

PUG
CHALLENGE
EXCHANGE
AMERICAS

OpenEdge SQL Performance
Troubleshooting and Understanding Query Plans

26/06/2016

Chandra Sekhar

Mohd Sayeed Akthar

Steven Pittman

Santhosh Kumar Duvvuru

Contents

Executive Summary.....	3
Preface	3
How to use this guide?.....	3
Prerequisite.....	3
Chapter 1 –Why are we here?	3
Chapter 2 – Generating query plans	4
Chapter 3 – Understanding different query operations from Query plan.....	13
Chapter 4 - Table Scan Vs. Index scan.....	24
Chapter 5 - Join Methods.....	30
Chapter 6 - Aggregations	39
Chapter 7 - What happened to my subquery?	44
Chapter 8 - Importance of Statistics	50
Chapter 9 – What goes into Statistics?	74
Chapter 10 - SQL hints	86
Chapter 11 – Puzzles.....	93

Executive Summary

In this workshop we will take you through step by step interactive activities where you understand Query plans generated by OpenEdge SQL and demonstrate how important it is to understand query plans while working on performance related problems. In this Workshop you will learn

- Generating and understanding Query plans
- Identifying bad queries which are causing performance degradation
- Importance of statistics
- Troubleshooting common performance problems
- Solving problems using the query plan

Preface

This guide is designed to quickly introduce you to the steps involved in identifying and fixing OE SQL performance problems using Query plans and understand different types of statistics and their role in helping OE SQL optimizer to generate better query plans.

How to use this guide?

This guide is designed to flow from one chapter to the next, because each chapter requires knowledge built in previous chapters. You may decide to skip few chapters if you are aware of the concepts present those chapters. Throughout this workshop, we will be using JDBC client Squirrel for execution of queries.

Prerequisite

This workshop requires basic knowledge on SQL.

Chapter 1 –Why are we here?

Having knowledge of concepts presented in this workshop would really make a big difference when you deal with OE SQL in day to day life. Most of the time in this workshop, you will be working on your machine, understanding query plans and solving performance problems.

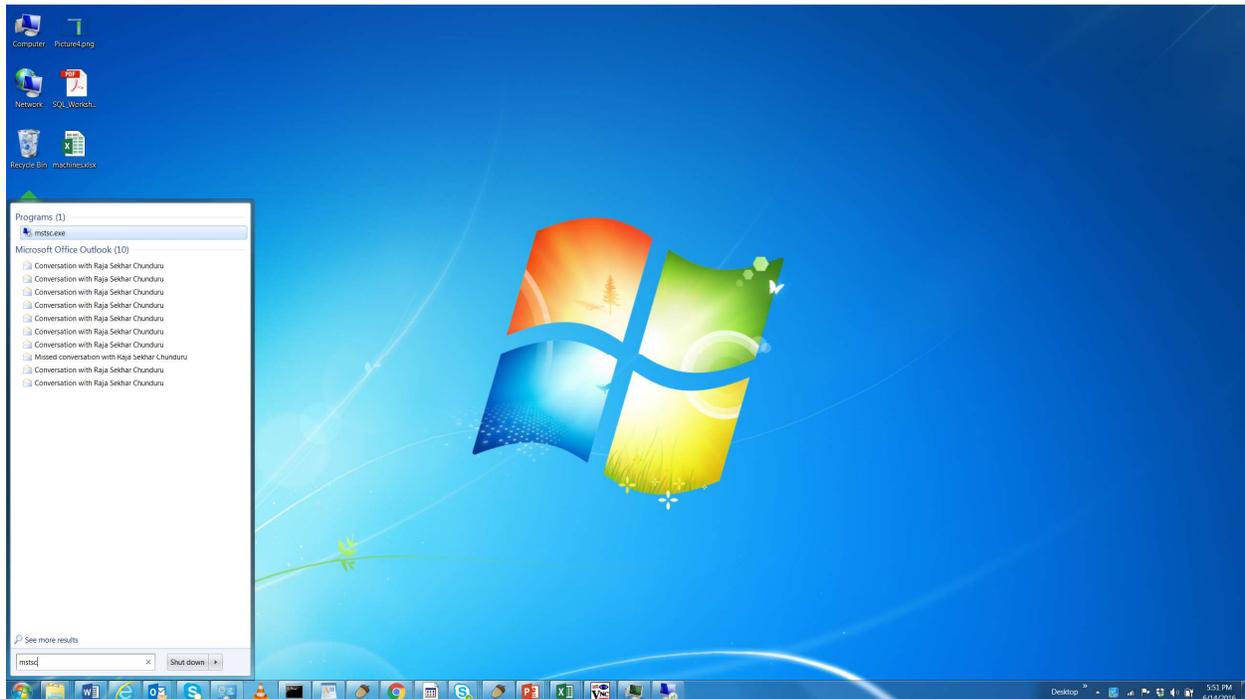
Chapter 2 – Generating query plans

In this chapter, you will learn different ways to generate Query plan. Query plan is the execution strategy determined by SQL optimizer to complete the query. For example, for a query like below, one of query plan can be “Perform the Index scan of CustIdx on table pub.customer with custnum value as 10 and return the custname as the result”.

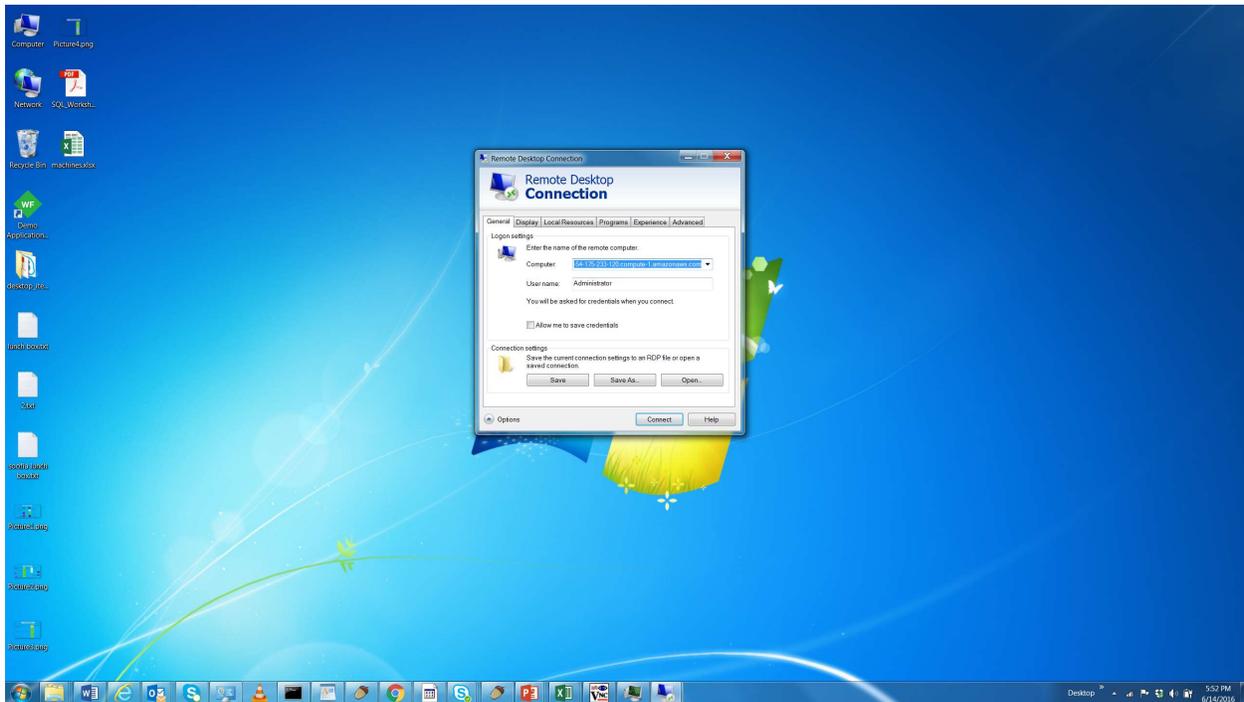
```
SELECT CustName FROM Pub.Customer WHERE CustNum=10;
```

Connecting to VM:

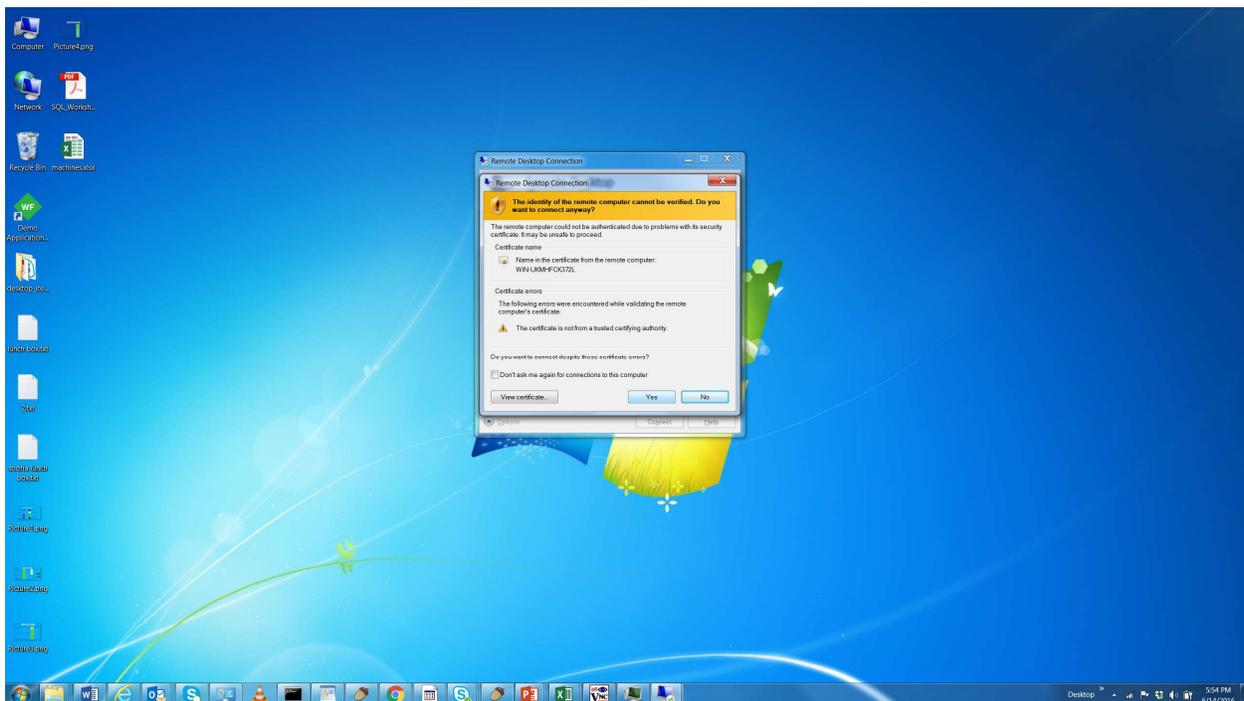
Search for “mstsc” and select.



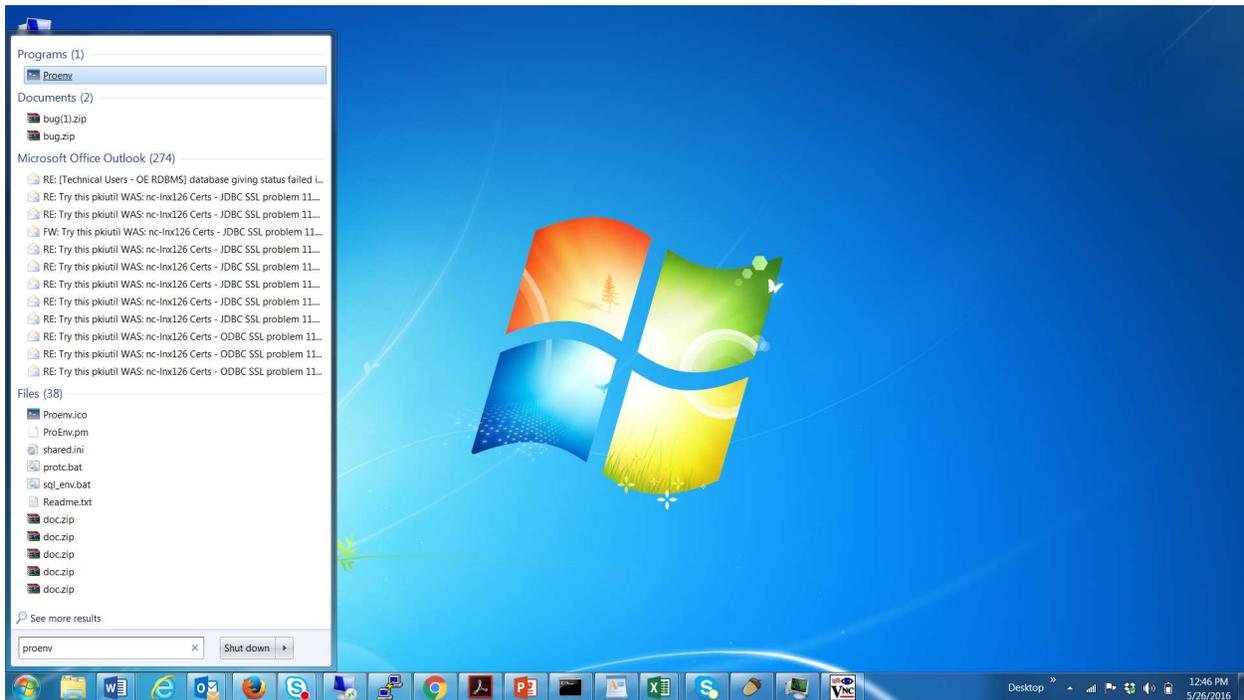
Enter the details of machine, user name and password which is allotted to you and click on connect.



Click on "Yes".

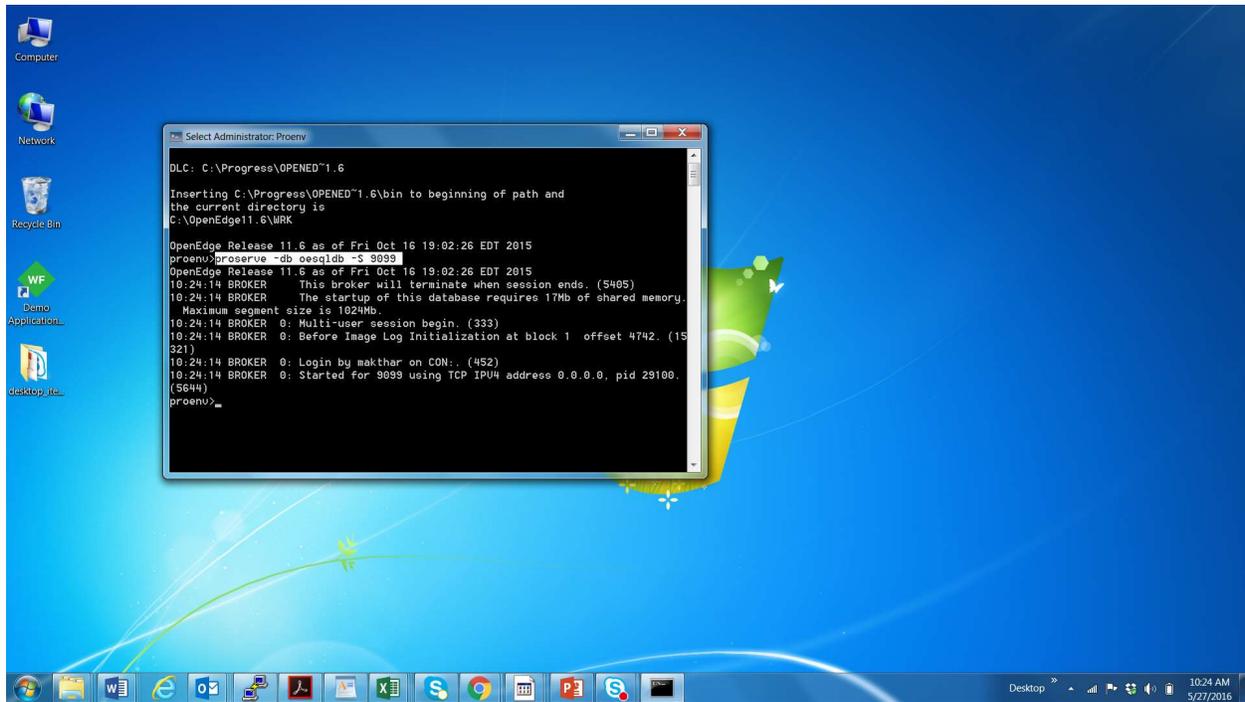


- a. Start the database and connect to the database using “Squirrel” as the SQL client.
Step 1: Search for the “proenv” and press “Enter”.

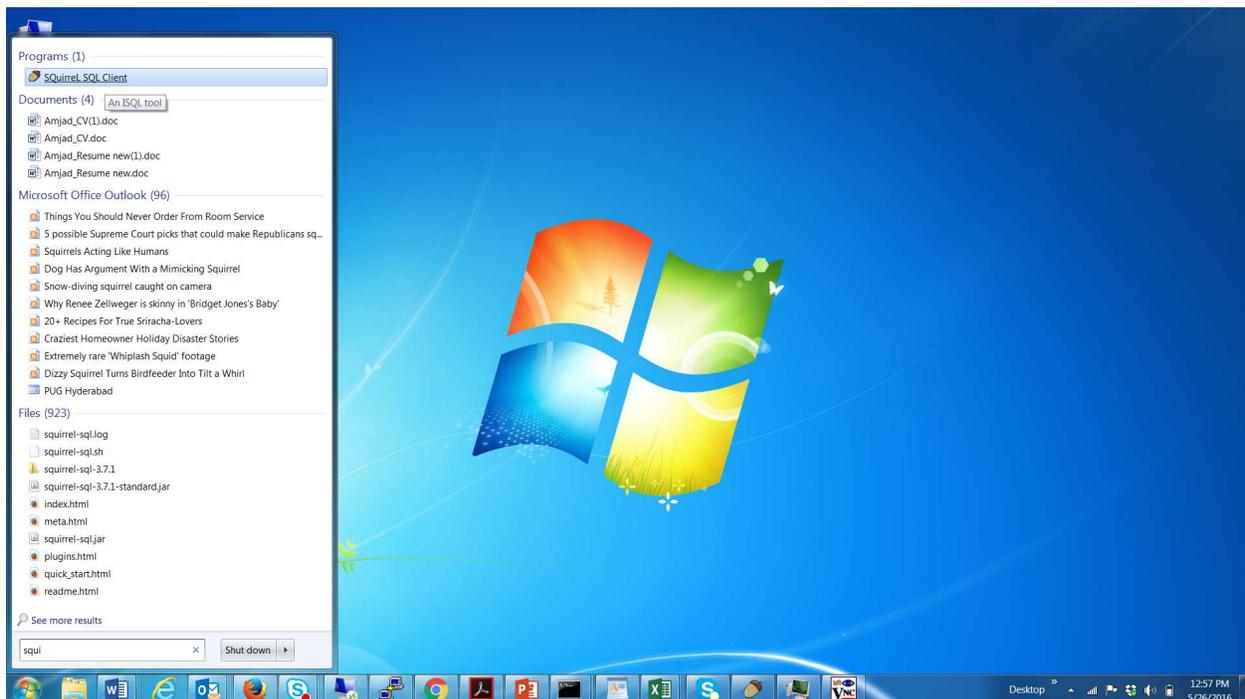


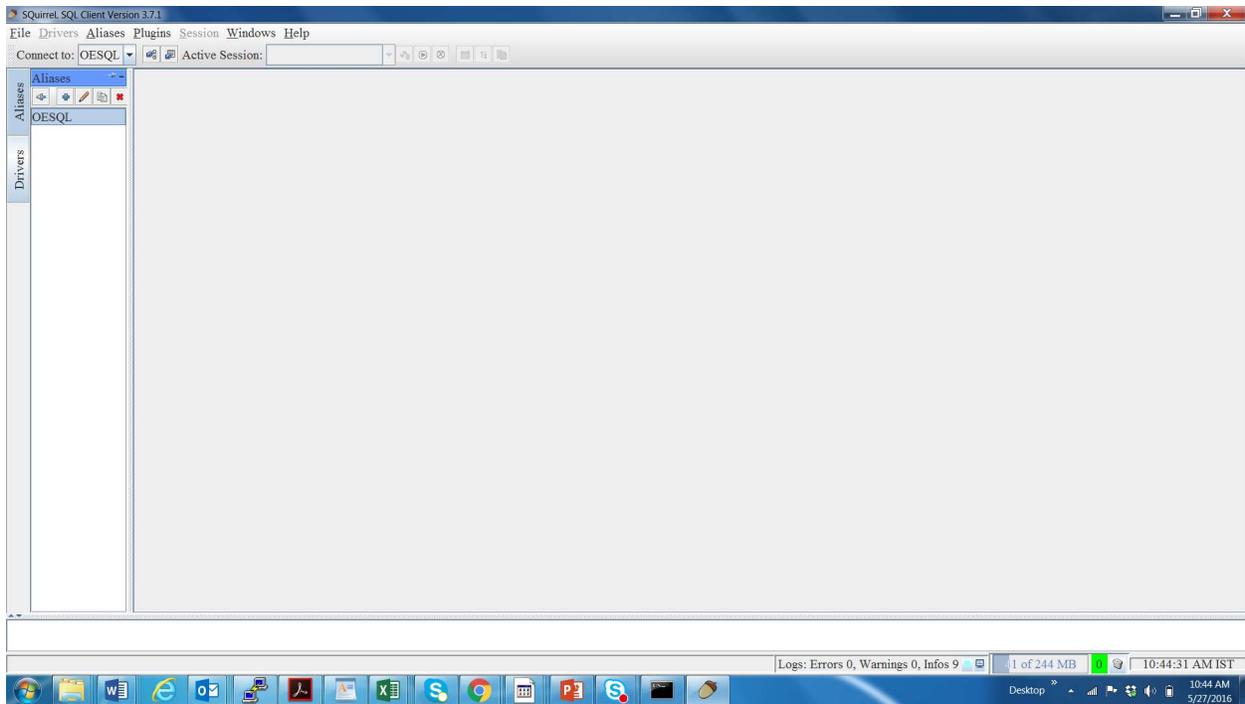
Step 2: Start the database using “proserve” command. This command accepts the database name and the port number where the database listens.

```
proserve -db oesqldb -S 9099
```

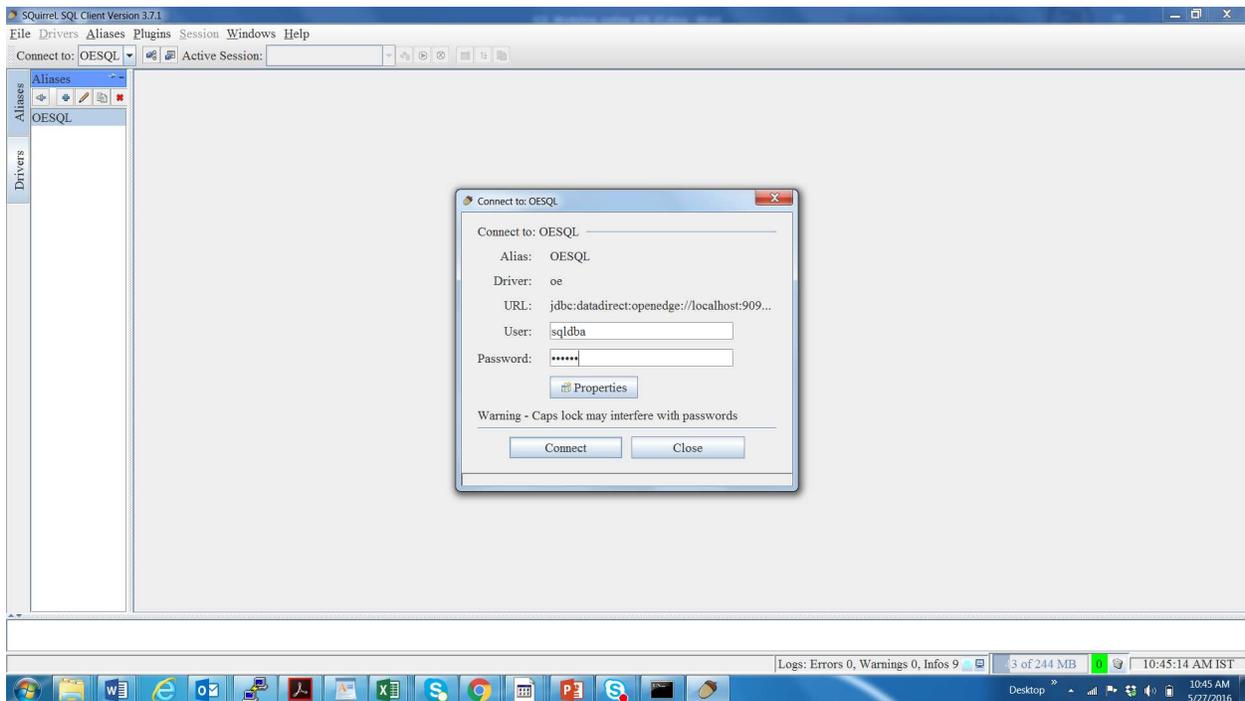


Step 3: Start Squirrel client by searching “Squirrel”.

