First key word is ‘SELECT’

‘SELECT’ by it self cannot do anything and with ‘SELECT’ we parse in some kind of columns-name of the attribute from a particular table.

But that doesn’t do anything, not yet. We have written the name of the column, but we need to specify which table are we calling that column from, and this is where we use the ‘FROM’ keyword. ‘FROM’ combined with the name of the table, gives us access to the table.

So in general our very simple query can look like this:

SELECT <column>

FROM <table>;

Which stands for, give me all the information of the column with name <column> from table with name <table>.

In SQL, if we wanted to print out all the columns of a table, instead of writing <column\_name1>, <column\_name2>, … , <column\_nameN>, we can write the star (\*), which stands for everything. The star (\*) jargon, with other symbols that we are going to see in the future, are called ‘Wildcards’.

SELECT \*

FROM <table>

Which stands for: Give me every column/everything from the table called <table>.

All these codes that we have given as an example, their jargon is ‘Queries’.

NOTE: SQL is not case sensitive. Meaning if we have written the names correctly, then the Query sent to the data base will be valid and will work.

‘DISTINCT’

-Let’s say we are printing only one column from a table. And that column has values, but some values are repeated. For example let’s say we have a column called ‘Position’ or ‘Occupation’ for some kind of ‘Employee’ table. Well it’s normal in real life to see employees have same occupations right? For example 2 people can be ‘sales person’, multiple people can be ‘engineers’ and so on and so forth. And if we wanted to get these occupations, but only once - our only goal is to see what kind of occupation a person can have, then we can use the ‘DISTINCT’ key word.

And ‘DISTINCT’ is written right after the ‘SELECT’ form and right before the column that we want to print out.

It will look something like this

SELECT DISTINCT <column>

FROM <table>

-We can further filter the queries or our tables that we are trying to print, by adding the keyword ‘WHERE’.

‘WHERE’ is a very powerful tool, that can work in many ways. Imagine if we have an if statement in other programming language that tells us: ‘if something is true then do this’. Well it’s how ‘WHERE’ works as well. For example if we ‘SELECT’ some column ‘FROM’ a table, then we can write some kind of condition that ‘WHERE’ is going to further filter our column, so that it matches the data that we want to get. For example – Let’s say we have a table of employees. And we want to select the column of the ‘positions’ of the employees. So for example we can have ‘manager’, ‘boss’, etc. And we can write WHERE position = ‘manager’. And that is going to print us only the rows that have ‘manager’ as part of their position. Another example is that let’s say we select everything from the table table employees. And there is a column of age. We can select all the employees by simply writing age >= <the\_age>.

Both examples are going to look like this:

SELECT position

FROM employees

WHERE position = ‘manager’;

SELECT \*

FROM employees

WHERE age >= <some\_age>

-It’s important to remember that the keywords SELECT, FROM, WHERE and many more in the future that we are going to see, are case insensitive. Meaning that it doesn’t matter if we write ‘SELECT’ or ‘select’ and etc. But if we want to compare a string in ‘WHERE’ it’s IMPORANT that we write it the EXACT way that it is written in the Data Base. Otherwise it won’t work. VERY IMPORTANT. So in that case when we compare things, it is CASE SENSETIVE.

So TLDR: key words are case insensitive but comparing data is case sensitive.

-‘WHERE’ can be combined with multiple statements with ‘AND’ and ‘OR’ for further filtering.

- Bute let’s say we have a lot of ORs. For example we have something like:

SELECT …

FROM …

WHERE <cond 1>

OR <cond 2>

OR <cond 3>

…

OR <cond n>

Well instead of having it like that, we can write it more compact with the key form ‘IN’. It will basically take an argument, in our case the column name, and it will check if the there is value, that matches in the ‘IN’ “list”. So it will look like this:

SELECT …

FROM …

WHERE <something> IN (<cond 1>, <cond 2>, … , <cond n>).

What is powerful is that we can write ‘NOT IN’ and then it will print us the rows after filtering the column, to take out the elements not in the list.

SELECT …

FROM …

WHERE <something> NOT IN (<cond 1>, <cond 2>, … , <condn>).

-‘BETWEEN’ is also a powerful key word/operator and it gives filters the data that are between two values.

SELECT …

FROM …

WHERE <something> BETWEEN val1 and val2

‘BETWEEN’ includes val1 and val2 as well!

And we can still use ‘NOT’ with ‘BETWEEN’ which will filter the columns that are not between val1 and val2.

-‘IN’ can work with all types of data, ‘BETWEEN’ work with numeric and date types.

Just as in other programming languages we can write “not” something to our conditional statement. In SQL it is done by writing

“’WHERE’ != …”

And it’s important that we write the keywords are correctly written in the correct order. Meaning that we write:

SELECT …

FROM …

WHERE …

AND/OR …

More notes -> SELECT-ing a column FROM a table, we need to write the correct column name, or else it will give us error.

