

S2.2: SQL 2

We are going to continue working with the EMP and DEPT tables that we created last week. Go to <https://apex.oracle.com/en/> and log in to your Apex Workspace.

Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	11/17/2001	82000	-	10
7782	CLARK	MANAGER	7839	06/09/2009	27500	-	10
7934	MILLER	CLERK	7782	01/23/2006	13200	-	10
7566	JONES	MANAGER	7839	04/02/2012	26850	-	20
7788	SCOTT	ANALYST	7566	12/09/2008	19500	-	20
7902	FORD	ANALYST	7566	12/03/2014	21500	-	20
7369	SMITH	CLERK	7902	12/17/2002	13750	-	20
7876	ADAMS	CLERK	7788	01/12/2011	11900	-	20
7698	BLAKE	MANAGER	7839	05/01/2005	24000	-	30
7499	ALLEN	SALESMAN	7698	02/20/2006	19000	6300	30
7521	WARD	SALESMAN	7698	02/22/2018	18500	4500	30
7654	MARTIN	SALESMAN	7698	09/28/2008	15650	3400	30
7844	TURNER	SALESMAN	7698	09/08/2011	18500	0	30
7900	JAMES	CLERK	7698	12/03/1998	11950	-	30
4422	STEVENS	MANAGER	7839	02/14/2019	24750	-	50
3258	GREEN	SALESMAN	4422	07/24/2003	18200	2750	50
6548	BARNES	CLERK	4422	01/16/2018	11950	-	50

Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	LONDON
20	RESEARCH	PRESTON
30	SALES	LIVERPOOL
40	OPERATIONS	MANCHESTER
50	MARKETING	LUTON

NOTE: There are some SQL exercises at the end of this worksheet, however, read through all of the pages prior to attempting any of the questions. It is also very important to try out all the examples as you will be able to see the output which will help you gain a better understanding of how SQL works.

JOINING TABLES

The real power of relational databases comes from the relationships between the tables. These relationships allow us to write queries that combine rows from more than one table, based on a related column between them.

Note: in SQL there is more than one way to join tables. See [Oracle Documentation](https://www.w3schools.com/sql/default.asp) or <https://www.w3schools.com/sql/default.asp> to gain an understanding of alternatives.

Example 1: Find Allen's name and salary from the EMP table and the location of Allen's department from the DEPT table...

```
SELECT ename, sal, loc FROM emp
JOIN dept ON emp.deptno = dept.deptno
WHERE ename = 'ALLEN';
```

Notice that the relationship between the EMP and DEPT tables is the DEPTNO column.

Example 2: List the name and department of all employees in name order...

```
SELECT ename, dname FROM emp
JOIN dept ON emp.deptno = dept.deptno
ORDER BY ename;
```

Because we are now referencing two tables which each have a column with the same name (deptno), we must qualify deptno with its table name to prevent confusion. This qualification must be used whenever ambiguous column names are used within an SQL statement.

For example, in the following query, if we didn't have ORDER BY emp.deptno, but had ORDER BY deptno, we would get a syntax error because it would not know whether to sort on the deptno in Emp or Dept (even though it is actually the same thing). Try it without the emp prefix!

```
SELECT dname, emp.* FROM emp
JOIN dept ON emp.deptno = dept.deptno
WHERE loc = 'LUTON'
ORDER BY emp.deptno;
```

If the related column is named the same in both tables (as it is in this case) we could also use the **USING** keyword and write the query as follows:

```
SELECT ename, sal, loc FROM emp
JOIN dept USING (deptno)
WHERE ename = 'ALLEN';
```

Note: when using the USING keyword we must place the related column (deptno) in parenthesis

Aliases

Aliases can be used to apply a temporary name to a table or a column to make queries more readable.

Look at the following two examples:

Without Aliases:

```
SELECT dname, emp.* FROM emp
JOIN dept ON emp.deptno = dept.deptno
WHERE loc = 'LUTON'
ORDER BY emp.deptno;
```

With Aliases:

```
SELECT dname, E.* FROM emp E
JOIN dept D ON E.deptno = D.deptno
WHERE loc = 'LUTON'
ORDER BY E.deptno;
```

The example above demonstrates use of an alias for table names. You can also apply aliases to column names as in the example below:

```
SELECT ename AS Employee_Name, dname FROM emp
JOIN dept USING (deptno)
WHERE dname = 'SALES'
ORDER BY Employee_Name;
```

If you run the above query you will see in the output that instead of the column header being displayed as 'ename', it is now shown as 'Employee_Name'.