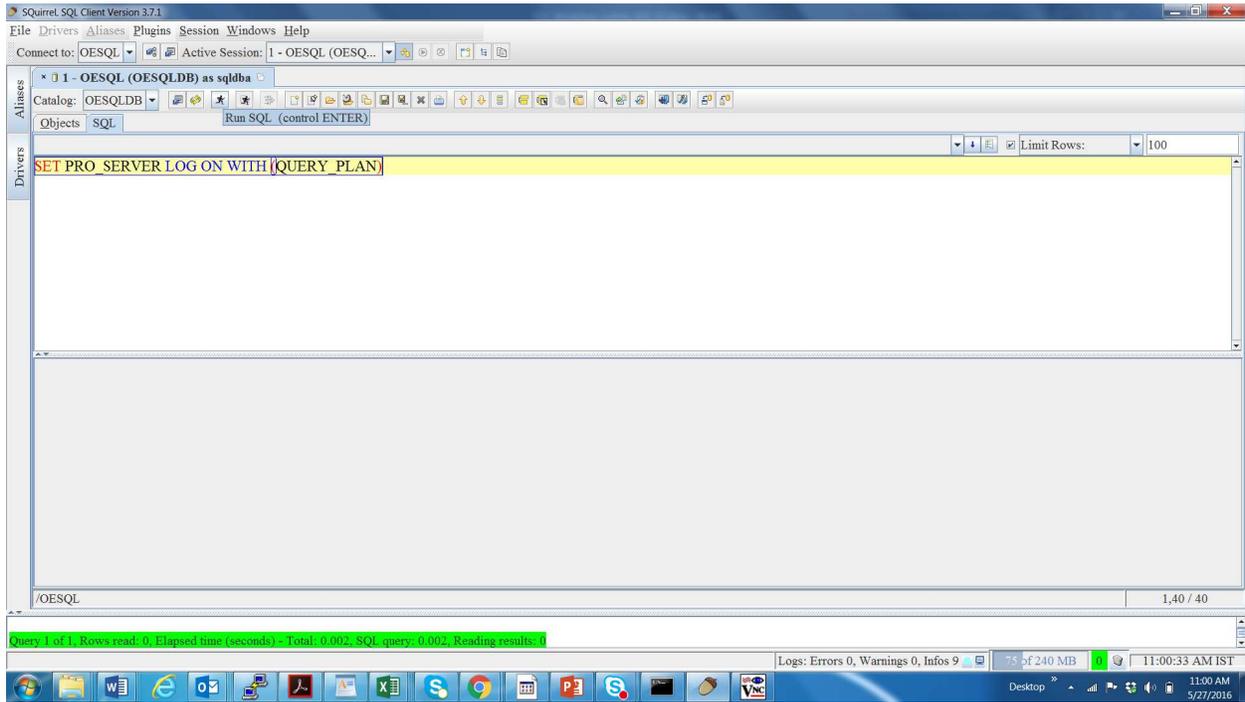




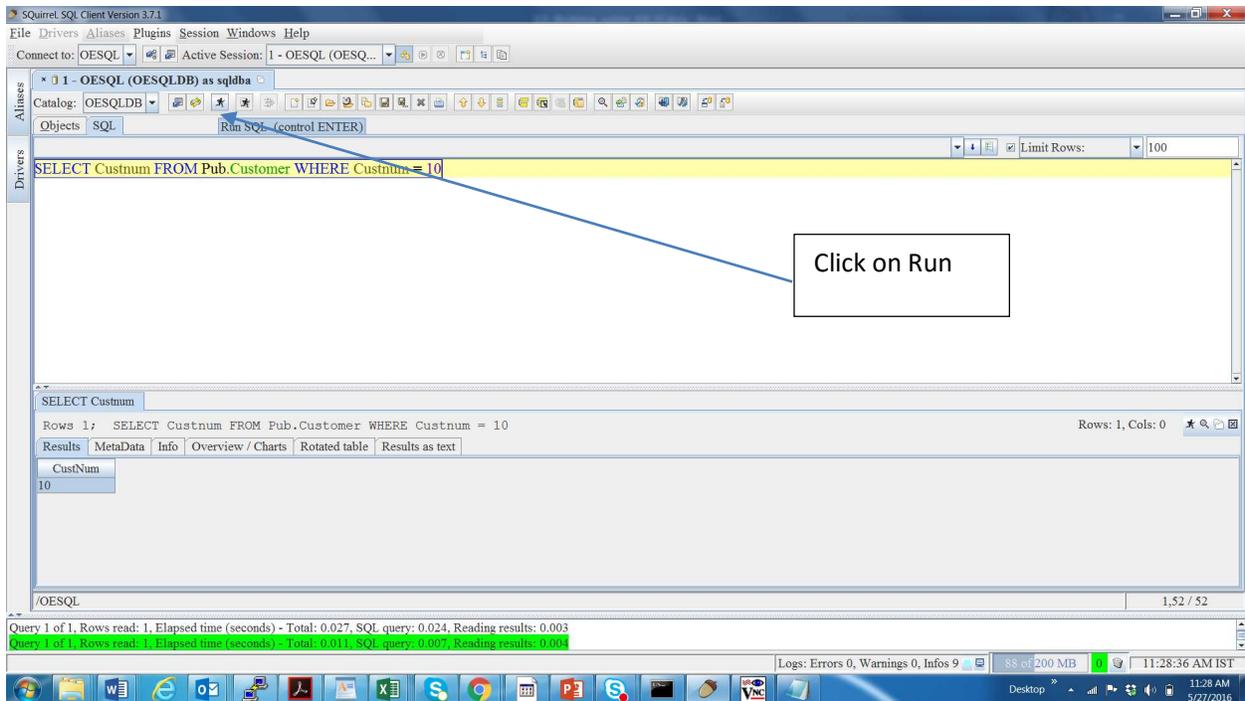
Step 4: Double click on “OESQL” alias name and click on “Connect”. User ‘sqldba’ is an existing user password for this user is “sqldba” (same as user name).



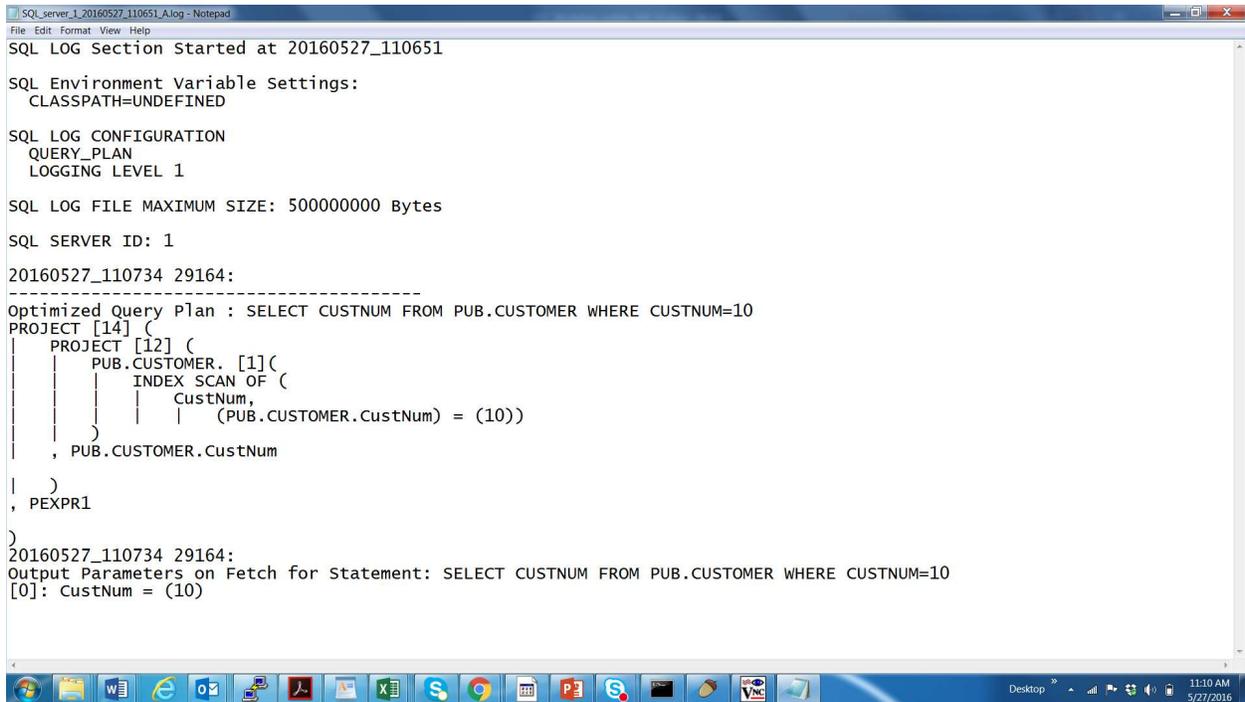
Step 5: Enable server log to display query plans, by executing below mentioned command and then “Run”
SET PRO_SERVER LOG ON WITH (QUERY_PLAN);



Step 6: Run the below query for which we want to see the query plan then “Run”
SELECT Custnum FROM Pub.Customer WHERE Custnum = 10;



Step 7: Open the log file with name "SQL_server_X_XXXX_XXXX_A.log" at location C:\OpenEdge\WRK



```
SQL_server_1_20160527_110651_A.log - Notepad
File Edit Format View Help
SQL LOG Section Started at 20160527_110651

SQL Environment Variable Settings:
  CLASSPATH=UNDEFINED

SQL LOG CONFIGURATION
  QUERY_PLAN
  LOGGING LEVEL 1

SQL LOG FILE MAXIMUM SIZE: 500000000 Bytes

SQL SERVER ID: 1

20160527_110734 29164:
-----
Optimized Query Plan : SELECT CUSTNUM FROM PUB.CUSTOMER WHERE CUSTNUM=10
PROJECT [14] (
  PROJECT [12] (
    PUB.CUSTOMER. [1] (
      INDEX SCAN OF (
        CustNum,
        (PUB.CUSTOMER.CustNum) = (10))
      )
    , PUB.CUSTOMER.CustNum
  )
  , PEXPR1
)
20160527_110734 29164:
Output Parameters on Fetch for Statement: SELECT CUSTNUM FROM PUB.CUSTOMER WHERE CUSTNUM=10
[0]: CustNum = (10)
```

Step 8: Query plan using the System Table, Pub."_Sql_QPlan"

Run the below mentioned query in Squirrel window.

```
SELECT "_Description" FROM Pub."_Sql_Qplan" WHERE "_Pnumber"=(SELECT MAX("_Pnumber")  
FROM PUB."_Sql_Qplan" WHERE "_Ptype">0);
```

The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

```
SELECT "_Description" FROM Pub."_Sql_Qplan" WHERE "_Pnumber"=(SELECT MAX("_Pnumber") FROM PUB."_Sql_Qplan" WHERE "_Ptype">0);
```

The query execution plan is shown below the query text:

```
SELECT Command. SELECT "_Descri  
Rows 12; SELECT "_Description" FROM Pub."_Sql_Qplan" WHERE "_Pnumber"=(SELECT MAX("_Pnumber") FROM PUB."_Sql_Qplan" WHERE "_Ptype">0) Rows: 1, Cols: 0
```

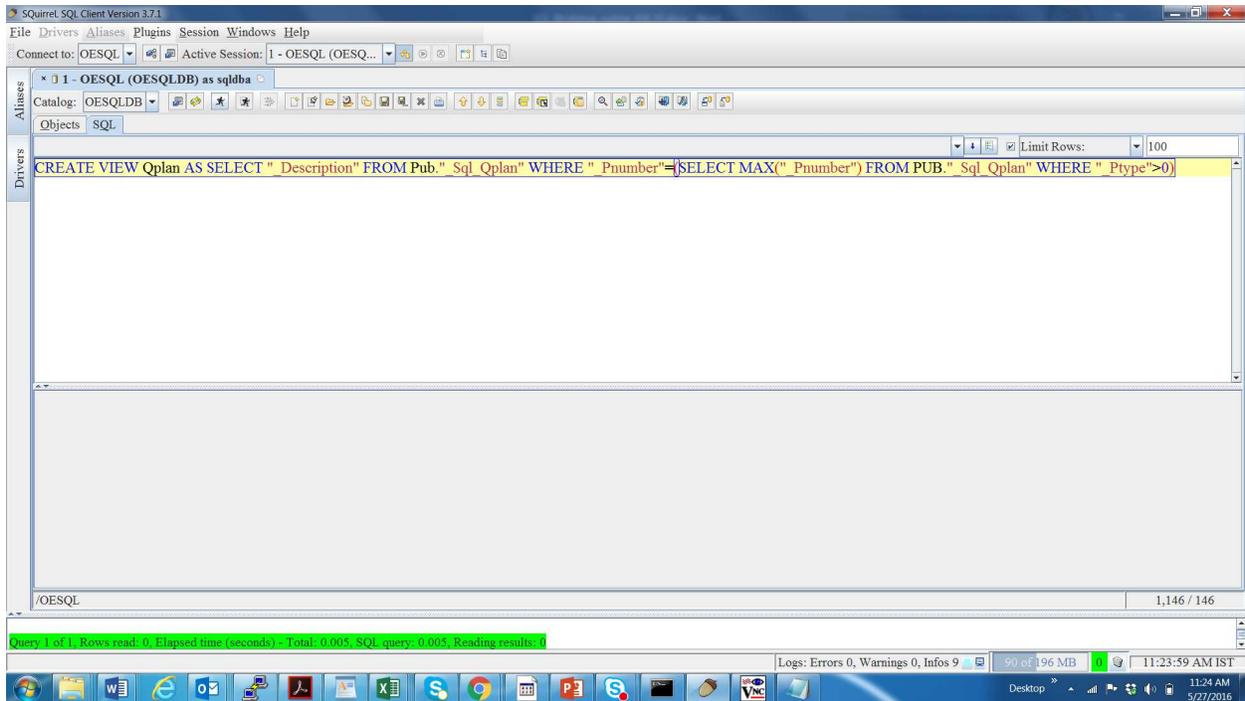
The execution plan details are as follows:

Step	Operation	Cost	Time	Rows	Bytes
1	SELECT COMMAND				
2	PROJECT [14] (
3	PROJECT [12] (
4	PUB.CUSTOMER_ [1] (
5	INDEX SCAN OF (
6	CustNum,				
7	(PUB.CUSTOMER.CustNum) = (10))				
8	PUB.CUSTOMER.CustNum				
9	PEXPRI				

The status bar at the bottom indicates: Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.011, SQL query: 0.007, Reading results: 0.004.

Step 2-9: Create the below view which helps us in looking at the query plan with ease. Throughout this session we will be using this view to check the query plan.

CREATE VIEW Qplan AS SELECT "_Description" FROM Pub."_Sql_Qplan" WHERE "_Pnumber"=(SELECT MAX("_Pnumber") FROM PUB."_Sql_Qplan" WHERE "_Ptype">0)



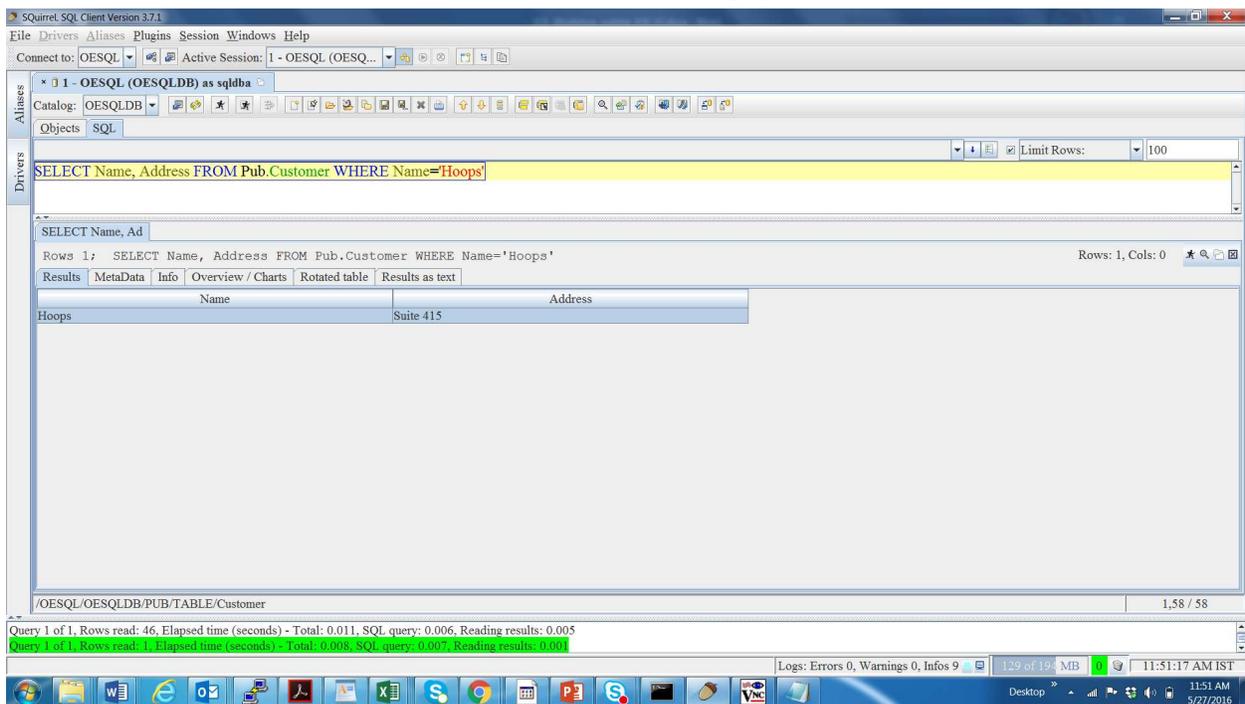
Chapter 3 – Understanding different query operations from Query plan

In this chapter, most of the query operations are explained with examples. You need to execute below queries, where each query will demonstrate one or some set of query operations using query plans.

- a. Table scan
- b. Index scan
- c. Project
- d. Restrict
- e. Join
- f. Aggregation
- g. Sort
- h. Union
- i. TIDUNION
- j. Dynamic Index (DIDX)

Query 1 – Below query demonstrates “Project” and “Index Scan” operations.

SELECT Name, Address FROM Pub.Customer WHERE Name='Hoops';



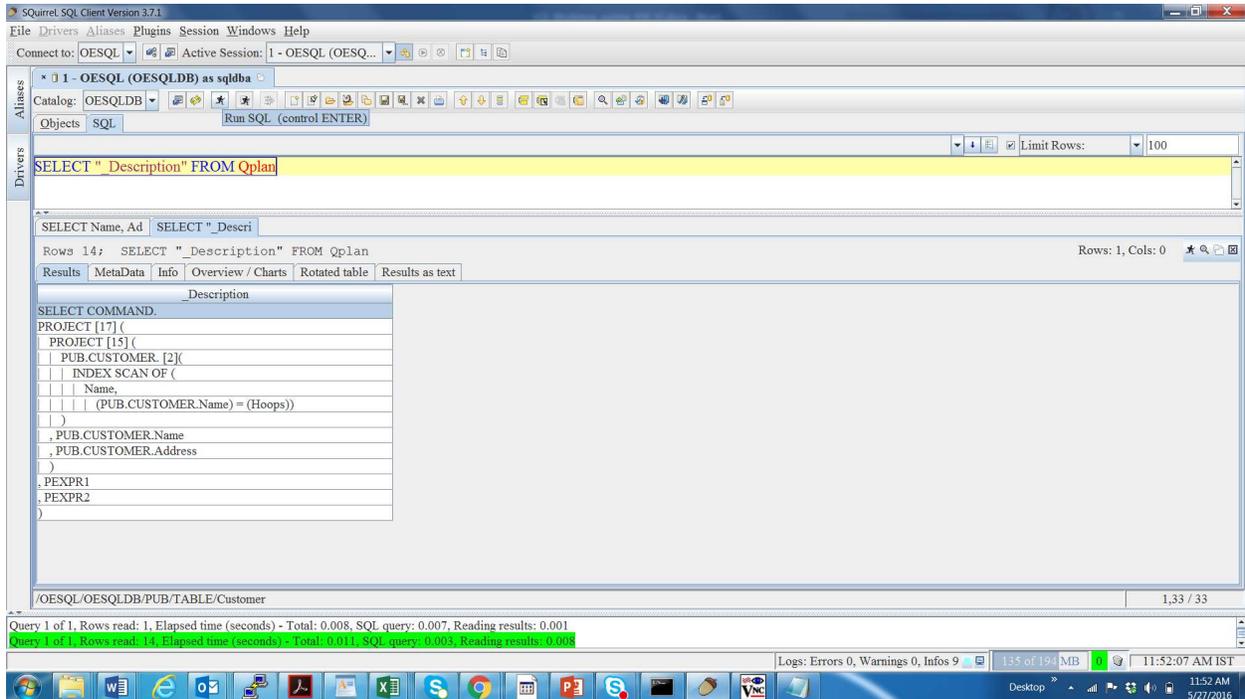
The screenshot shows the Squirrel SQL Client interface. The query editor contains the SQL statement: `SELECT Name, Address FROM Pub.Customer WHERE Name='Hoops'`. The results pane displays a single row with the following data:

Name	Address
Hoops	Suite 415

The status bar at the bottom indicates: Query 1 of 1, Rows read: 46, Elapsed time (seconds) - Total: 0.011, SQL query: 0.006, Reading results: 0.005. The system tray shows the date and time as 11:51 AM on 5/27/2016.

Then run select on Qplan view to see the Query plan

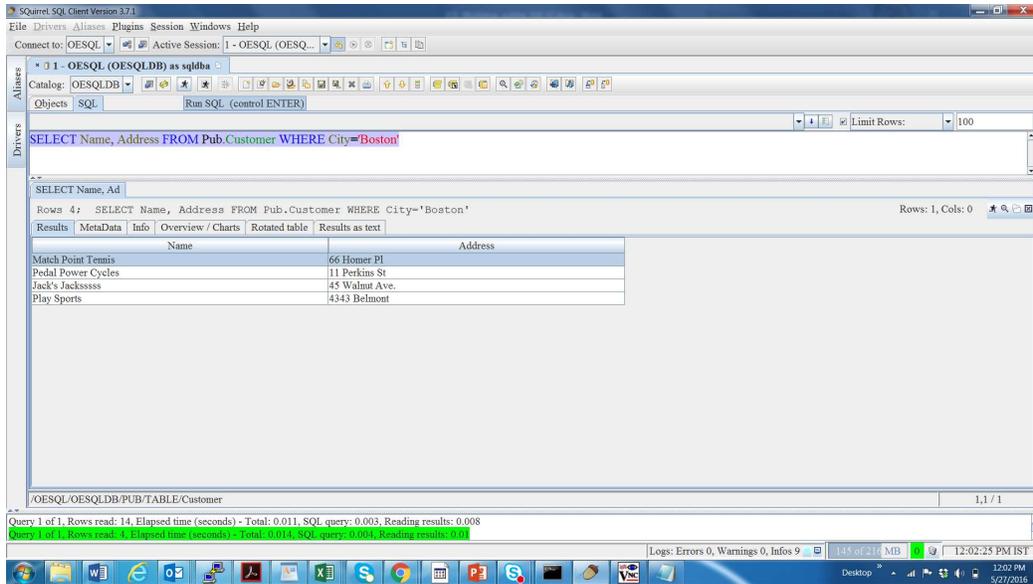
SELECT "_Description" FROM Qplan;



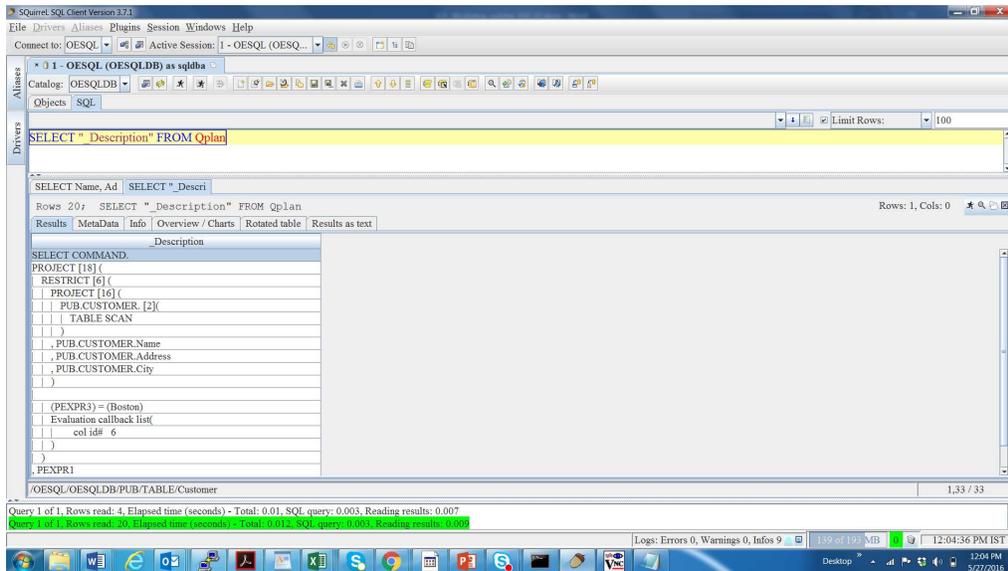
```
SELECT COMMAND.  
PROJECT [17] (  
  PROJECT [15] (  
    PUB.CUSTOMER. [2](  
      INDEX SCAN OF (  
        Name,  
        (PUB.CUSTOMER.Name) = (Hoops))  
    )  
  , PUB.CUSTOMER.Name  
  , PUB.CUSTOMER.Address  
  )  
, PEXPR1  
, PEXPR2  
)
```

Query 2 – Below query demonstrates “Restrict” and “Table Scan” operations.

SELECT Name, Address FROM Pub.Customer WHERE City='Boston';



Now Check Query plan.



```

SELECT COMMAND.
PROJECT [18] (
  | RESTRICT [6] (
    | | PROJECT [16] (
    | | | PUB.CUSTOMER. [2](
    | | | | TABLE SCAN
    | | | )
    | | , PUB.CUSTOMER.Name
  )
)

```

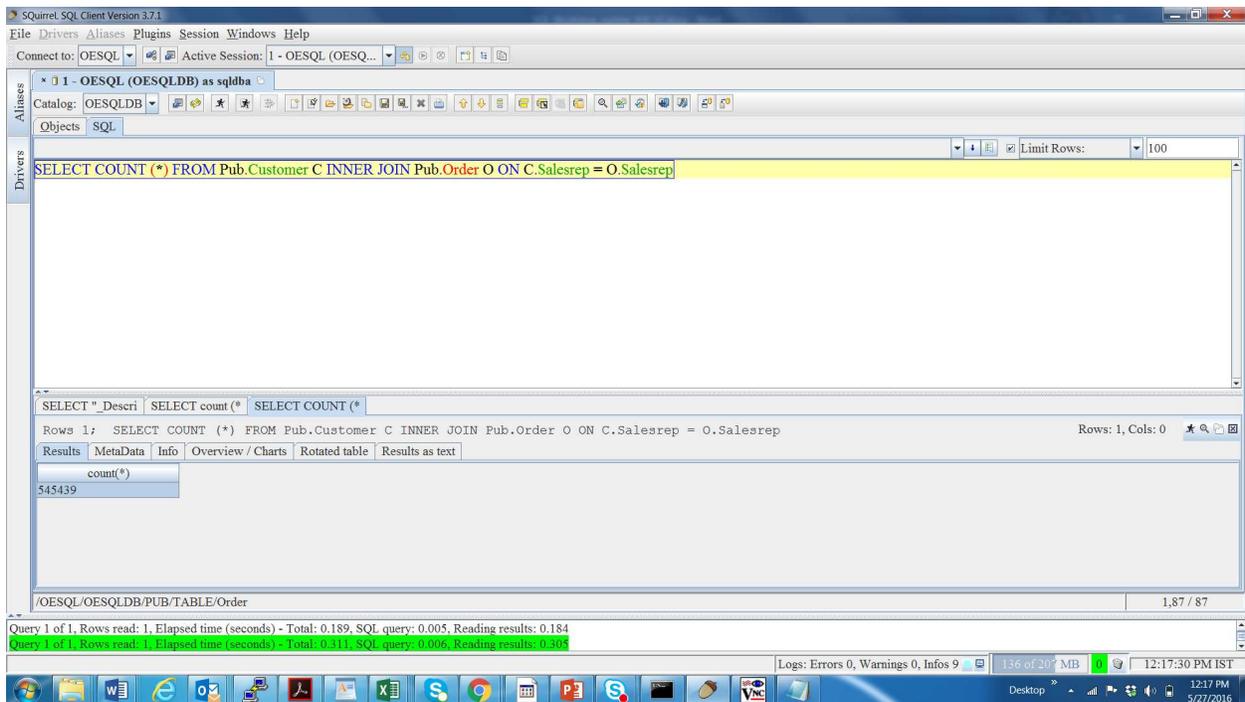
```

| | , PUB.CUSTOMER.Address
| | , PUB.CUSTOMER.City
| | )
|
| | (PEXPR3) = (Boston)
| | Evaluation callback list(
| | | col id# 6
| | )
| )
, PEXPR1
, PEXPR2
)

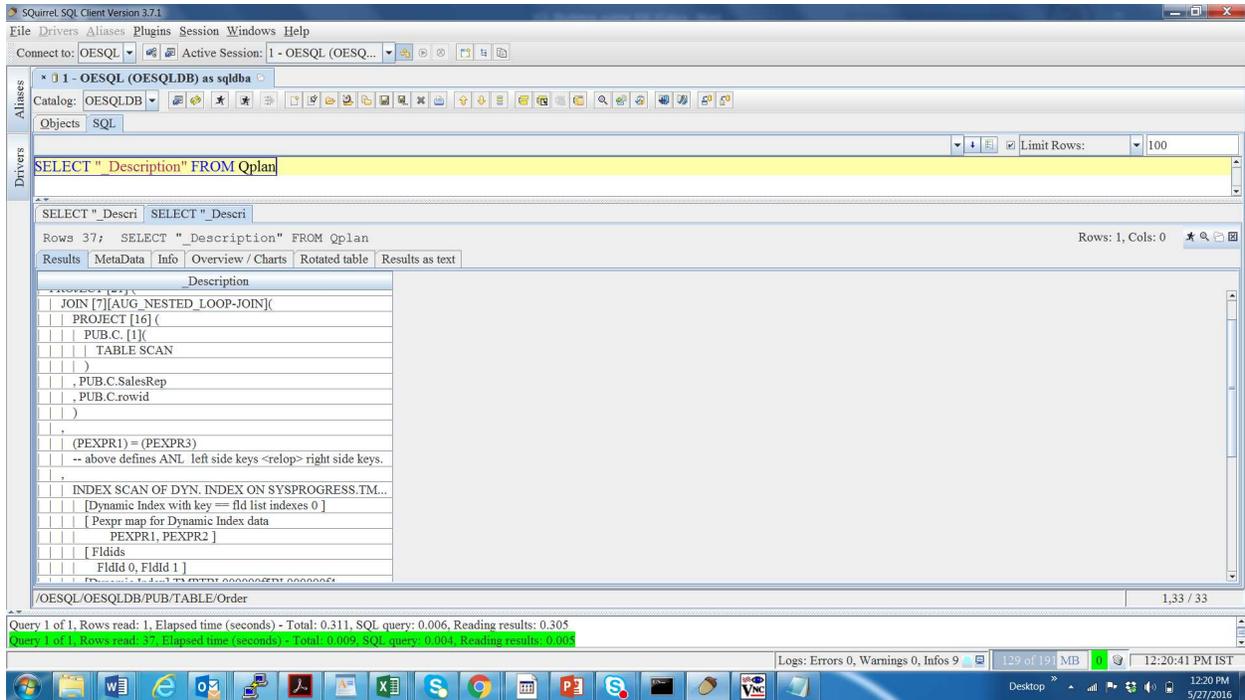
```

Query 3 – Below query demonstrates demonstrate “Join” and “DIDX” operations.

SELECT COUNT() FROM Pub.Customer C INNER JOIN Pub.Order O ON C.Salesrep = O.Salesrep;*



Now Check Query plan



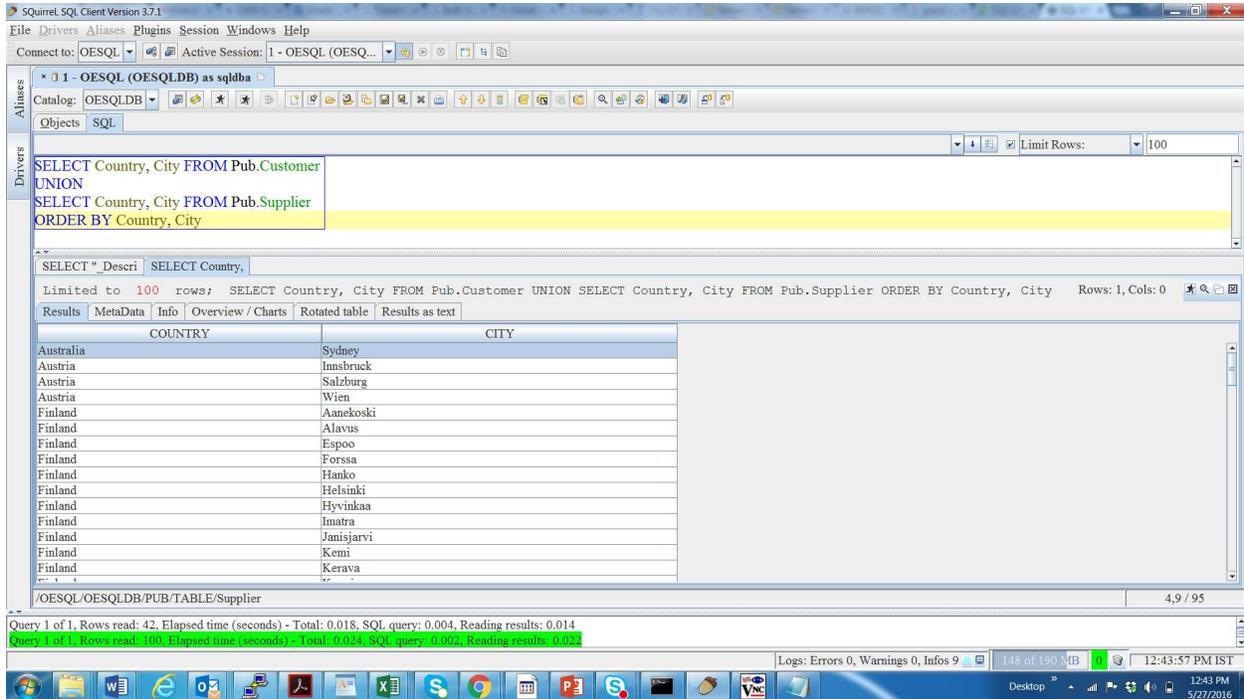
```

SELECT COMMAND.
PROJECT [22] (
| PROJECT [21] (
| | JOIN [7][AUG_NESTED_LOOP-JOIN](
| | | PROJECT [16] (
| | | | PUB.C. [1](
| | | | | TABLE SCAN
| | | | , PUB.C.SalesRep
| | | | , PUB.C.rowid
| | | | ,
| | | | (PEXPR1) = (PEXPR3)
| | | | -- above defines ANL left side keys <
| | | | ,
| | | | INDEX SCAN OF DYN. INDEX ON SYSPROGRES
| | | | [Dynamic Index with key == fld lis
| | | | [ Pexpr map for Dynamic Index data
| | | | PEXPR1, PEXPR2 ]
| | | | [ Fldids
| | | | Fldid 0, Fldid 1 ]
| | | | [Dynamic Index] TMPTBL000000f5BL00
| | | | (SYSPROGRESS.TMPTBL000000f4.0)
| | | | PROJECT [19] (
| | | | | PUB.O. [2](
| | | | | | TABLE SCAN

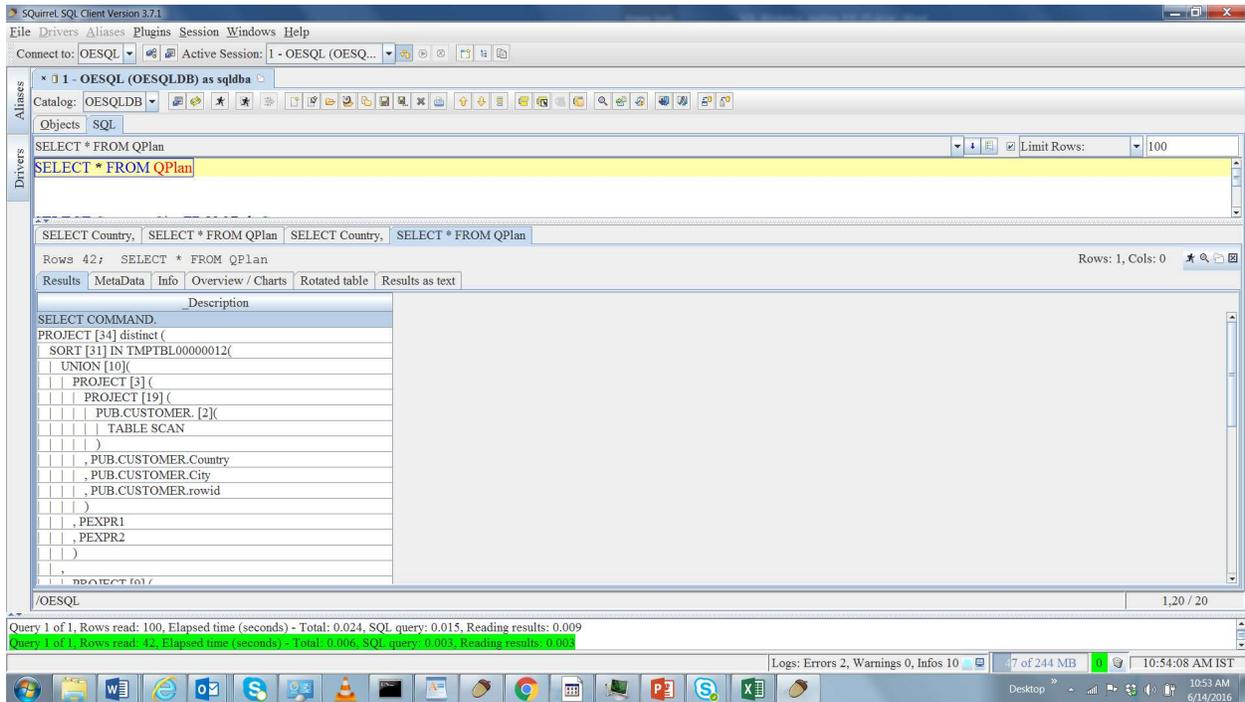
```


Query 5 - Below query demonstrates "Union" and "Sort" operations.

```
SELECT Country, City FROM Pub.Customer
UNION
SELECT Country, City FROM Pub.Supplier
ORDER BY Country, City;
```



Now check Query plan



```

SELECT COMMAND.
PROJECT [34] distinct (
  | SORT [31] IN TMP... [17c]
  | | UNION [10]
  | | | PROJECT [3] (
  | | | | PROJECT [19] (
  | | | | | PUB.CUSTOMER. [2] (
  | | | | | | TABLE SCAN
  | | | | | )
  | | | | | , PUB.CUSTOMER.Country
  | | | | | , PUB.CUSTOMER.City
  | | | | | , PUB.CUSTOMER.rowid
  | | | | )
  | | | | , PEXPR1
  | | | | , PEXPR2
  | | | )
  | | | ,
  | | | | PROJECT [9] (
  | | | | | PROJECT [27] (
  | | | | | | PUB.SUPPLIER. [6] (
  | | | | | | | TABLE SCAN
  | | | | | | | , PUB.SUPPLIER.Country
  | | | | | | | , PUB.SUPPLIER.City
  | | | | | | | , PUB.SUPPLIER.rowid
  | | | | | | )
  | | | | | | , PEXPR1
  | | | | | | , PEXPR2
  | | | | | , PEXPR1
  | | | | | , PEXPR2
  | | | | SORT BY (
  | | | | | , Sort project expression #0 == PEXPR1
    
```

| | , Sort project expression #1 == PEXPR2

Query 6 – Below query demonstrates TIDUNION operation.

SELECT custnum, name, Address, City from pub.customer where custnum=10 OR name = 'Hoops';

The screenshot shows the Squirrel SQL Client interface. The query editor contains the following SQL statement:

```
SELECT custnum, name, Address, City from pub.customer where custnum=10 OR name = 'Hoops'
```

The results pane displays the following data:

CustNum	Name	Address	City
3	Hoops	Suite 415	Atlanta
10	Just Joggers Limited	Fairwind Trading Est	Ramsbottom

Below the results, the status bar indicates: Query 1 of 1, Rows read: 22, Elapsed time (seconds) - Total: 0.008, SQL query: 0.003, Reading results: 0.005.

Now check the query plan

The screenshot shows the Squirrel SQL Client interface with the query plan for the same query. The query plan is displayed in a tree view:

```
SELECT "Description" FROM Qplan
```

The query plan details are as follows:

- SELECT COMMAND.
- PROJECT [28] (
 - PROJECT [26] (
 - TIDUNION [29] (
 - PROJECT [37] (
 - PUB.CUSTOMER. [34] (
 - INDEX SCAN OF (
 - CustNum,
 - (PUB.CUSTOMER.CustNum) = (10)

- PUB.CUSTOMER.rowid
- PUB.CUSTOMER.CustNum
- PROJECT [45] (
- PUB.CUSTOMER. [42] (
 - INDEX SCAN OF (
 - Name,
 - (PUB.CUSTOMER.Name) = ('Hoops'))

```

SELECT COMMAND.
PROJECT [28] (
| PROJECT [26] (
| | TIDUNION [29] (
| | | PROJECT [37] (
| | | | PUB.CUSTOMER. [34](
| | | | | INDEX SCAN OF (
| | | | | | CustNum,
| | | | | | | (PUB.CUSTOMER.CustNum)
| | | | | )
| | | , PUB.CUSTOMER.rowid
| | | , PUB.CUSTOMER.CustNum
| | | ),
| | | PROJECT [45] (
| | | | PUB.CUSTOMER. [42](
| | | | | INDEX SCAN OF (
| | | | | | Name,
| | | | | | | (PUB.CUSTOMER.Name) =
| | | | | )
| | | , PUB.CUSTOMER.rowid
| | | , PUB.CUSTOMER.Name
| | | )
| | )
| , PUB.CUSTOMER.CustNum
| , PUB.CUSTOMER.Name
| , PUB.CUSTOMER.Address
| , PUB.CUSTOMER.City
| )

```

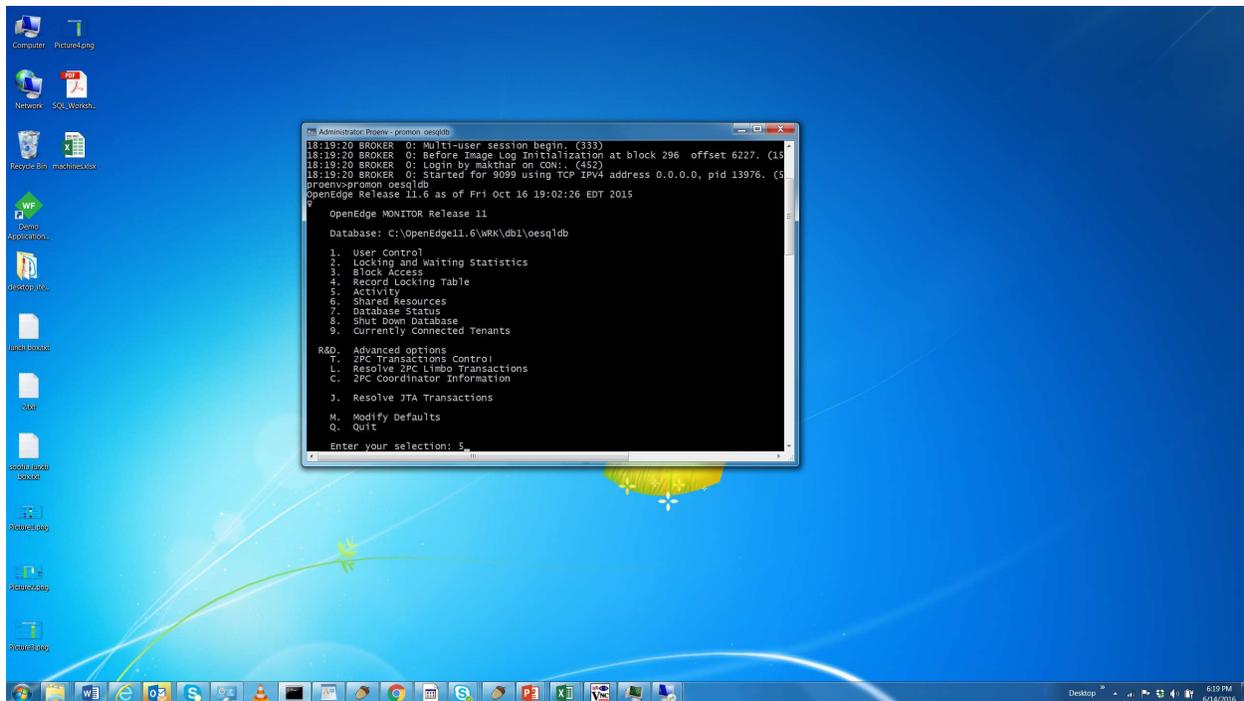
Chapter 4 - Table Scan Vs. Index scan

In this chapter, you will be asked to run a simple query which does table scan and capture the number of IOs using promon utility. Then, you will be asked to create an index and run the same query again and check the number of IOs using promon utility. Then, we compare the number of IOs which took place in table scan versus index scan. In both the cases, we will look at query plan to confirm the query operation.

How to start promon?

Go to proenv window and type below command and select option 5. Option 5 will display the number of records read.

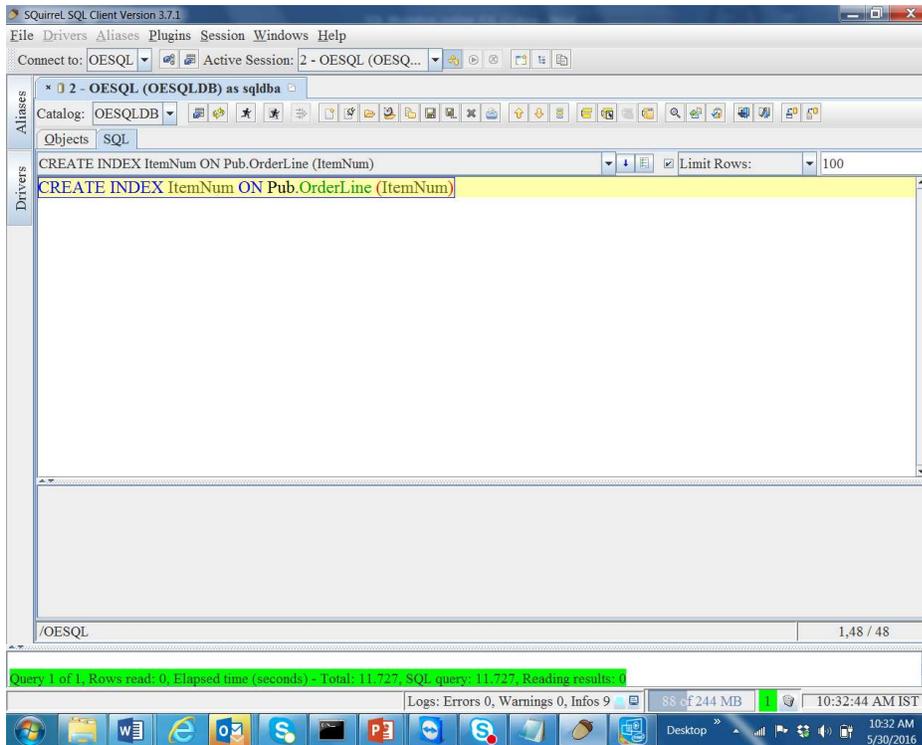
promon oesqldb



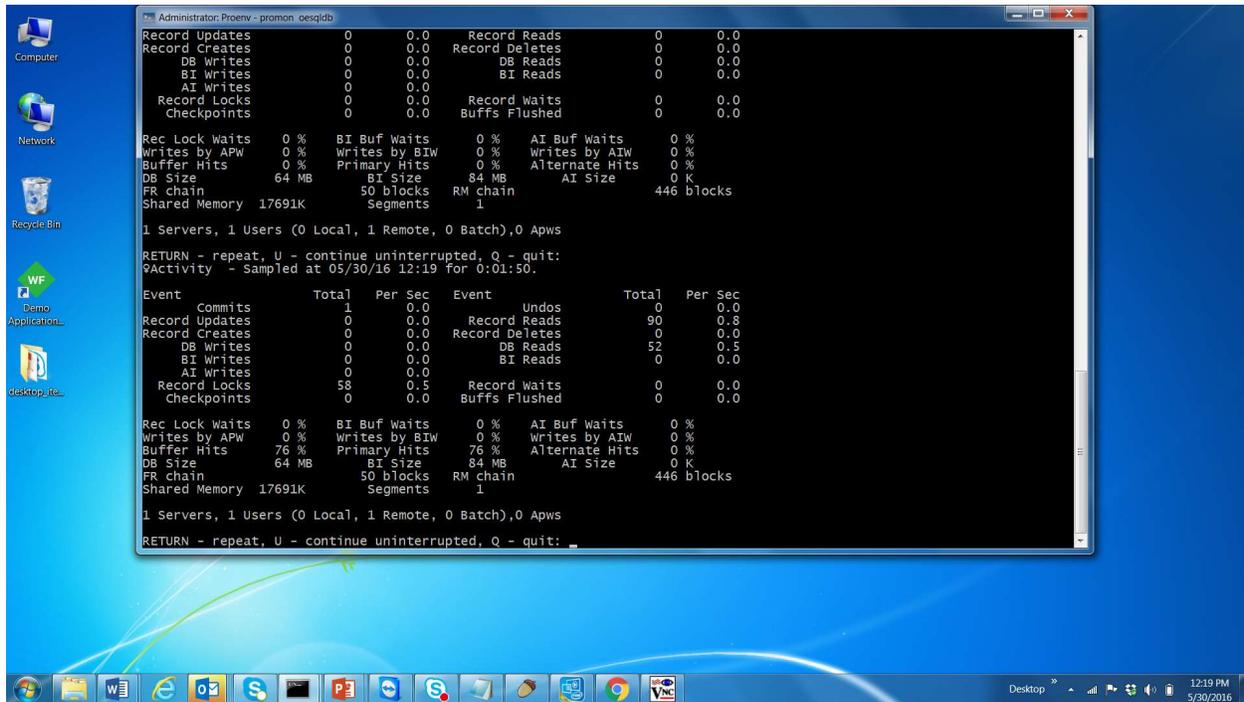

```
| , PEXPR2  
| )  
, PEXPR1  
, PEXPR2  
)
```

Now, let us create an index on ItemNum column of Pub.OrderLine table.

CREATE INDEX ItemNum ON Pub.OrderLine (ItemNum);



Now, check the number of records read.



Observation – Did you observe the drastic change in the number of records read during Index scan when compared to Table scan? This difference is because, Index scan will have intelligence to read only the required records and skip all other records of a table.

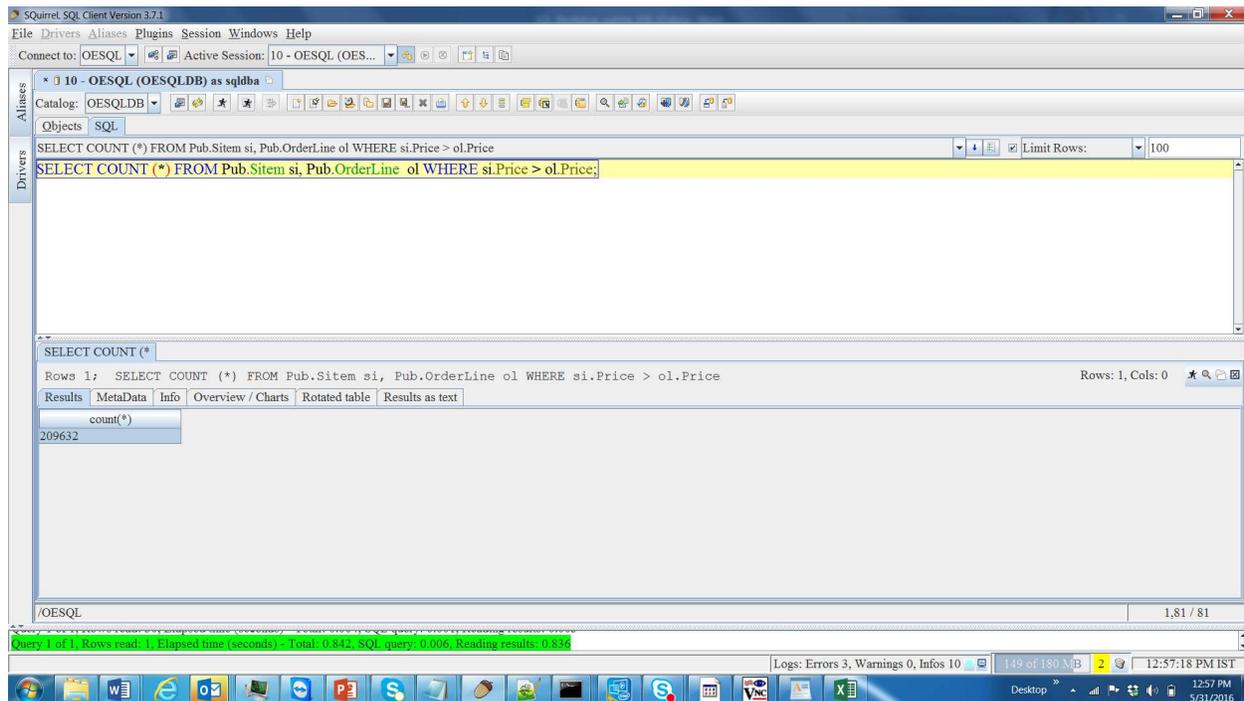
Chapter 5 - Join Methods

In this chapter, we will explain the different join methods used by OE-SQL and you will be learning what are “Good” join methods (Augmented Nested Loops and Merge Joins in OE world) and what is a “Bad” join method (Nested Loop). Basically, “Good” joins are the ones which uses physical index or Dynamic index on the right side of the join to evaluate the join conditions. Here, you will be asked to run a join query which results in “Bad” join and then, create an index and then, asked to run the same query again to see the “Good” join. We will check the join methods through query plans. You will also use promon utility to check the number of IOs in both cases. There can be cases where, SQL optimizer selects “Nested Loop Join” which might be better than other choices.

Nested Loop Joins:

Run the Below query which results in Nested Loop Join

```
SELECT COUNT (*) FROM Pub.Sitem si, Pub.OrderLine ol WHERE si.Price > ol.Price;
```



The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

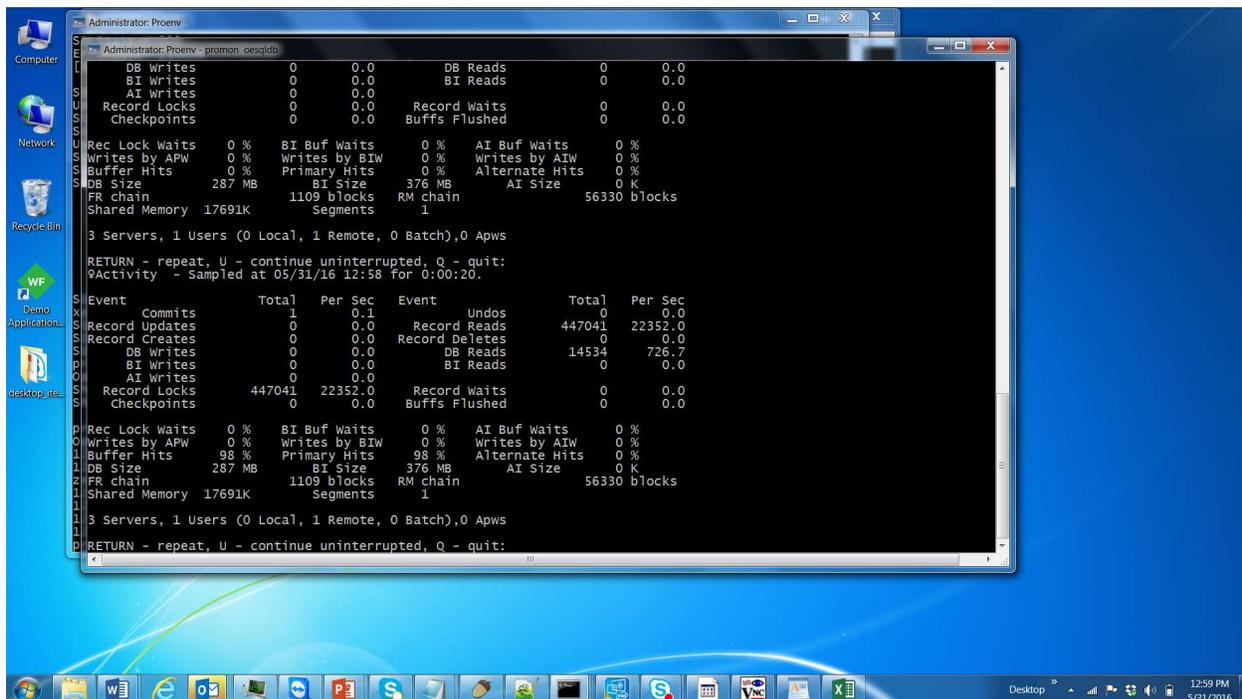
```
SELECT COUNT (*) FROM Pub.Sitem si, Pub.OrderLine ol WHERE si.Price > ol.Price;
```

The query has been executed, and the results are shown in a table with one row and one column:

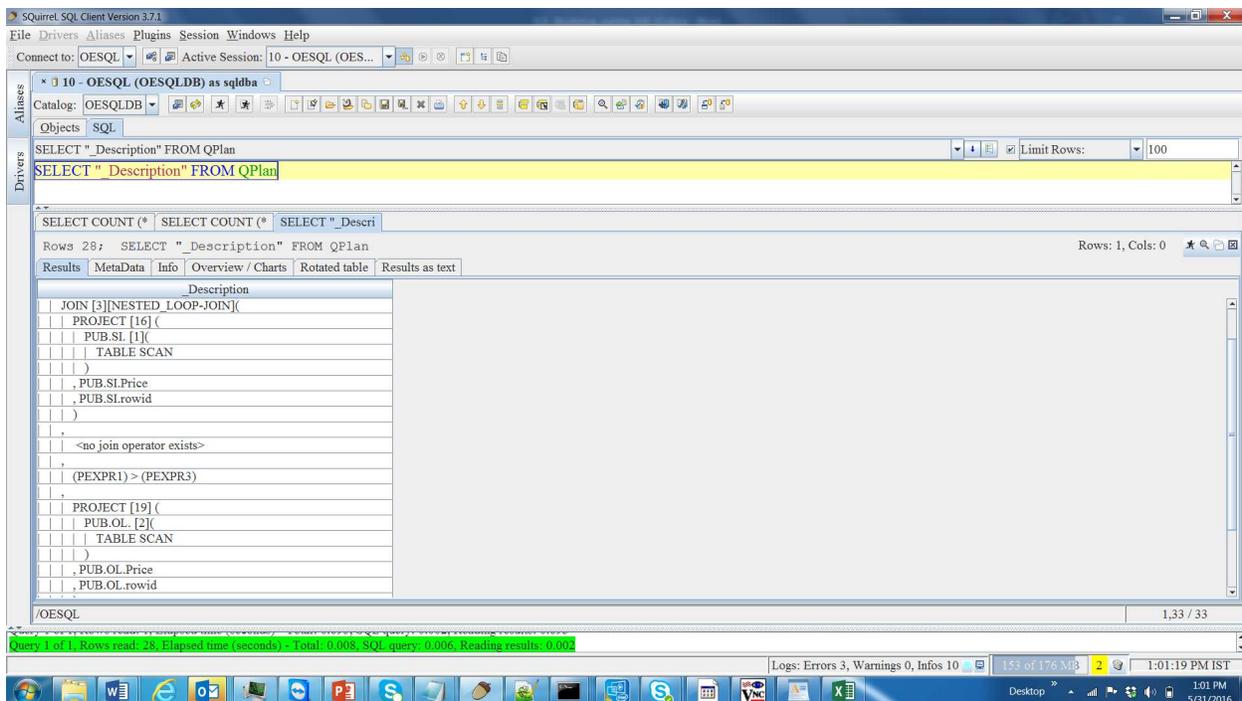
count(*)
209632

The status bar at the bottom of the window indicates: "Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.842, SQL query: 0.006, Reading results: 0.834". The system tray at the bottom shows the date and time as 5/21/2016, 12:57:18 PM IST.

Now check the number of DB Requests.



Now, check the plan



_Description

SELECT COMMAND.

```

PROJECT [22] (
| PROJECT [21] (
| | JOIN [3][NESTED_LOOP-JOIN](
| | | PROJECT [16] (
| | | | PUB.SI. [1](
| | | | | TABLE SCAN
| | | | )
| | | | , PUB.SI.Price
| | | | , PUB.SI.rowid
| | | )
| | | ,
| | | <no join operator exists>
| | | ,
| | | (PEXPR1) > (PEXPR3)
| | | ,
| | | PROJECT [19] (
| | | | PUB.OL. [2](
| | | | | TABLE SCAN
| | | | | )
| | | | , PUB.OL.Price
| | | | , PUB.OL.rowid
| | | | )
| | | )
| | | , count (*)
| | )
, PEXPR1

```

Augmented Nested Loop Joins:

Now, in order to get ANL join, create an index on Pub.OrderLine table

CREATE INDEX PricIdx ON Pub.OrderLine (Price);

The screenshot shows the Squirrel SQL Client interface. The main window displays the SQL statement: `CREATE INDEX PricIdx ON Pub.OrderLine (Price);`. The status bar at the bottom indicates the execution of a query: "Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 17.411, SQL query: 17.411, Reading results: 0". The Windows taskbar at the bottom shows the time as 11:49 AM on 5/31/2016.

Now, run the same query again. This time, it takes ANL.

SELECT COUNT () FROM Pub.Sitem si, Pub.OrderLine ol WHERE si.Price > ol.Price;*

The screenshot shows the Squirrel SQL Client interface. The main window displays the SQL statement: `SELECT COUNT (*) FROM Pub.Sitem si, Pub.OrderLine ol WHERE si.Price > ol.Price`. The results pane shows a single row with the value 209632. The status bar at the bottom indicates the execution of a query: "Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.315, SQL query: 0.005, Reading results: 0.31". The Windows taskbar at the bottom shows the time as 1:05 PM on 5/31/2016.

count(*)
209632

Check the number of DB requests

The screenshot shows a Windows desktop with a blue background. A command prompt window titled "Administrator: Proenv - promon oesqldb" is open, displaying SQL Server performance statistics. The statistics are as follows:

Rec Lock Waits	0 %	BI Buf Waits	0 %	AI Buf Waits	0 %
Writes by APW	0 %	Writes by BIW	0 %	Writes by AIW	0 %
Buffer Hits	0 %	Primary Hits	0 %	Alternate Hits	0 %
DB Size	398 MB	BI Size	236 MB	AI Size	0 K
FR chain	17696K	50 blocks	RM chain	445 blocks	
Shared Memory		Segments	1		

Below the statistics, it shows "1 Servers, 1 Users (0 Local, 1 Remote, 0 Batch), 0 Apws". The prompt "RETURN - repeat, U - continue uninterrupted, Q - quit:" is visible. The system tray at the bottom right shows the date and time as 6/15/2016 1:11 PM.