Order of how the interpretator reads the SQL code:

5) SELECT {\* | column | expression

[alias],..}

1) FROM table

2) WHERE <condition>

3) GROUP BY <column>

4) HAVING <condition>

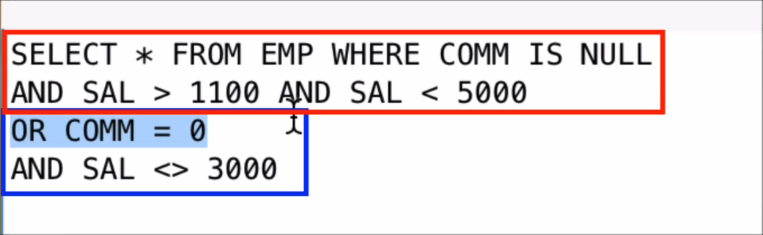
6) ORDER BY <column>;

We are going to see the other keywords from this order later on.

- ‘null’ are empty cells in the columns. Usually in Data Base null cells indicate that the row has no such value of the attribute. We can also check if a certain column has null cells by asking:

‘WHERE’ <column> ‘IS’ null.

Something very interesting:



Is different from:



‘OR’ statement divides logically the ‘AND’s.

To avoid any errors from misplacing the statements, we can just add brackets and that will logically help but also help us to read the code of what is happening.

-‘LIKE’ is a more powerful tool to compare and filter string cells. Rather using “=”, where it searches the exact match, we can use ‘LIKE’ where we can parse a substring and find the cells that contain that substring. It is especially very good, when combined with wild cards- “%” and “\_”.

For example if we have a column of “JOBS” and we want to find all the jobs that start with “S”, we can use the “WHERE JOB LIKE ‘S%’. % works, like it can fill with random number of letters and random letters. “\_” on the other side fills in with only one letter.

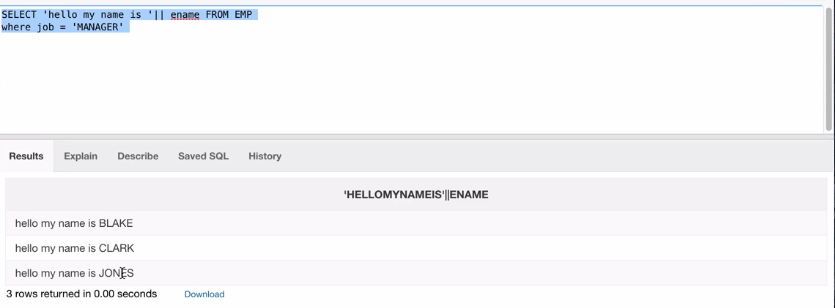
-Aliasing: An alias is basically a nickname for the columns. For example if we have a Data Base for employees, we can have columns/attributes, that are obscure or confusing for new readers. Such as ENAME, SAL, COMM. So we can use aliasing like this:

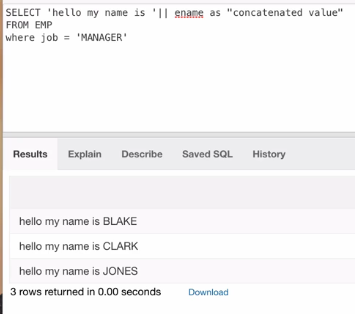
SELECT ENAME as Employee\_name, SAL as Salary, COM as Commission.

…

So we can see that we use the key work ‘AS’. And printing the code above will print out the names of the columns with the nicknames that we have given.

-Concatenation: they are done with “||”



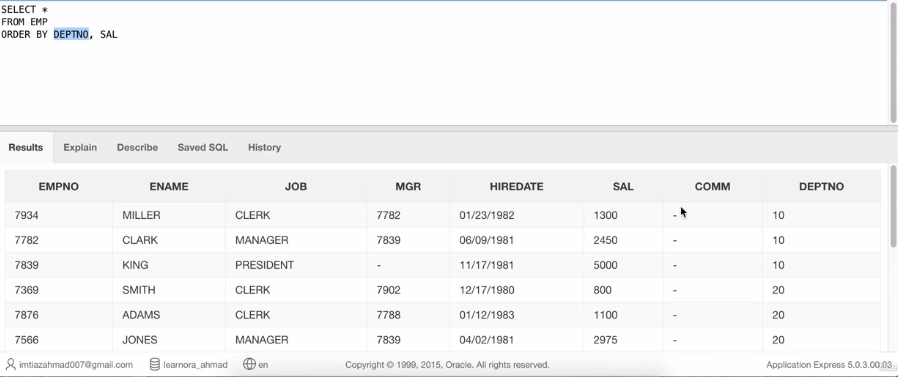


-‘ORDER BY’ sorts based on the column that we have specified.

If we specify a column that has strings it will order it alphabetically. If we specify a column that has numeric then it will order numerically.

And the order by default will be ascendingly. If we want descendingly we would write the special key work ‘DESC’ after we specified the column. We can also order by two columns. How it will work is that it will order the first column that we specified and then it will order the second column.

The example bellow will show order by two columns.



Something interesting:

