example, suppose we wanted to find the employees whose salary is below the average. We could run a query to find the average salary across all employees:

```
SELECT AVG(Salary)
FROM employee;
```

AVG(Salary)
60529.411765

To find the employees paid less than this amount, we could manually type this number into the WHERE clause of a second query:

```
SELECT FirstName, LastName, Salary
FROM employee
WHERE Salary < 60529.411765;
```

FirstName	LastName	Salary
Andrew	Jackson	55000.00
Clare	Underwood	52000.00
Nancy	Cartwright	52000.00
Sanjay	Patel	45000.00
Rita	Skeeter	45000.00
Gigi	Montez	46000.00
Maggie	Smith	46000.00
Paul	Innit	41000.00
James	Mason	45000.00
Pat	Clarkson	45000.00
Mark	Zhang	45000.00

However, this is a time-consuming approach. Using subqueries, we can combine these two queries into a single SELECT statement:

```
SELECT FirstName, LastName, Salary
FROM employee
WHERE Salary < (SELECT AVG(Salary)
                FROM employee);
```

FirstName	LastName	Salary
Andrew	Jackson	55000.00
Clare	Underwood	52000.00
Nancy	Cartwright	52000.00
Sanjay	Patel	45000.00
Rita	Skeeter	45000.00
Gigi	Montez	46000.00
Maggie	Smith	46000.00
Paul	Innit	41000.00
James	Mason	45000.00
Pat	Clarkson	45000.00
Mark	Zhang	45000.00

The query in the parentheses is known as the *inner query* or *subquery*, and the other query is known as the *outer query*.

The inner query is run first, and the result is returned (60529.411765). Then each row in the outer query compares that row's salary value with the value returned by the inner query. Only those rows where the condition is true are returned.

◆ **Task 2.14** Find the names of employees who work in the 'Marketing' department. *Hint: Break the problem into two parts: find the department ID of the 'Marketing' department (inner query) and find the names of employees with this department ID (outer query).*

Subqueries with many results

If the subquery returns more than one result, = will not work. You must use the IN keyword.

Consider the following query:

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

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

1. Which departments are located on the fifth floor?
2. Count the number of employees whose departmentID is in the result set of the inner query.

Query 1, the inner query:

```
SELECT departmentID
FROM department
WHERE Floor = 5;
```

	departmentid
	1
	8
	9
	10
	11

This returns the result set {1,8,9,10,11}.

Now we need to count the number of rows 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 as part of the WHERE clause:

```
SELECT COUNT(*) AS fifth_floor_count
FROM employee
WHERE departmentID IN (SELECT departmentID
                       FROM department
                       WHERE Floor = 5);
```

	fifth_floor_count
▶	9

Try replacing the keyword IN with = and observe the error in the query window (Error Code: 1242. Subquery returns more than 1 row).

Natural and inner joins

Typically, relational database management systems have many entities in a database schema. To retrieve all the necessary data, tables frequently need to be joined together. Table

joins can take many forms in SQL. They include natural joins, inner joins, left joins, right joins and full outer joins. We will now look at natural and inner joins in SQL.

NATURAL JOIN

Natural joins are used when the joining column name is identical in both tables that are being joined together.

Consider the department table:

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

And the employee table:

EmployeeID	FirstName	LastName	Salary	DepartmentID	BossID	DateOfBirth
1	Alice	Munro	125000.00	1	NULL	1966-12-14
2	Ned	Kelly	85000.00	11	1	1970-07-16
3	Andrew	Jackson	55000.00	11	2	1958-04-01
4	Clare	Underwood	52000.00	11	2	1982-09-22
5	Todd	Beamer	68000.00	8	1	1965-05-24
6	Nancy	Cartwright	52000.00	8	5	1993-04-11
7	Brier	Patch	73000.00	9	1	1981-10-16
8	Sarah	Fergusson	86000.00	9	7	1978-11-15
9	Sophie	Monk	75000.00	10	1	1986-12-15
10	Sanjay	Patel	45000.00	6	3	1984-01-28
11	Rita	Skeeter	45000.00	2	4	1988-02-22
12	Gigi	Montez	46000.00	3	4	1992-03-20
13	Maggie	Smith	46000.00	3	4	1991-04-29
14	Paul	Innit	41000.00	4	3	1998-06-02
15	James	Mason	45000.00	4	3	1995-07-30
16	Pat	Clarkson	45000.00	5	3	1997-08-28
17	Mark	Zhang	45000.00	7	3	1996-10-01
NULL	NULL	NULL	NULL	NULL	NULL	NULL

The departmentID column is common to both tables. There is a foreign key (departmentID) in the employee table that references the primary key (departmentID) in the department table.

To find the department name for each employee, we would need to retrieve the name from the department table and firstname and lastname from the employee table (joined over departmentID).

```
SELECT department.name, employee.firstname, employee.lastname
FROM employee NATURAL JOIN department;
```

The result set would be:

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

Because each column name in the SELECT command is unique to the query, the department name and employee names could also be written without the table name prefacing the column name:

```
SELECT name, firstname, lastname
FROM employee NATURAL JOIN department;
```

Natural Joins work when the column name is identical in **both** name and purpose for the two tables being joined together. However, there are times when the column names are identical, but the meaning is different, and they refer to different characteristics.

For example, both the *department* and *item* tables have a column called *name*. But the purpose is different. The name column in the *department* table is the department's name and the name column in the *item* table is the item's name. Yet despite there being no PK-FK relationship between *item* and *department* tables, the following statement will execute:

```
SELECT itemid, departmentID
FROM item NATURAL JOIN department;
```

It returns 0 rows because there are no common names in the respective name columns of the *item* and *department* tables. The MySQL database server does **not** return an error because the SQL is *syntactically* correct, although it is *logically* incorrect.

INNER JOIN

Inner joins are used when the joining column names are not identical. Inner joins provide an explicit definition of what the join condition is. INNER JOIN must always be followed with the ON clause which specifies the join condition.

The syntax for an INNER JOIN is as follows (note the ON clause):

```
SELECT table1.column1, table1.column2, table2.column2
FROM table1
  INNER JOIN table2 ON table1.column1 = table2.column4;
```

Inspect the earlier Department / Employee name query now rewritten as an inner join:

```
SELECT name, firstname, lastname
FROM department
  INNER JOIN employee ON department.departmentID = employee.departmentID;
```

Joins can be combined with other SQL features, such as GROUP BY. The following query lists the supplier names and number of deliveries made to the department store:

```
SELECT supplier.Name, COUNT(delivery.DeliveryID) AS Deliveries
FROM supplier NATURAL JOIN delivery
GROUP BY supplier.Name;
```

Name	Deliveries
All Points Inc.	3
All Sports Manufacturing	2
Global Books & Maps	3
Nepalese Corp.	3
Sao Paulo Manufacturing	4
Sweatshops Unlimited	1

◆ **Task 2.15** Type the names and salaries of the employees who earn more than any employee in the marketing department.
Hint: Split the problem up into 2 components – calculate the maximum salary of people in marketing (inner query) and then who earns more than that (outer query).

Your result set should look like this:

FirstName	LastName	Salary
Alice	Munro	125000.00
Sarah	Fergusson	86000.00

◆ **Task 2.16** Find the name and salary of Clare Underwood's manager.

```
SELECT firstname, lastname, salary
FROM employee
WHERE employeeid =
    (SELECT bossid
     FROM employee
     WHERE firstname = 'Clare'
       AND lastname = 'Underwood');
```

FirstName	LastName	Salary
Ned	Kelly	85000.00

◆ **Task 2.17** List the first and last names of bosses who supervise more than two staff and who also manage a department.

```
SELECT boss.employeeid, boss.firstname, boss.lastname,
  COUNT(emp.employeeid) AS employee_count
FROM employee AS emp
  INNER JOIN employee AS boss ON emp.bossid = boss.employeeid
WHERE boss.employeeid IN (SELECT managerid
                          FROM department)
GROUP BY boss.employeeid
HAVING COUNT(emp.employeeid) > 2;
```

	employeeid	firstname	lastname	employee_count
	1	Alice	Munro	4
	3	Andrew	Jackson	5
	4	Clare	Underwood	3