SIM GLOBAL EDUCATION

UNIVERSITY OF WOLLONGONG AUSTRALIA

CSCI317 – Database Performance Tuning

Tutorial

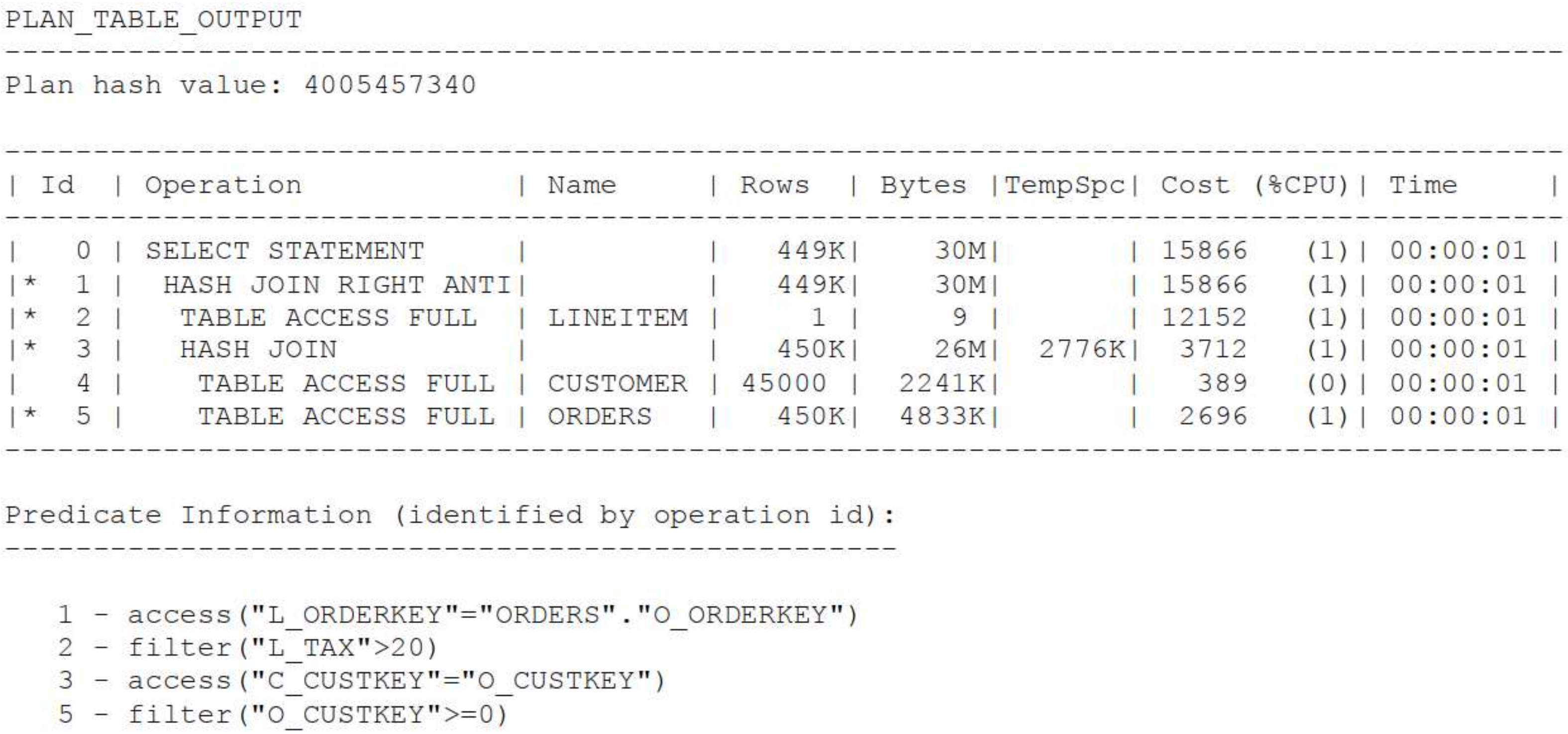
sjapit@uow.edu.au

17 August 2022

Query Processing Plans

Interpretation of query processing plans

Consider the following conceptual schema of a two-dimensional data cube.



