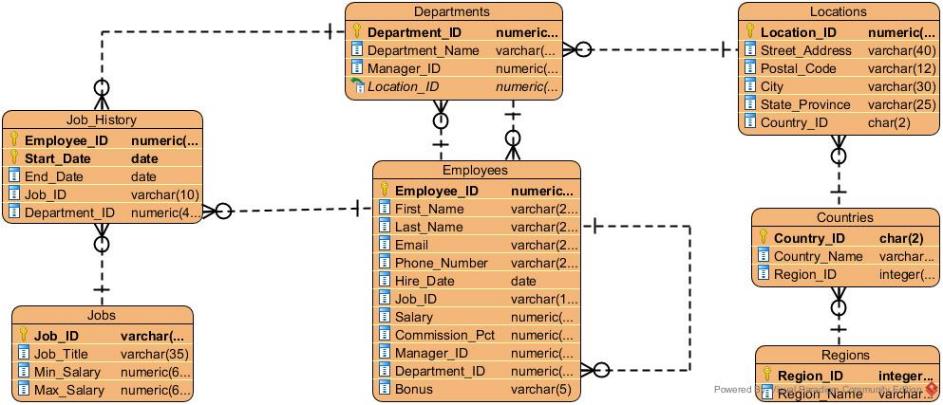
# Topic 2 – SELECT, Aliases and Row Functions

## Familiarise yourself with the Human Resource (HR) schema from Oracle

Whereas in the lectures we use a small database consisting of only three tables, for the labs we will be using the demo tables (schema) supplied by Oracle. The schema describes a human resources database used by a large multi-national company.

Use the Entity Relationship Diagram (ERD) below to familiarise yourself with the tables, relationships and attributes.



## Worked Examples, Completion Problems and Deliberate Practice

There are two main ways to learn new material and both are important. The first is through direct instruction, which is where you are taught how to do something by someone who already knows. This is what we do in lectures and we will also do it somewhat during the labs. In the labs, we will give you worked examples, which is where you are given a problem and immediately shown the correct answer with explanations. Your efforts should be focused on understanding why the provided answer is correct.

The second method is through deliberate practice, which is where you are given a problem and work to find the correct answer by yourself. Your efforts should be focused on breaking the problem down into its constituent parts and thinking through each part thoroughly.

Both of these methods are required. You cannot rely only on worked examples because you won’t have tested yourself to make sure you have properly understood the concept. Deliberate practice makes up for that by providing feedback by being able to check that the answer you thought was correct really was. You cannot rely only on deliberate practice because you won’t have learned how to approach the problem and what solutions work.

As a stepping stone between the worked examples and the deliberate practice, we will also give you some completion problems. These are questions where you are given a partial answer and you must work out the rest by yourself. This includes working out where the missing code needs to go.

It is very tempting to skip over or skim the worked examples because you may feel that once you have seen one or two examples then you have understood the concept. However, this is a mistake. It is possible to trick yourself into believing you understand something better than you do. It is worth the time and effort to follow all the examples and think about why the answer is correct. This last step is crucial because thinking brings the learning.

In a similar way, it is tempting to be superficial in your practice and rush to provide an answer without care if it’s exactly right or not. This includes not being concerned with “small errors” which you feel sure you would normally get right. This kind of approach does not lead to learning. Deliberate practice is effective precisely because you “sweat the small stuff” and do not allow yourself to be lulled into a false sense of security. This includes using the feedback (in this case the knowledge of the correct answer) properly, by examining why your answer is different and where you went wrong. But don’t stop there – ask *why* you went wrong. What assumptions did you make that led you to the error? What knowledge were you lacking?

## Mixing it Up

Another critical component of learning is to mix things up. You may think that practicing the same skill over and over is more effective than practicing a range of related skills but that is incorrect. Part of learning is knowing which skill or solution to apply in which situation. If every labsheet was independent then you would know what sort of solution is required for every question and you would not be learning to identify which solution to apply.

Therefore, these labsheets contain a mix of questions using the new skills you learnt in last week’s lecture as well as skills covered in previous lectures.

## New Material

One aspect of the labsheets is that they also introduce new material not covered in the lectures. SQL is a very large and complex language and there isn’t time during the lectures to cover every element of it. Therefore, some minor topics will be left to the labs and introduced during the worked examples. These topics may also be assessed, so don’t skip them.

## Copying and Pasting

Before you start on the actual work a quick comment on copying and pasting. In the worked examples, we provide the correct SQL code to use and ask that you try it yourself and check that the results you get are the same. It is tempting to simply copy and paste the code from this document into SQL Developer. Please don’t do this for two reasons. Firstly, making yourself type the code brings some muscle memory and is (slightly) more active than just copying and pasting. It should increase the amount of learning taking place.

Secondly, SQL syntax requires specific characters – especially for speech marks – and the characters that Word uses do not always match. So when you copy and paste from Word into SQL Developer you may run into syntax errors which can be hard to spot. So it will be less frustrating to type out the code yourself than to copy and paste and have to hunt around for the syntax error.

# Worked Examples

### Q1. List the regions that the company operates in.

The question asks us to list something, so we know we are retrieving information and will need to use the SELECT statement.

It asks us to list the regions and from looking at the schema above we see that there is a regions table, so we know we will need to get information from that table which requires a FROM regions clause.

Finally, we look at the attributes in that table and see that there are two attributes: region\_id and region\_name. The question does not explicitly say what attribute it wants, it only says we should list the regions. But the region\_id is not likely to be useful for end users whereas the region\_name is likely to be.

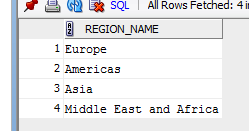
We can confirm this by using the SQL query “DESCRIBE regions;” which gives us the table structure of the regions table. We see that the region\_id is a NUMBER whereas region\_name is a VARCHAR, meaning that it contains text. We should therefore choose to show region\_name and use SELECT region\_name. We could return both but as a rule we should never return more than we need to because then we are doing more work.

The final answer is therefore:

SELECT region\_name

FROM regions;

You should get an output containing four rows:



### Q2. List the names of the departments in the company.

Again, the question asks for a list so we know it is a SELECT statement.

This time we are looking for information about departments so we see if there is a table that is obviously related to that and we find one called departments, so we know we will have a FROM departments clause.

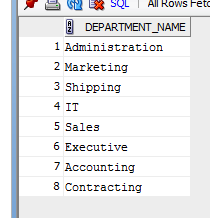
Finally, the question explicitly asks for the names of the departments and one of the attributes of the departments table is called department\_name so we will return that. This is done with a SELECT department\_name clause.

Putting it together we have the final answer:

SELECT department\_name

FROM departments;

The result should look like:



### Q3. List everything in the Countries table.

This question is more straightforward because it tells us which table to use. First we know we need a SELECT statement because we are asked to provide a list of information.

Second, we know which table to use because the question tells us. So we know we need a FROM countries clause.

Finally, we know we need to return all attributes from the table. We might be tempted to use the “all” wildcard and write “SELECT \*” but as we are novices we should stick to best practice and list the attributes explicitly. We therefore need SELECT country\_id, country\_name, region\_id.

The final answer is therefore:

SELECT country\_id, country\_name, region\_id

FROM countries;

The result should be:



### Q4. List the different salaries the company currently pays.

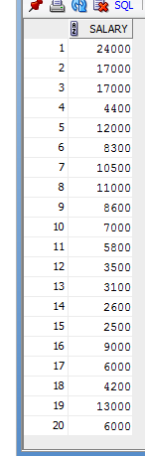
From examining the schema, we see that salary appears in two tables: employees and jobs. But the question asks for the salaries the company currently pays, in the present tense, so we’re looking for the salary employees receive rather than the minimum and maximum available for each job role. Therefore, the column we need is the salary column from the employees table.

Try the following query:

SELECT salary

FROM employees;

The result should be:



**NEW MATERIAL**

Notice that we have 20 results but that rows 2 and 3 are both 17000 and rows 17 and 20 are both 6000. A careful reading of the question shows that we’re looking for different salaries so we shouldn’t list 17000 twice or 6000 twice.

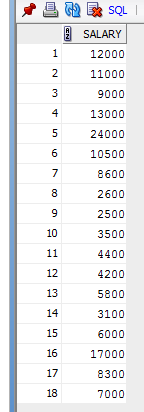
To solve this we use the special keyword: DISTINCT. It comes immediately after the SELECT keyword and requires the returned rows to be unique.

The final answer is therefore:

SELECT DISTINCT salary

FROM employees;

The results should be:

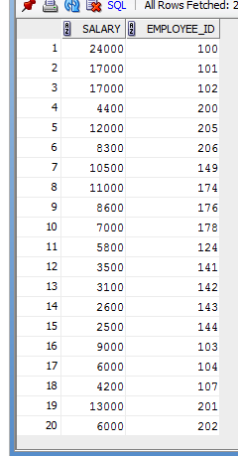


The DISTINCT keyword refers to the entire row and not to a single column. So try the following query:

SELECT DISTINCT salary, employee\_id

FROM employees;

You should get the following result:



Notice that again we have two rows with salary of 17000 and two with 6000. This is because the rows have different values for the employee\_id and so each row is unique.

Also note that Oracle has its own keyword UNIQUE that does the same as DISTINCT. You should not use it because, although it is identical, it is Oracle specific and it is good practice to use standard keywords and syntax wherever possible. This will reduce the amount of work needed to port a system from one back-end to another.

### Q5. The company wants to give a Christmas bonus to all employees of 5% of their annual salary. Produce a list of employee names (first and last name as one column) and their bonuses.

For this question, we will have to use a row function to calculate the bonus and to combine the first and last names. From the schema, we can see that all the columns we need are in the employees table and that we need the columns first\_name, last\_name and salary.

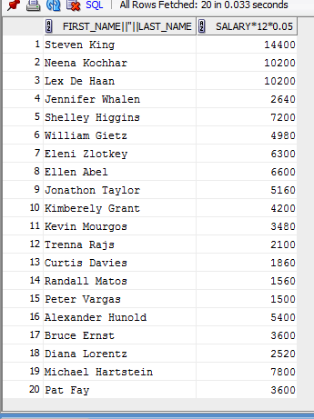
To combine the employees’ names we need to concatenate them. For this we could use the concatenate function (CONCAT(first\_name, last\_name)) but if we did this we would not have a space in between then names. To add the space we would have to use a second CONCAT function call around the first one which would quickly become messy. We therefore prefer the double pipes operator: first\_name || ‘ ‘ || last\_name.

To calculate the bonus, let’s assume that the salary listed is a monthly one and therefore we need to multiply it by 12 to get their annual salary. We then multiply the result by 5% (0.05) to find the bonus. This would be:

SELECT first\_name || ‘ ‘ || last\_name, salary\*12\*0.05

FROM employees;

The results would be:



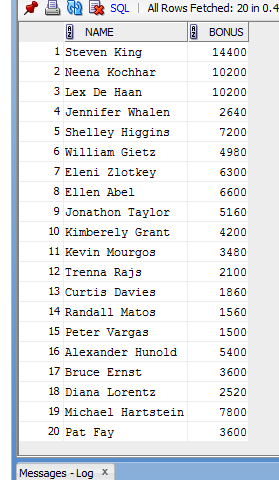
These results are technically correct but the column names are not readable because of the row functions. We should use aliases to provide clearer names. Our final query should therefore be:

SELECT first\_name || ‘ ‘ || last\_name AS name,

salary\*12\*0.05 AS bonus

FROM employees;

The results are then:



### Q6. For each job title, give the range of salary available.

Like many queries we might be asked this one is a little ambiguous – does it want the range of salaries that employees are actually receiving or the potential range? If we look at the employees table we see, for example, three programmers and they are paid different amounts (run SELECT \* FROM employees; and notice the salaries of 9000, 6000 and 4200). So is the question asking for the range of those value?

On the other hand, in the jobs table there are two columns called min\_salary and max\_salary and maybe the question is asking for the range based on those columns.

In this unit, we will endeavour to provide unambiguous questions wherever possible but in the real world you will sometimes be asked unclear questions and must either seek information or make judgement calls.

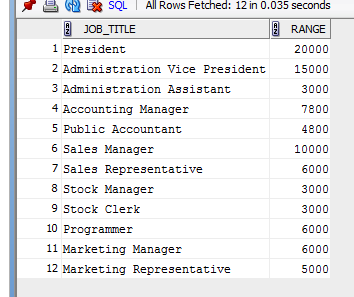
In this case, I would assume that because the question also asks for the job title that it is referring to the jobs table. This is because the jobs table has a column called job\_title whereas the employees table does not.

On that assumption we need to do some mathematics in the SELECT statement:

SELECT job\_title, max\_salary – min\_salary AS range

FROM jobs;

Notice the alias I have included to provide a more meaningful and easily understood column name. The results should be:



### Q7. For each employee, list the number of months they have been working for the company.

This question points us towards the employees table and specifically the hire\_date column. We can use the code we have used before to produce a nice column of employee names and we will need a row function to calculate how many months they have been working for the company.

Specifically, the function we need is MONTHS\_BETWEEN which takes in two dates and returns the number of months between the second date and the first. Note that the order of the dates is critical and the first date must be after the second. For our purposes we will use hire\_date as the second date and we use SYSDATE for the first date. SYSDATE is an Oracle reserved word that returns today’s date.

