Sub-queries or sub-selects are like regular queries but placed within paranthesis and nested inside another query. This allows you to form more powerful queries than would have been otherwise possible.

Consider the EMPLOYEES table from the previous lesson. Lets say we want to retrieve the list of employees who earn more than the average salary. To do so we could try:

```
select * from employees where salary > avg(salary)
```

However running this query will result in an error like:

```
SQL0120N   Invalid use of an aggregate function or OLAP function.SQLCODE=-120,

SQLSTATE=42903
```

One of the limitations of built-in aggregate functions like AVG() is that they cannot always be evaluated in the WHERE clause.

So to evaluate the AVG() function in the where clause we can make use of a sub-Select expression like the follows:

```
select EMP_ID, F_NAME, L_NAME, SALARY from employees

 where SALARY <

 (select AVG(SALARY) from employees);
```

EMP_ID	F_NAME	L_NAME	SALARY
E1008	Bharath	Gupta	65000.00
E1007	Mary	Thomas	65000.00
E1004	Santosh	Kumar	60000.00
E1003	Steve	Wells	50000.00

The IN operator can also be used and there can be multiple leves of sub-queries, such as:

```
select EMP_ID, F_NAME, L_NAME, DEP_ID from employees

 where DEP_ID IN

 ( select DEP_ID from employees where DEP_ID >

         ( select MIN(DEP_ID) from employees )

 );
```

The sub-select doesn't just have to go in the where clause, it can also go in other parts of the query such as in the list of columns to be selected:

```
select EMP_ID, SALARY,

  ( select AVG(SALARY) from employees )

  AS AVG_SALARY

  from employees ;
```

Another option is to make the sub-query be part of the FROM clause:

```
select * from

 ( select EMP_ID, F_NAME, L_NAME, DEP_ID

 from employees) ;
```

In this lesson you have seen how sub-queries and nested queries can be used to form more richer queries and overcome some of the limitations of aggregate functions.