**Inline view**

An **inline view** is a [**SELECT**](https://www.1keydata.com/sql/sqlselect.html) statement in the **FROM** clause. As mentioned in the [**View**](https://www.1keydata.com/sql/sql-view.html) section, a view is a virtual table that has the characteristics of a table yet does not hold any actual data. In an inline view construct, instead of specifying table name(s) after the **FROM** keyword, the source of the data actually comes from the inline view.

**Inline view** is sometimes referred to as **derived table**. These two terms are used interchangeably.

**Syntax**

The syntax for an inline view is,

**SELECT "column\_name" FROM (Inline View);**

**Example**

Assume we have two tables: The first table is ***User\_Address***, which maps each user to a ZIP code; the second table is ***User\_Score***, which records all the scores of each user. The question is, how to write a SQL query to find the number of users who scored higher than 200 for each ZIP code?

Without using an inline view, we can accomplish this in two steps:

Query 1

**CREATE TABLE User\_Higher\_Than\_200   
SELECT User\_ID, SUM(Score) FROM User\_Score   
GROUP BY User\_ID   
HAVING SUM(Score) > 200;**

Query 2

**SELECT a2.ZIP\_CODE, COUNT(a1.User\_ID)   
FROM User\_Higher\_Than\_200 a1, User\_Address a2   
WHERE a1.User\_ID = a2.User\_ID   
GROUP BY a2.ZIP\_CODE;**

In the above code, we introduced a temporary table, ***User\_Higher\_Than\_200***, to store the list of users who scored higher than 200. ***User\_Higher\_Than\_200*** is then used to [**join**](https://www.1keydata.com/sql/sqljoins.html) to the ***User\_Address*** table to get the final result.

We can simplify the above SQL using the inline view construct as follows:

Query 3

**SELECT a2.ZIP\_CODE, COUNT(a1.User\_ID)   
FROM   
(SELECT User\_ID, SUM(Score) FROM User\_Score GROUP BY User\_ID HAVING SUM(Score) > 200) a1,   
User\_Address a2   
WHERE a1.User\_ID = a2.User\_ID   
GROUP BY a2.ZIP\_CODE;**

The code that is in **red** represents an inline view. There are two advantages on using inline view here:

1. We do not need to create the temporary table. This prevents the database from having too many objects, which is a good thing as each additional object in the database costs resources to manage.

2. We can use a single SQL query to accomplish what we want.

Notice that we treat the inline view exactly the same as we treat a table. Comparing Query 2 and Query 3, we see that the only difference is that we replace the temporary table name in Query 2 with the inline view statement in Query 3. Everything else stays the same.

**ROWNUM:**

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on.

You can use ROWNUM to limit the number of rows returned by a query, as in this example:

SELECT \* FROM employees WHERE ROWNUM < 10;

If an ORDER BY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDER BY clause. The results can vary depending on the way the rows are accessed. For example, if the ORDER BY clause causes Oracle to use an index to access the data, then Oracle may retrieve the rows in a different order than without the index. Therefore, the following statement will not have the same effect as the preceding example:

SELECT \* FROM employees WHERE ROWNUM < 11 ORDER BY last\_name;

If you embed the ORDER BY clause in a subquery and place the ROWNUM condition in the top-level query, then you can force the ROWNUM condition to be applied after the ordering of the rows. For example, the following query returns the employees with the 10 smallest employee numbers. This is sometimes referred to as top-N reporting:

SELECT \* FROM

(SELECT \* FROM employees ORDER BY employee\_id)

WHERE ROWNUM < 11;

In the preceding example, the ROWNUM values are those of the top-level SELECT statement, so they are generated after the rows have already been ordered by employee\_id in the subquery.

Conditions testing for ROWNUM values greater than a positive integer are always false. For example, this query returns no rows:

SELECT \* FROM employees

WHERE ROWNUM > 1;

The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

You can also use ROWNUM to assign unique values to each row of a table, as in this example:

UPDATE my\_table

SET column1 = ROWNUM;