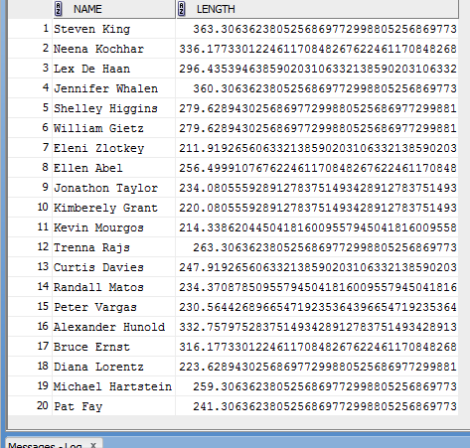
Try the following query:

SELECT first\_name || ' ' || last\_name AS name,

MONTHS\_BETWEEN(SYSDATE, hire\_date) AS length

FROM employees;

You should get the following results:



This doesn’t look that great because the MONTHS\_BETWEEN function returns lots of decimal points. To neaten it up we need to round off the answer to zero decimal places and we do this using the ROUND function. The input to the ROUND function is the number to be rounded and the number of decimal places to round off to. In this case we are not giving it an exact number to round but rather the result of the MONTHS\_BETWEEN function.

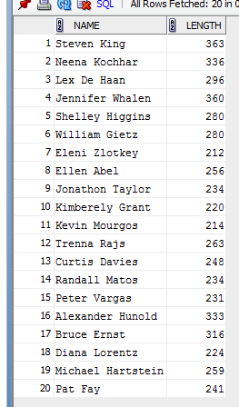
The final query is therefore:

SELECT first\_name || ' ' || last\_name AS name,

ROUND(MONTHS\_BETWEEN(SYSDATE, hire\_date),0) AS length

FROM employees;

The final result is:



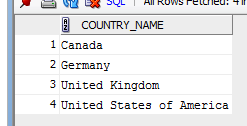
### Q8. List the names of all the countries the company operates in

This question directs us to the countries table which has a column called country\_name. The answer is fairly straightforward:

SELECT country\_name

FROM countries;

The results should be:



**NEW MATERIAL**

Sometimes you will make syntax mistakes or spelling mistakes in your SQL code. When you do this, SQL Developer will give an error and you will have to fix the mistake. Try the following SQL:

SELECT country\_nme

FROM countries;

Notice that the “a” in country\_name is missing. SQL Developer does not show the results, it shows an error message. On the top line it reports the error code, the keyword that contains the error and the type of error. In this case it is an ORA-00904 error in the word “country\_nme” and it is an “invalid identifier” error. This means that the word highlighted ought to be a column name or table name but that no such column or table exists.

Try the following query:

SELECT country\_name

FROM jobs;

Notice that the error message is the same even though the column name is spelt correctly. That is because there is no column called country\_name in the jobs table. So sometimes the “invalid identifier” error message indicates a spelling mistake but sometimes it means you’ve chosen the wrong table or column.

Other common error messages include:

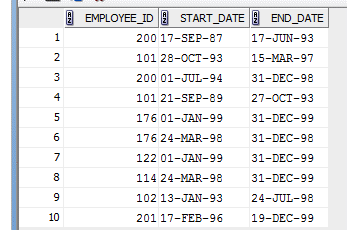
* “missing expression” – this typically indicates that you’ve put a comma at the end of the list of columns in the SELECT clause making SQL Developer think that the next word will be the name of a column. But instead you put a FROM clause or some other clause and not a column name.
* “FROM keyword not found where expected” – this indicates that you have forgotten to include the FROM clause.
* “Unknown Command” – this means you have probably misspelled SELECT at the beginning of the query so SQL Developer does not know what type of query you are trying to run.

There are other types of errors that you may encounter in the future as we learn new material. The key thing to remember is **read the error message**. Don’t panic that something is not working. The error message itself usually tells you exactly what the problem is and exactly where the problem is.

# Completion Problems: Write the SQL, with help

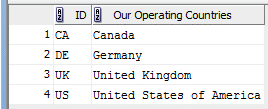
### Q1. List the ID, start date and end date of employees who have left the company.

A: SELECT EMPLOYEE\_ID, START\_DATE, END\_DATE

FROM job\_history;  


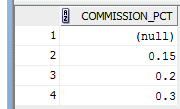
### Q2. List the ID and names of the countries the company works in. Use an appropriate alias.

A: SELECT COUNTRY\_ID AS ID, COUNTRY\_NAME AS “Our operating countries”  
FROM COUNTRIES;



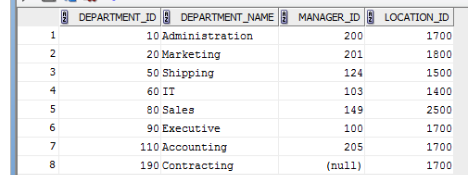
### Q3. List the different commission percentages for the employees.

A: SELECT DISTINCT commission\_pct   
FROM employees;

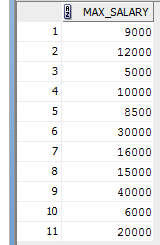


### Q4. List all the information in the Departments table.

A: SELECT \*  
FROM departments;

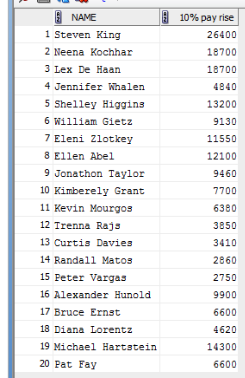


### Q5. List the different maximum salaries available.

A: SELECT DISTINCT max\_salary   
FROM jobs;  


### Q6. For every employee, calculate what their pay would be if they received a 10% pay rise.

A: SELECT first\_name || ' ' || last\_name AS name , salary\*1.1 AS “Pay raise”  
FROM employees;



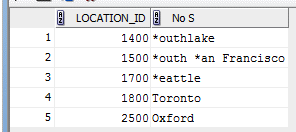
### Q7. For every employee, generate their email address by adding “@company.com”

A: SELECT CONCAT(LOWER(email), ‘@company.com’) AS “Complete email”  
FROM employees;



### Q8. List the locations and cities the company operates in but use an asterisk in place of capital ‘S’.

A: SELECT location\_id, REPLACE(City, ‘S’, ‘\*’)  
FROM locations;

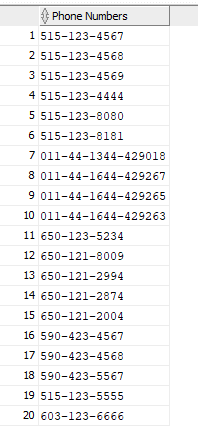


# Deliberate Practice: Write the SQL

### Q1. Give the phone numbers of employees but instead of full-stops to separate the groups of numbers, use dashes instead

SELECT REPLACE(Phone\_Number, ‘ .‘, ‘-‘)

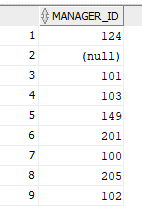
FROM employees;



### Q2. List the employee IDs of the managers

SELECT DISTINICT manager\_ID

FROM employees;

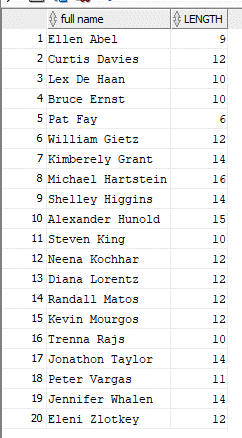


### Q3. Why should you not use a query like SELECT \* FROM employees; in production code?

### Q4. For each employee, give their full name and the number of letters in their name

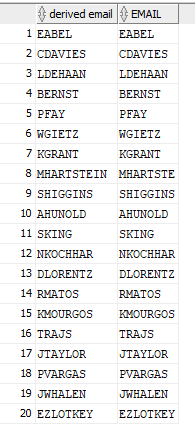
SELECT first\_name || ‘ ‘ || surname AS “Full name”, LENGTH(first\_name) + LENGTH(surname) AS Length

FROM employees;



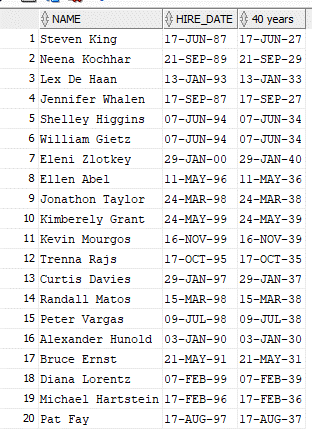
### Q5. Use the columns first\_name and last\_name to generate the values in the email address column

SELECT CONCAT(SUBSTR(first\_name, 1, 1), UPPER(last\_name)) AS email

FROM employees;

### Q6. For every employee list their full name, hire date and the date exactly 40 years after their initial employment

SELECT EXTRACT(DAY FROM HIRE\_DATE) || '-' || EXTRACT(MONTH FROM HIRE\_DATE)|| '-' || (EXTRACT(YEAR FROM HIRE\_DATE) + 40)

FROM EMPLOYEES;

SELECT first\_name || ' ' || last\_name AS name, hire\_date,   
 ADD\_MONTHS(hire\_date, 480) AS "40 years"   
FROM employees;

### Q7. For every job, list the title and the min and max salaries after first increasing them by 25%

SELECT JOB\_TITLE, min\_salary\*1.25 AS "min salary", max\_salary\*1.25 AS "max salary"

FROM jobs;