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 4005457340

------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |  449K|    30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|          |  449K|    30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL   | LINEITEM |     1 |     9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN           |          |  450K|    26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL  | CUSTOMER | 45000 |  2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL  | ORDERS   |  450K|  4833K|       |  2696   (1)| 00:00:01 |
------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("L_ORDERKEY"="ORDERS"."O_ORDERKEY")
   2 - filter("L_TAX">20)
   3 - access("C_CUSTKEY"="O_CUSTKEY")
   5 - filter("O_CUSTKEY">=0)
```

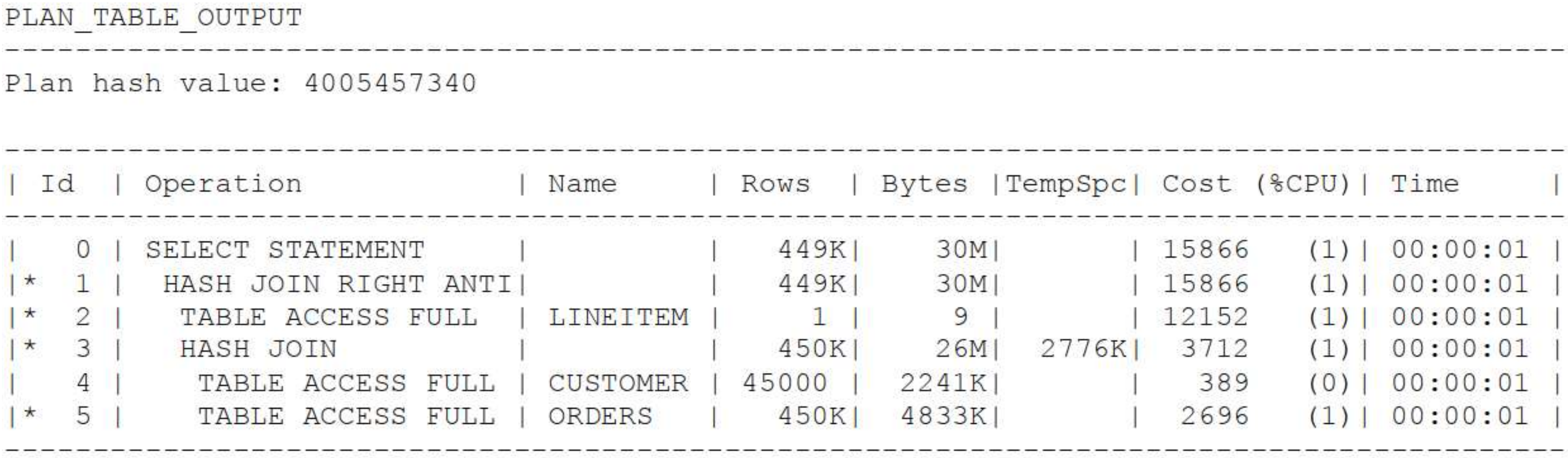
Interpretation of query processing plans

Find and draw a syntax tree of a query processing plan listed above and discover a respective SELECT statement that may have such query processing plan.

Query Processing Plan Explained:

- An execution plan shows the detailed steps necessary to execute a SQL statement.
- These steps are expressed as a set of database operators that consume and produce rows.
- The order of the operators and their implementations is decided by the query optimizer using a combination of query transformations and physical optimization techniques.
- While the display is commonly shown in a tabular format, the plan is in fact tree-shaped.

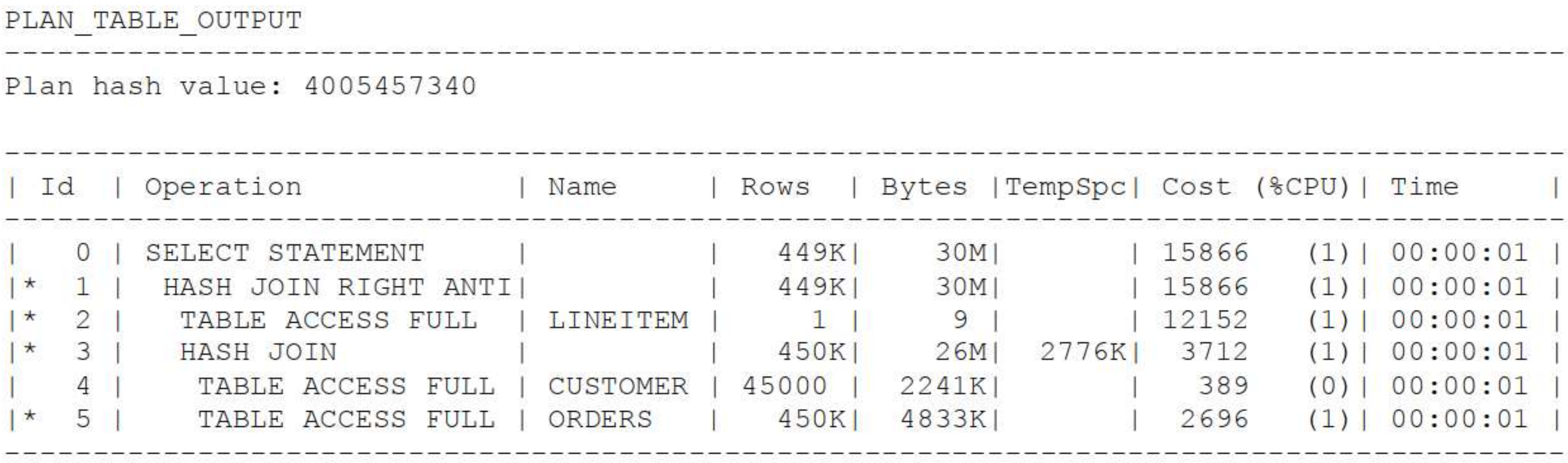
For example:



```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------
Plan hash value: 4005457340

-----------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |  449K|    30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|          |  449K|    30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL   | LINEITEM |     1 |     9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN           |          |  450K|    26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL  | CUSTOMER | 45000 |  2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL  | ORDERS   |  450K|  4833K|       |  2696   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

- The tabular representation is a top-down, left-to-right traversal of the execution tree.
- When you read a plan tree you should start from the inner most indented lines (also known as the leaf nodes of the execution tree) and work across and follows with next outer indented lines.

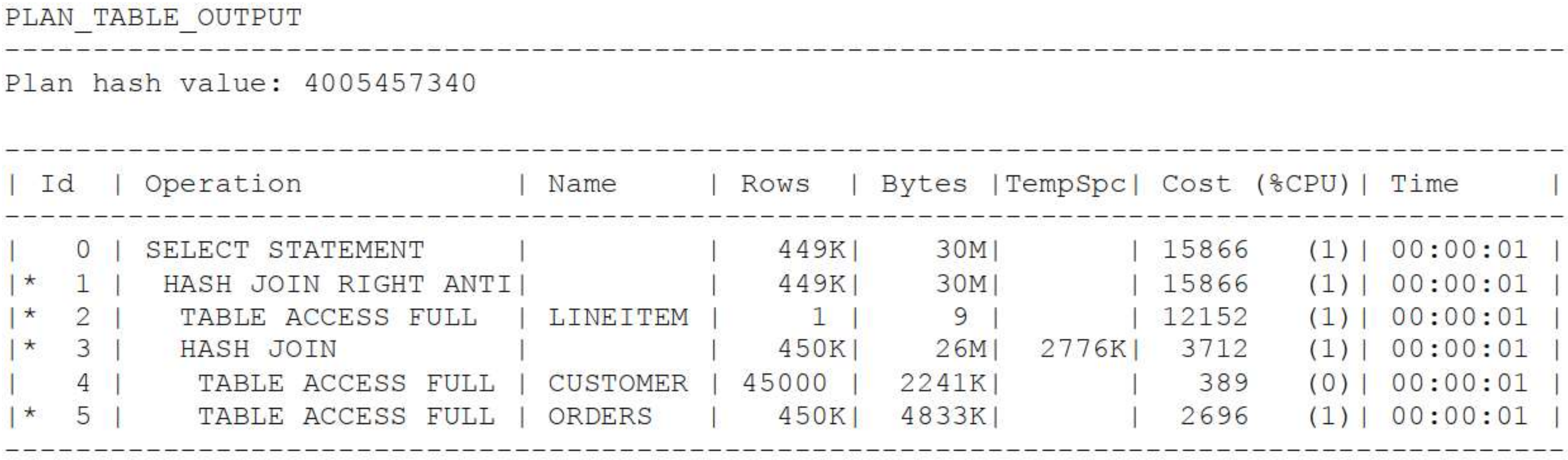
For example:



```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------
Plan hash value: 4005457340

-----------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|          |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL  | LINEITEM |     1 |     9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN          |          |   450K|    26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL | CUSTOMER | 45000 |  2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL | ORDERS   |   450K|  4833K|       |  2696   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

- In the above example, begin by looking at the leaves of the tree. In this case the leaves of the tree are implemented using a full table scans of the ORDERS and the CUSTOMER tables.

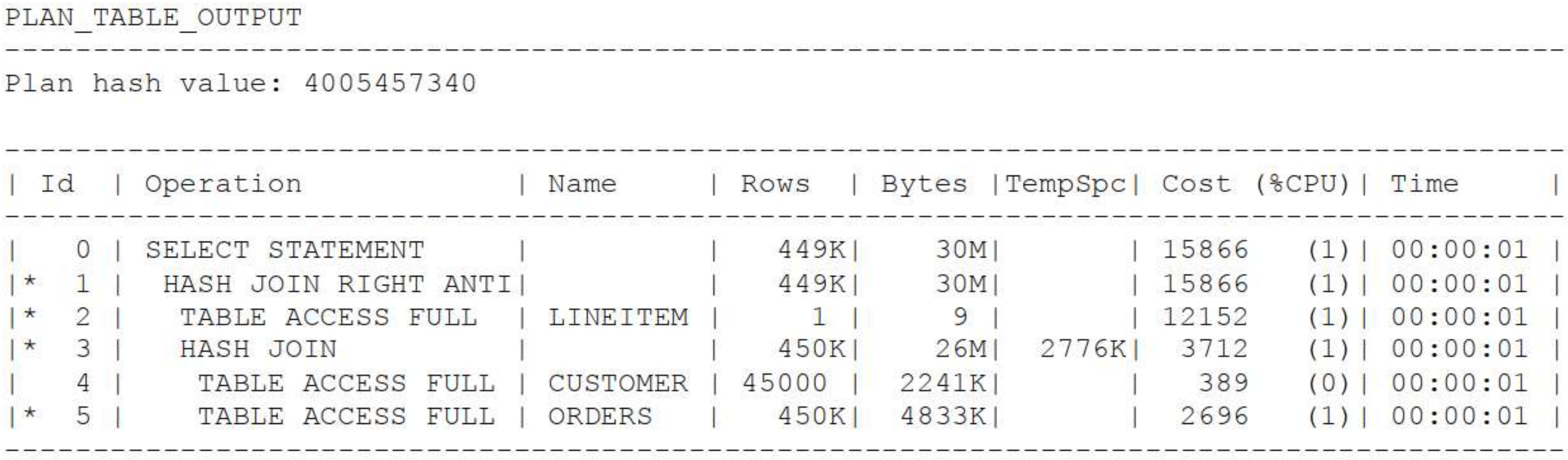
For example:



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 4005457340

--------------------------------------------------------------------------------------------
| Id  | Operation               | Name       | Rows   | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |            |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|            |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL    | LINEITEM |     1 |     9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN             |            |   450K|    26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL   | CUSTOMER | 45000 |  2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL   | ORDERS     |   450K|  4833K|       |  2696   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------
```

- The rows produced by these table scans will be consumed by the join operator. Here the join operator is a hash-join (other alternatives include nested-loop or sort-merge join).

For example:



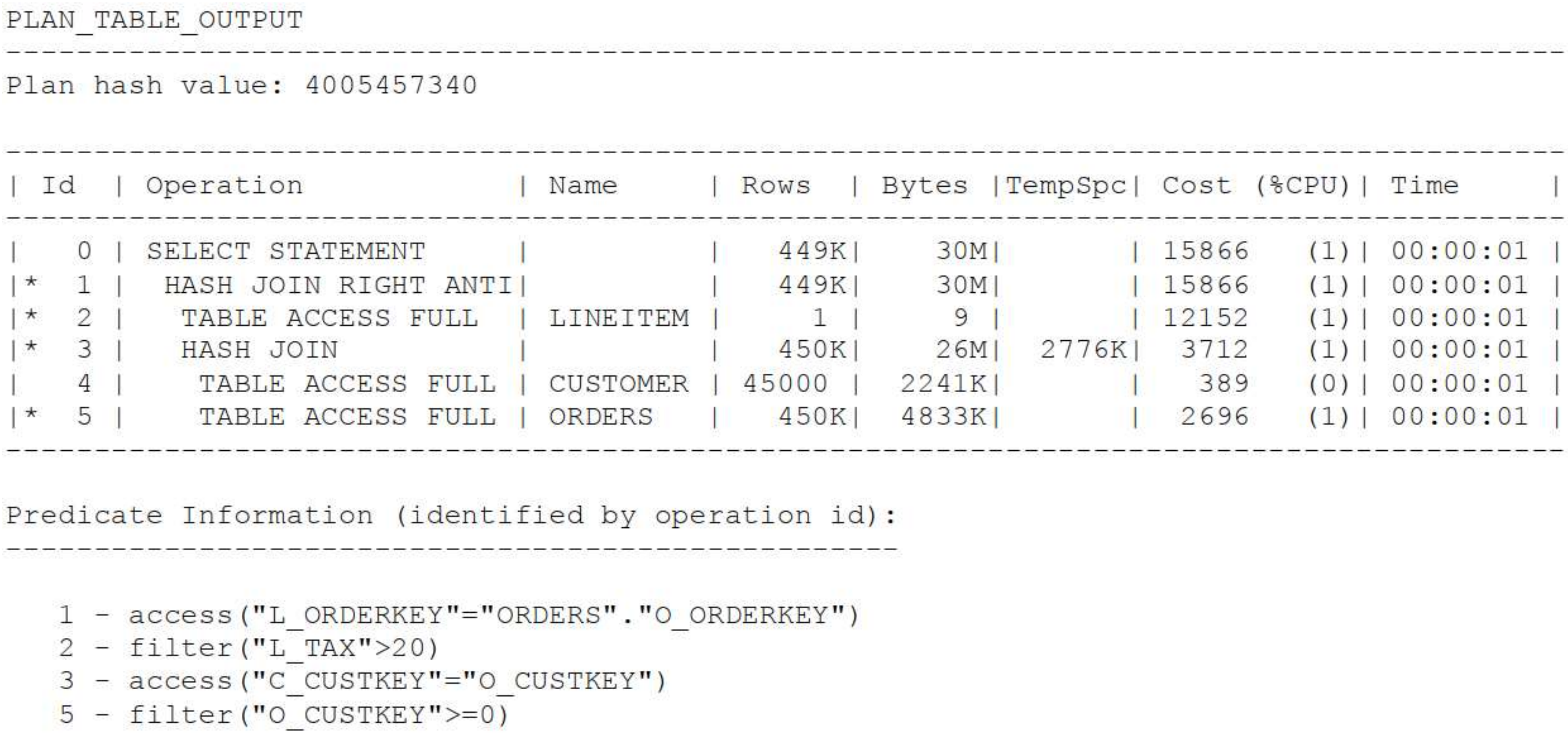
```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 4005457340

-------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows   | Bytes  |TempSpc| Cost  (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |    449K|     30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|           |    449K|     30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL  | LINEITEM |      1 |      9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN            |           |    450K|     26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL | CUSTOMER | 45000 |   2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL | ORDERS    |    450K|   4833K|       |  2696   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

- Finally the result of the hash-join is hash-join with rows produced by a full-table scan of the table LINEITEM using right-anti-join.
- The final result is then sent to the end user.

Query Processing Plans

Drawing the syntax tree

Drawing a syntax tree



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 4005457340

--------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI|           |   449K|    30M|       | 15866   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL  | LINEITEM  |     1 |     9 |       | 12152   (1)| 00:00:01 |
|*  3 |   HASH JOIN          |           |   450K|    26M|  2776K|  3712   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL | CUSTOMER  | 45000 |  2241K|       |   389   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL | ORDERS    |   450K|  4833K|       |  2696   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("L_ORDERKEY"="ORDERS"."O_ORDERKEY")
   2 - filter("L_TAX">20)
   3 - access("C_CUSTKEY"="O_CUSTKEY")
   5 - filter("O_CUSTKEY">=0)
```

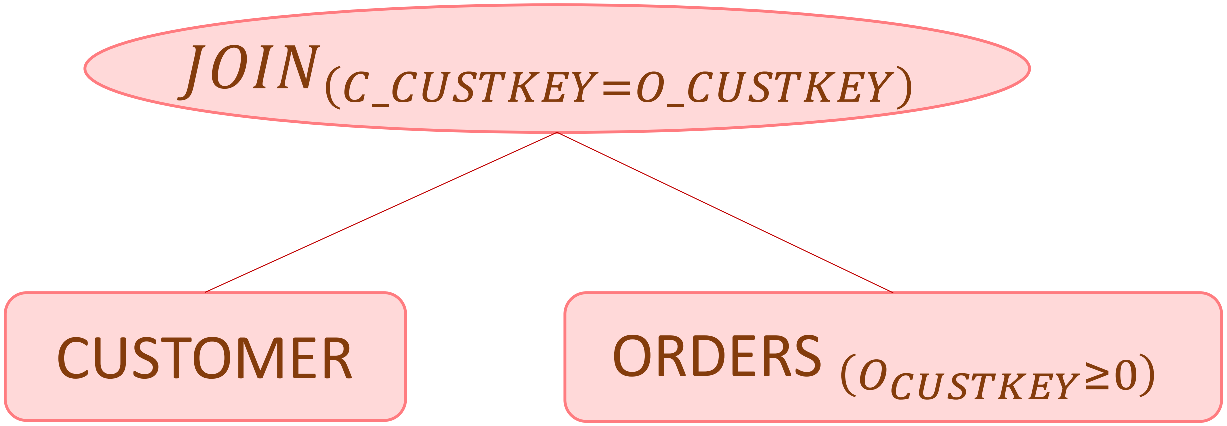
Starting from the inner most indented lines (in this example, lines 4 and 5, draw a leaf node for each line.

- Starting from the inner most indented lines (in this example, lines 4 and 5, draw a leaf node for each line.
- Indicate any filter condition if exist.

CUSTOMER

ORDERS $(O_{CUSTKEY} \geq 0)$

- Next draw the join operation.
- Indicate the join condition.

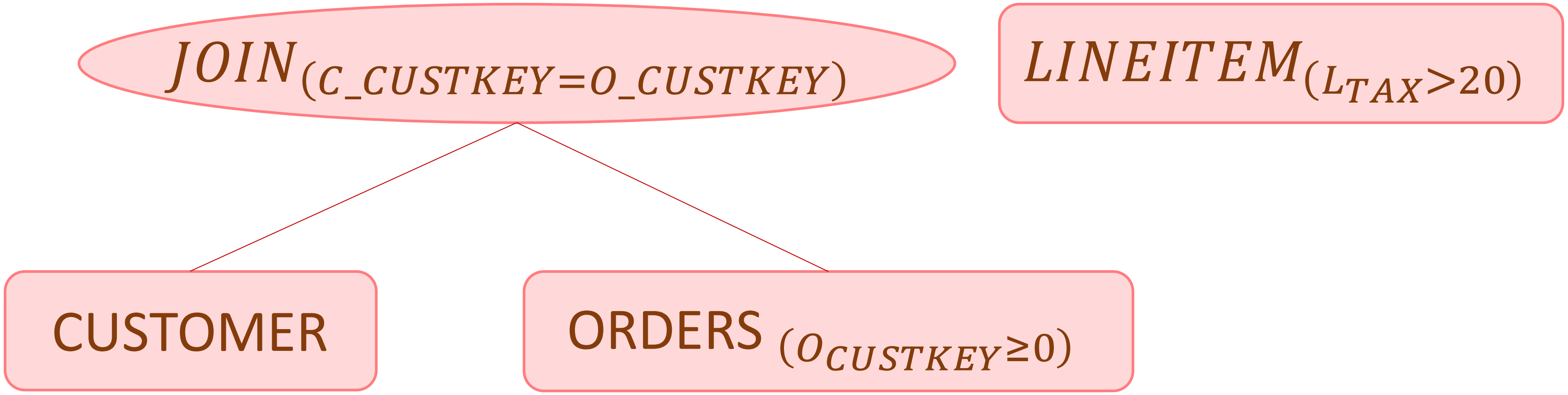


$JOIN_{(C\_CUSTKEY=O\_CUSTKEY)}$

CUSTOMER

ORDERS $(O_{CUSTKEY}\geq 0)$

- Fall all operations that have the same indentation level as the join operator, draw a node for each of the operation.
- Indicate the filter condition if exist.



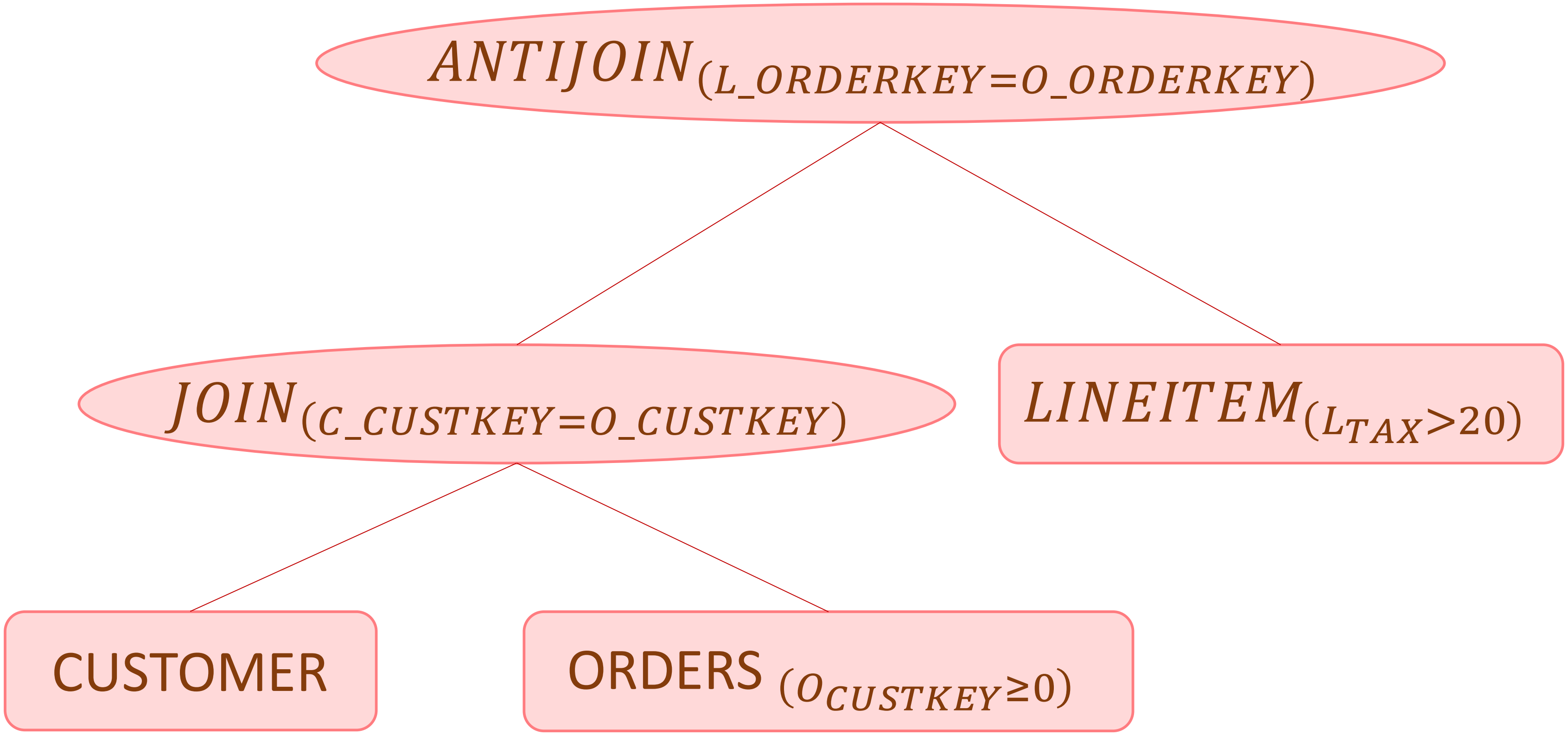
$JOIN_{(C\_CUSTKEY=O\_CUSTKEY)}$

$LINEITEM_{(L_{TAX}>20)}$

CUSTOMER

ORDERS $(O_{CUSTKEY} \geq 0)$

- Next draw the join operation.
- Indicate the join condition.



$ANTIJOIN_{(L\_ORDERKEY=O\_ORDERKEY)}$

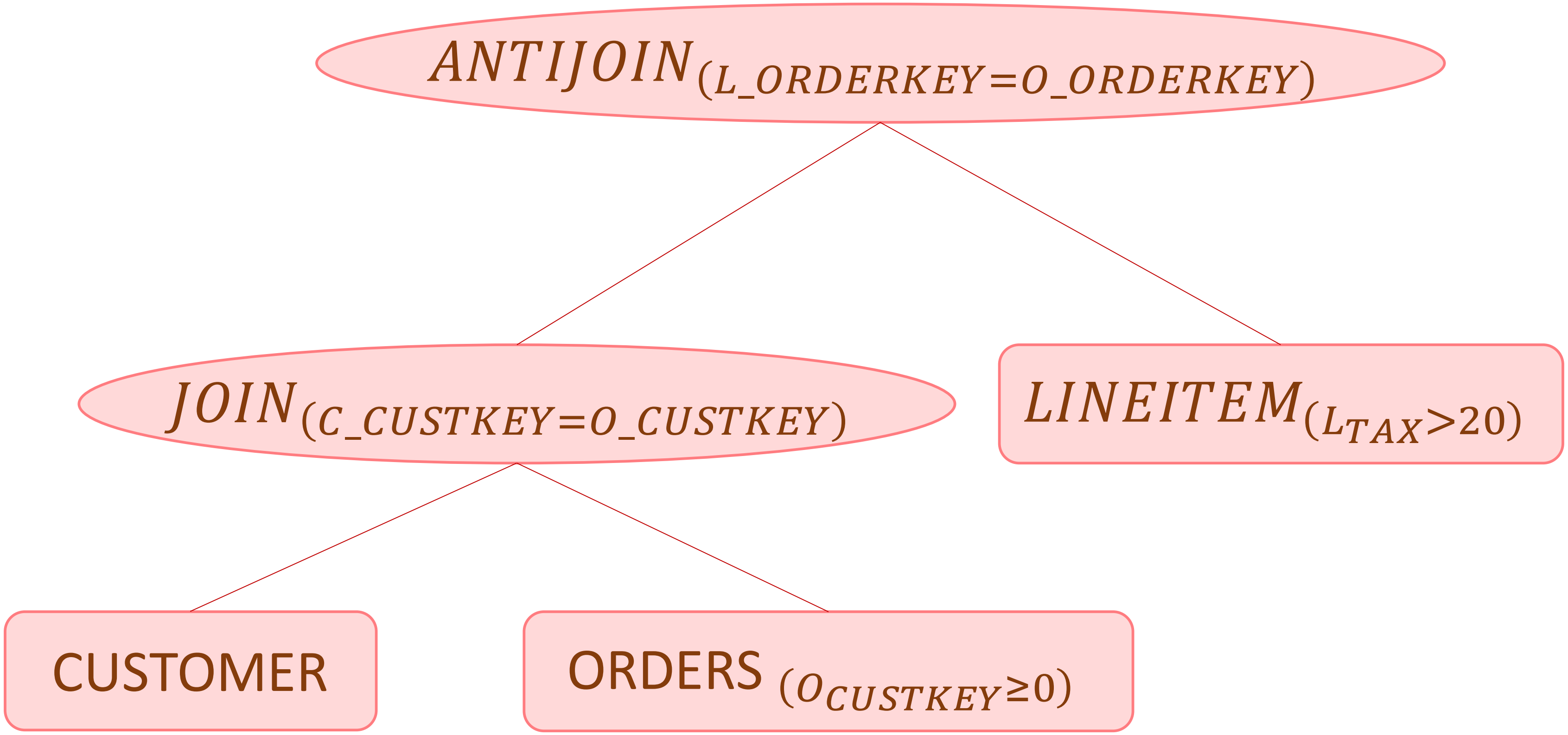
$JOIN_{(C\_CUSTKEY=O\_CUSTKEY)}$

$LINEITEM_{(L_{TAX}>20)}$

CUSTOMER

ORDERS $_{(O_{CUSTKEY}\geq 0)}$

Stop when reach to the top (outer most indentation.)



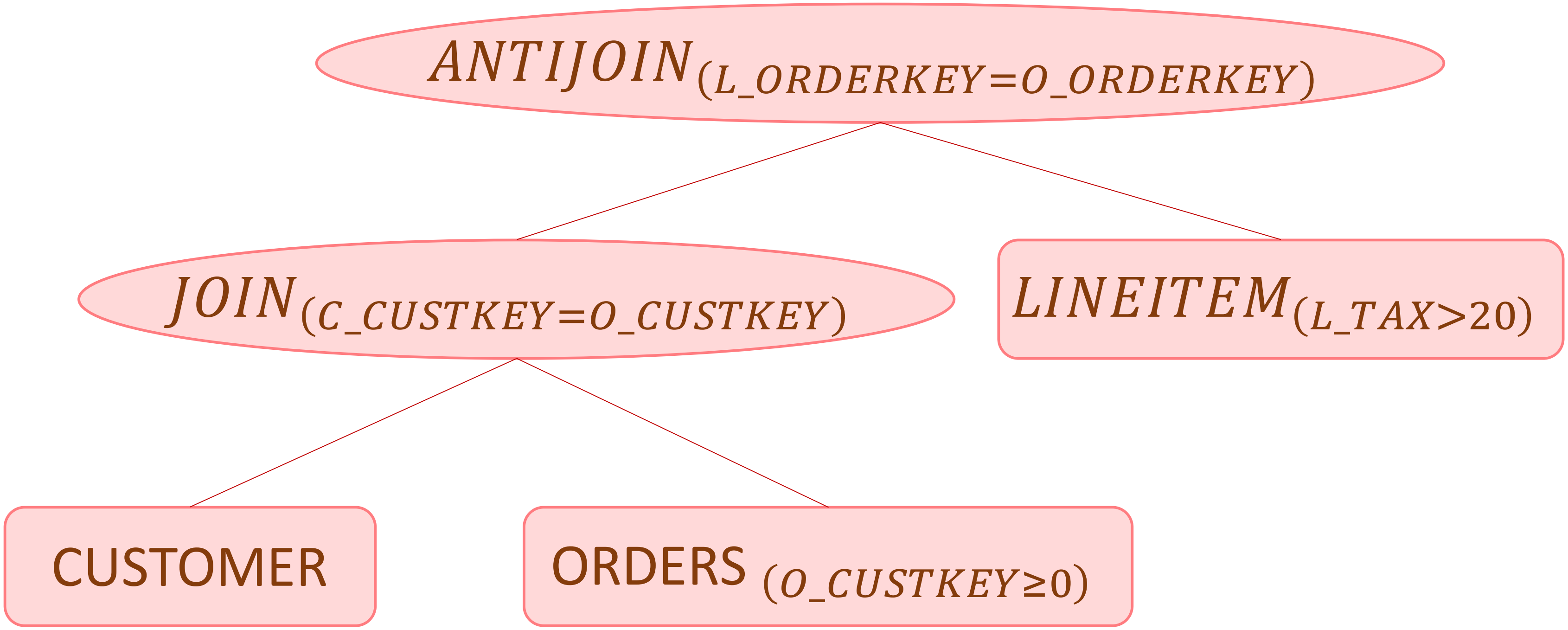
$ANTIJOIN_{(L\_ORDERKEY=O\_ORDERKEY)}$

$JOIN_{(C\_CUSTKEY=O\_CUSTKEY)}$

$LINEITEM_{(L_{TAX}>20)}$

CUSTOMER

ORDERS $_{(O_{CUSTKEY}\geq 0)}$



$ANTIJOIN_{(L\_ORDERKEY=O\_ORDERKEY)}$

$JOIN_{(C\_CUSTKEY=O\_CUSTKEY)}$

$LINEITEM_{(L\_TAX>20)}$

CUSTOMER

ORDERS $_{(O\_CUSTKEY\geq 0)}$

```
SELECT *
FROM ORDERS JOIN CUSTOMER
        ON C_CUSTKEY = O_CUSTKEY
WHERE O_ORDERKEY NOT IN (SELECT L_ORDERKEY
                         FROM LINEITEM
                         WHERE L_TAX > 20)
```