It also works without the 'AS' keyword:

```
SELECT ename Employee_Name, dname FROM emp
JOIN dept USING (deptno)
WHERE dname = 'SALES'
ORDER BY Employee_Name;
```

You will notice that if we take the underscore out from between 'Employee' and 'Name', and leave a space we get an error. The way around this is to enclose the alias in double quotes as below:

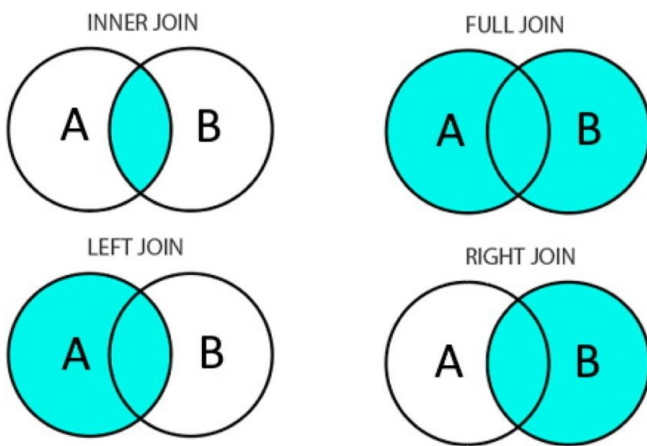
```
SELECT ename "Employee Name", dname FROM emp
```

```
JOIN dept USING (deptno)
WHERE dname = 'SALES'
ORDER BY "Employee Name";
```

NOTE: if you are copying queries containing quotes out of this worksheet you will get an 'invalid character' error. Oracle does not like some of the characters copied and pasted from Word, one of these is both the single and double quotes. So, after copying into APEX just replace (retype) the quotes in the command window.

The Different Join Types

There are four main JOIN types in SQL:



INNER Join

An INNER JOIN returns rows that have matching values in both tables. This is the most common type of join and as such is the default so we don't actually have to use the 'INNER' keyword.

Example:

```
SELECT dname, emp.* FROM emp
JOIN dept ON emp.deptno = dept.deptno
```

Gives the same as:

```
SELECT dname, emp.* FROM emp
INNER JOIN dept ON emp.deptno = dept.deptno
```

Try the following example:

```
SELECT dept.deptno, dname, ename FROM emp
```

JOIN dept ON emp.deptno = dept.deptno

You will notice that details of department 40 never appear in the output. This is because department 40 currently has no employees, so has no corresponding rows in the emp table and therefore cannot take part in the join. If it is required to include records which are outside of the relationship between tables another type of join must be used. See if you can work out which one!

LEFT Join

Sometimes known as a LEFT OUTER JOIN. A LEFT JOIN returns rows from the left table, and matching values from the right table. The left table is the table mentioned first in the SELECT FROM statement, and the right table is the one mentioned in the JOIN line. In the example below EMP is the left table and DEPT is the right table.

**SELECT dept.deptno, dname, ename FROM emp
LEFT JOIN dept ON emp.deptno = dept.deptno**

You will see that a left join does not give us details of department 40 in the results because there is no match in the emp table.

RIGHT Join

Sometimes known as a RIGHT OUTER JOIN. A RIGHT JOIN returns rows from the right table, and the matching values from the left table.

**SELECT dept.deptno, dname, ename FROM emp
RIGHT JOIN dept ON emp.deptno = dept.deptno**

You have probably worked out that a RIGHT JOIN will show us the details for department 40. This is because in the query above emp is the left table and dept is the right table (the only table containing details of department 40).

FULL Join

Sometimes known as a FULL OUTER JOIN. A FULL JOIN returns all rows when there is a match in either the left or the right table.

**SELECT dept.deptno, dname, ename FROM emp
FULL JOIN dept ON emp.deptno = dept.deptno**

Joining a Table to Itself

As well as the four joins mentioned above you can also join a table to itself as though it were two separate tables. Why would you want to do this? Well, this can be useful as it allows you to find a specific record in the first copy and then search the second copy for comparisons...

Example 1: Find each employees Managers name...

```
SELECT WORKER.ename, MANAGER.ename as MANAGER FROM emp WORKER, emp MANAGER
WHERE WORKER.mgr = MANAGER.empno
```

In this query the EMP table is treated as if it were two separate tables (**WORKER** and **MANAGER**).

In the WHERE clause all the **WORKERS** are joined to their **MANAGERS** using the **WORKER's** manager's employee number (**WORKER.mgr**) and the **MANAGER's** employee number (**MANAGER.empno**).

Example 2: Find all employees that earn more than Jones...

```
SELECT X.ename, X.sal, X.job, Y.job, Y.ename, Y.sal FROM emp X, emp Y
WHERE X.sal > Y.sal
AND Y.ename = 'JONES';
```

Or use this variation to produce a better output:

```
SELECT X.ename, X.sal, X.job FROM emp X, emp Y
WHERE X.sal > Y.sal
AND Y.ename = 'JONES';
```

SQL FUNCTIONS

SQL has a wide range of functions which may be applied to data. There are four classes of functions:

String functions for searching and manipulating strings.

Numeric functions for performing calculations on numeric values.

Date and Time functions for reformatting and performing data arithmetic.

Aggregate functions for calculations on groups of data.

Useful String Functions

String functions are built-in functions that allow you to manipulate string data in various ways.

NOTE: when you wish to SELECT something, but the data is not in a table (as some of the examples below), you can use a dummy table name called **DUAL**. This is necessary because SQL in Oracle insists on a FROM clause.

***This table is only recognised by Oracle as a dummy table and will never appear as an actual structure.**

Try the following examples in APEX to see the results of each:

LOWER (string)	Converts upper case alphabetic characters to lower case. Other characters are not affected: SELECT LOWER ('MR. SAMUEL HILLHOUSE') FROM DUAL;
INITCAP (string)	Converts the first character of each word in a string to uppercase and other characters to lowercase: SELECT INITCAP ('MR. SAMUEL HILLHOUSE') FROM DUAL;
UPPER (string)	Converts lowercase letters in a string to uppercase: SELECT UPPER ('Mr . Rodgers') FROM DUAL;
LENGTH (char)	Gives the length in characters of specified string: SELECT LENGTH ('Anderson') FROM dual;
SUBSTR (string,startposition,length)	Shows a part of the string starting at the start position of the specified length: SELECT SUBSTR('ABCDEF',2,3) FROM DUAL;
INSTR (string1,string2)	Finds the start position of one string inside another string: SELECT INSTR('ABCDEF','DEF') FROM DUAL;
LPAD (string,length[,character])	Left pads the string with the specified fill characters to the specified length. Spaces are used by default: SELECT LPAD(ename,10) FROM emp;
RPAD (string,length[,character])	Right pads the string with the specified fill characters to the specified length. Spaces are used by default: SELECT RPAD(ename,10) FROM emp;
LTRIM (string,character[s])	Removes from the left all leading occurrences of the specified character(s): SELECT LTRIM(000050,0) FROM DUAL; SELECT LTRIM(ename,' ') FROM emp;
RTRIM (string,character[s])	Removes from the right all trailing occurrences of the specified character(s): SELECT RTRIM(ename,' ') FROM emp;
NVL (string1,string2)	Outputs string 2 only if string 1 is NULL: SELECT NVL(Comm, 0) FROM emp;

	(if an employee has no commission then 0 is displayed)
--	--

Useful Arithmetic Functions

Try the following examples in APEX to see the results of each:

ABS(numeric)	Absolute value of the number: SELECT ABS(-15) "Absolute" FROM DUAL;
MOD(num1, num2)	Returns the remainder when num1 is divided by num2: SELECT MOD (7,5) "modulo" FROM DUAL;
ROUND(numeric[,d])	Rounds the number to d decimal places, the rounding can occur to either side of the decimal point: SELECT ROUND (15.193,1) "round" FROM DUAL;
TRUNC(numeric[,d])	Truncates to d decimal places: SELECT TRUNC(15.79,1) "truncate" FROM DUAL;
CEIL(numeric)	Rounds the number up to the nearest integer: SELECT CEIL(10.6) FROM dual;
FLOOR(numeric)	Truncates the number to the nearest integer: SELECT FLOOR(10.6) FROM dual;
SQRT(numeric)	Returns the square root of the number (returns NULL if the number is negative): SELECT SQRT(25) FROM dual;
TO_CHAR(numeric[,format])	Converts a number to a character string in the specified format: SELECT TO_CHAR(sysdate,'DY') FROM dual; SELECT TO_CHAR(sysdate,'MONTH') FROM dual;

*The TO_CHAR function is one of the most useful for converting data from one form to another.

It is very important to note that if any variable contains a null value, then an SQL statement involving arithmetic will ignore it...

E.g. SELECT SUM(sal-comm) FROM emp;

Will only produce results for employees who have a non-null commission (or salary)

However, we can use COALESCE or NVL to replace the NULL values with a zero. Now the NULL values will be used in the SUM calculation:

**SELECT SUM(sal-NVL(comm, 0))
FROM emp;**

Aggregate or Grouping Functions

An aggregate function performs a calculation on multiple values and returns a single value.

COUNT (column-name)	COUNT(*) : returns the number of rows in the table, rows that contain null values are included. COUNT(column-name) returns the number of column values. COUNT(DISTINCT column-name) filters out duplicate column values. SELECT COUNT(*) FROM emp
MIN (column name)	Returns the smallest value from the specified column: SELECT MIN(sal) FROM emp WHERE deptno = 10;
MAX (column name)	Returns the largest value from the specified column: SELECT MAX(sal) FROM emp WHERE deptno = 10;
SUM (column name)	Returns the total sum of a specified column (values must be numbers) SELECT SUM(sal) FROM emp WHERE job = 'SALESMAN';
AVG (column name)	Returns the average value for a specified column (values must be

	numbers) SELECT AVG(sal) FROM emp
--	---

Aggregate functions work down the columns listed in the SELECT statement, they DO NOT act across the rows. So, if you include an aggregate in a select command, you may not select individual results as well.

For example, the following command is invalid:

```
SELECT ename, AVG(sal) FROM emp
```

ENAME has a value for each row selected while AVG(sal) has a single value for the whole query. If you use such a command SQL will display an error message.

As such it is very common to use a GROUP BY statement with aggregate functions...

Group By

The GROUP BY statement groups rows with the same values into summary rows.

Example 1: Suppose you want to know how many different job titles there are...

```
SELECT job FROM emp  
GROUP BY job;
```

GROUP BY statements are used extensively with the aggregate functions (COUNT, MIN, MAX, SUM, AVG).

Example 2: If we want to know the average salary of all employees then we can simply use the AVG function:

```
SELECT AVG(sal) FROM emp;
```

However, what if we want to know the average salary of the employees in each department. We could enter several separate AVG(sal) queries, one per department, but this is not the most efficient way to do it. This where the GROUP BY statement comes in. The GROUP BY clause divides a table into groups of rows so that the rows in each group have the same value in a specified column. We can use the GROUP BY clause to create department groups so that we can find the average salary for each department. Try the following...

```
SELECT deptno, AVG(sal) FROM emp  
GROUP BY deptno;
```

The **GROUP BY deptno** clause divides all the employees into groups on their department number (10, 20, 30, 40, 50). Then the **AVG(sal)** function is applied to the rows in each group (each department) and provides a calculation for each.

Example 3: How many employees are there in each department?

```
SELECT COUNT(*) FROM emp  
GROUP BY deptno;
```

Without the GROUP BY clause, we would just get the count of all employees. Using the GROUP BY gives us a break down of how many employees are in each department.

Exercises:

1. List the name and salary of employees in Luton.
2. List the details about all employees and the departments they work in. Show in department number order.
3. List the names and salaries of all salesmen who work in SALES.
4. List all departments that do not have any employees.
5. Display the department name alongside a sum of the salary for each department.
6. Display the department name with the number of employees working in it (include department 40 in the list even though it has no employees).
7. For each employee whose salary exceeds his manager's salary, list the employee's name and salary and the manager's name and salary (HINT: use a self-join).
8. List the employees who have BLAKE as their manager (HINT: use a self-join).
9. Find how many employees have a title of manager without listing them.
10. Calculate the average annual salary plus commission for all salesmen.
11. Find the highest and lowest salaries and the difference between them (single statement).
12. Find the number of characters in the longest department name.
13. Count the number of people in department 30 who receive a salary and a commission.
14. List the average commission of employees who receive a commission, and the average commission of all employees (assume employees who do not receive a commission attract zero commission) (single statement).
15. Find the average salary plus commission of all employees including those that do not receive a commission (assume employees who do not receive a commission receive zero commission) (single statement).
16. Calculate the daily and hourly salary for each employee in department 30, round to the nearest penny. Assume there are 22 working days in a month and 8 working hours in a day.
17. Issue the same query as the previous one except that this time truncate (TRUNC) to the nearest penny rather than round.
18. Find the total salary budget for each department, the average salary, and the number of people in each department.

19. Add another three employees to the EMP table.
20. Modify the details of one of the employees details added above.
21. Add another department to the department table.

Now have a go at writing some questions for other things you might want to find out from the data. After you have solved them with queries, swap with the person next to you to see if they can write queries for them too.

Use any remaining lab time looking at the SQL resources on [W3 Schools](#) and [Oracle Docs](#) and trying out some of the examples.

Any work not finished should be completed in your own time before the next class.