Check the query plan

The screenshot shows the SQL Server Enterprise Manager interface. The "Query Plan" tab is selected, displaying the execution plan for the query: "SELECT 'Description' FROM QPlan". The plan shows a "PROJECT [22]" operator, which is a "TABLE SCAN" of the "PUB.SI" table. The plan also shows a "PROJECT [19]" operator, which is an "INDEX SCAN OF (PRICEIDX)" on the "PUB.OL" table. The status bar at the bottom indicates "Query 1 of 1, Rows read: 30, Elapsed time (seconds) - Total: 0.012, SQL query: 0.007, Reading results: 0.005".

_Description

```
-----  
SELECT COMMAND.  
PROJECT [22] (  
| PROJECT [21] (  
| | JOIN [3][AUG_NESTED_LOOP-JOIN]  
| | | [RHS-SORTED(-ASC-DUPS) ](  
| | | PROJECT [16] (  
| | | | PUB.SI. [1](  
| | | | | TABLE SCAN  
| | | | )  
| | | | , PUB.SI.Price  
| | | | , PUB.SI.rowid  
| | | )  
| | | ,  
| | | (PEXPR1) > (PEXPR3)  
| | | -- above defines ANL left side keys <  
| | | ,  
| | | PROJECT [19] (  
| | | | PUB.OL. [2](  
| | | | | INDEX SCAN OF (  
| | | | | PRICEIDX,  
| | | | | | (PUB.OL.Price) < (null  
| | | | )  
| | | | , PUB.OL.Price  
| | | | , PUB.OL.rowid  
| | | )  
| | )  
| | , count (*)  
| )  
| , PEXPR1  
| )
```

Observation: Did you observe the drastic difference in number of records read? It is because, right side of the join in ANL has the index scan, which reads only the required records from that table.

Augmented Nested Loop joins with Dynamic Index:

When there is no index available for join, then SQL creates Dynamic Index during the runtime on the Right Side Table of Join. This join is called as ANL with DIDX (DIDX stands for Dynamic Index). This is also called as Hash Join.

Run the below query (Note that, there is no index available on ShipToId column of Pub.Order or Pub.ShipTo).

SELECT COUNT () FROM Pub.Order o, Pub.ShipTo s where s.ShipToId = o.ShipToId*

The screenshot displays the Squirrel SQL Client interface. The main window shows the following SQL query:

```
SELECT COUNT (*) FROM Pub.Order o, Pub.ShipTo s where s.ShipToId = o.ShipToId
```

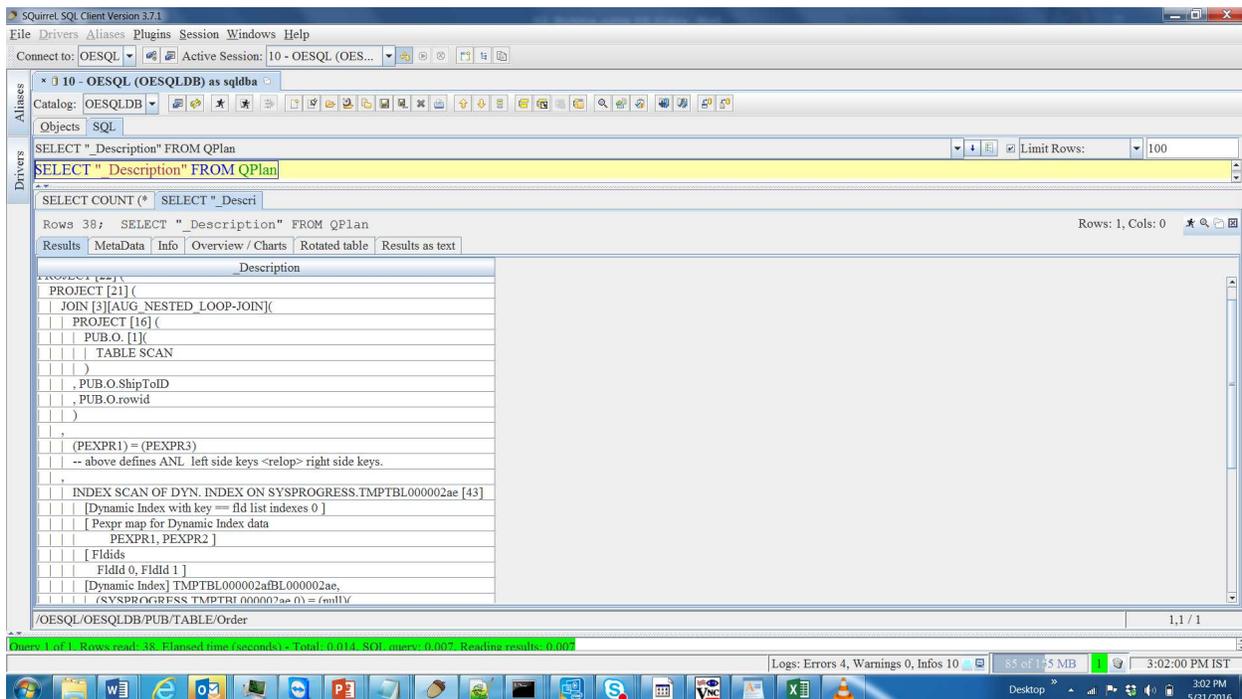
The results pane below the query shows the following output:

count(*)
1286

The status bar at the bottom of the client window displays the following information:

Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.024, SQL query: 0.012, Reading results: 0.041

Check the query plan, it should generate Dynamic Index.



_Description

SELECT COMMAND.

```
PROJECT [22] (  
| PROJECT [21] (  
| | JOIN [3][AUG_NESTED_LOOP-JOIN](  
| | | PROJECT [16] (  
| | | | PUB.O. [1](  
| | | | | TABLE SCAN  
| | | | )  
| | | , PUB.O.ShipToID  
| | | , PUB.O.rowid  
| | | )  
| | ,  
| | | (PEXPR1) = (PEXPR3)  
| | | -- above defines ANL left side keys <  
| | | ,  
| | | INDEX SCAN OF DYN. INDEX ON SYSPROGRES  
| | | | [Dynamic Index with key == fld lis  
| | | | [ Pexpr map for Dynamic Index data
```

```
| | | | PEXPR1, PEXPR2 ]
| | | | [ Fldids
| | | | FldId 0, FldId 1 ]
| | | | [Dynamic Index] TMPTBL00000011BL00
| | | | | (SYSPROGRESS.TMPTBL00000010.0)
| | | | PROJECT [19] (
| | | | | PUB.S. [2](
| | | | | | INDEX SCAN OF (
| | | | | | | custnumshipto, - F
| | | | | | )
| | | | | , PUB.S.ShipToID
| | | | | , PUB.S.rowid
| | | | | )
| | |
| | | )
| | )
| , count (*)
| )
, PEXPR1
)
```

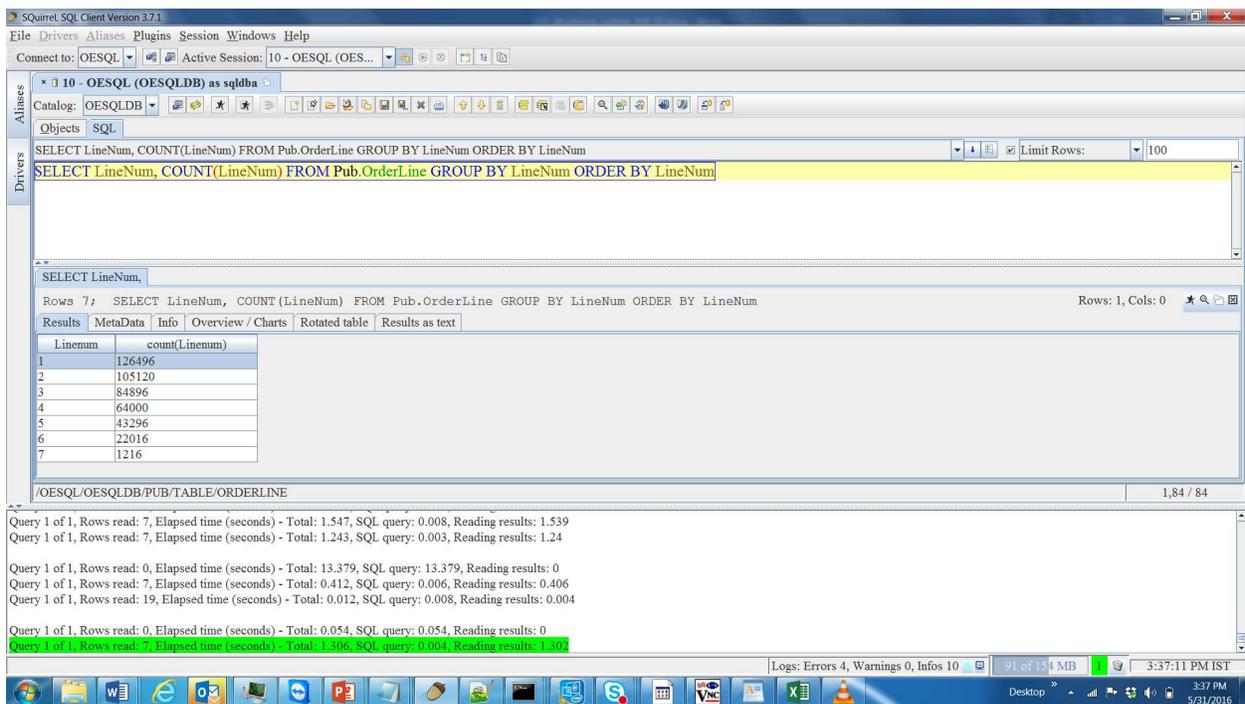
Chapter 6 - Aggregations

In this chapter, we will explain the different aggregation methods used by OE-SQL and then, you will be asked to run a query which results in “Hash Aggregation”. Then, we will create an index and run the same query which now results in “Stream Aggregation”. Basically, “Hash aggregation” uses “Hash Table” to group the same column values, whereas, “Stream Aggregation” uses the “Index scan” operation on the grouping columns and thus does not require any “Hash Table” to group the same column values. We will compare the difference between these two aggregation methods in terms of execution time and the reason behind this difference. All this will be done by analyzing the query plan for both the cases.

Hash Aggregation:

Run the below query which results in Hash Aggregation.

```
SELECT LineNum, COUNT(LineNum) FROM Pub.OrderLine GROUP BY LineNum ORDER BY LineNum ;
```



The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

```
SELECT LineNum, COUNT(LineNum) FROM Pub.OrderLine GROUP BY LineNum ORDER BY LineNum
```

The results pane shows the following data:

Linenum	count(Linenum)
1	126496
2	105120
3	84896
4	64000
5	43296
6	22016
7	1216

The status bar at the bottom indicates: /OESQL/OESQLDB/PUB/TABLE/ORDERLINE 1,84 / 84. The logs pane shows the following execution details:

```
Query 1 of 1, Rows read: 7, Elapsed time (seconds) - Total: 1.547, SQL query: 0.008, Reading results: 1.539  
Query 1 of 1, Rows read: 7, Elapsed time (seconds) - Total: 1.243, SQL query: 0.003, Reading results: 1.24  
Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 13.379, SQL query: 13.379, Reading results: 0  
Query 1 of 1, Rows read: 7, Elapsed time (seconds) - Total: 0.412, SQL query: 0.006, Reading results: 0.406  
Query 1 of 1, Rows read: 19, Elapsed time (seconds) - Total: 0.012, SQL query: 0.008, Reading results: 0.004  
Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.054, SQL query: 0.054, Reading results: 0  
Query 1 of 1, Rows read: 7, Elapsed time (seconds) - Total: 1.306, SQL query: 0.004, Reading results: 1.306
```

Check number of Records Read

The screenshot shows a Windows desktop with a blue background. A command prompt window titled "Administrator: Proenv - promon oesqldb" is open, displaying SQL performance statistics. The window shows various metrics such as DB Writes, BI Writes, AI Writes, Record Locks, Checkpoints, Record Waits, and Buffs Flushed. It also displays a summary of server and user activity, followed by a table of events and their rates.

Event	Commits	Total	Per Sec	Event	Undos	Total	Per Sec
Record Updates	0	0	0.0	Record Reads	447088	4657.2	
Record Creates	0	0	0.0	Record Deletes	0	0.0	
DB Writes	124	1.3	1.3	DB Reads	14532	151.4	
BI Writes	0	0.0	0.0	BI Reads	0	0.0	
AI Writes	0	0.0	0.0	Record Waits	0	0.0	
Record Locks	447049	4656.8	4656.8	Record Waits	0	0.0	
Checkpoints	0	0.0	0.0	Record Waits	0	0.0	

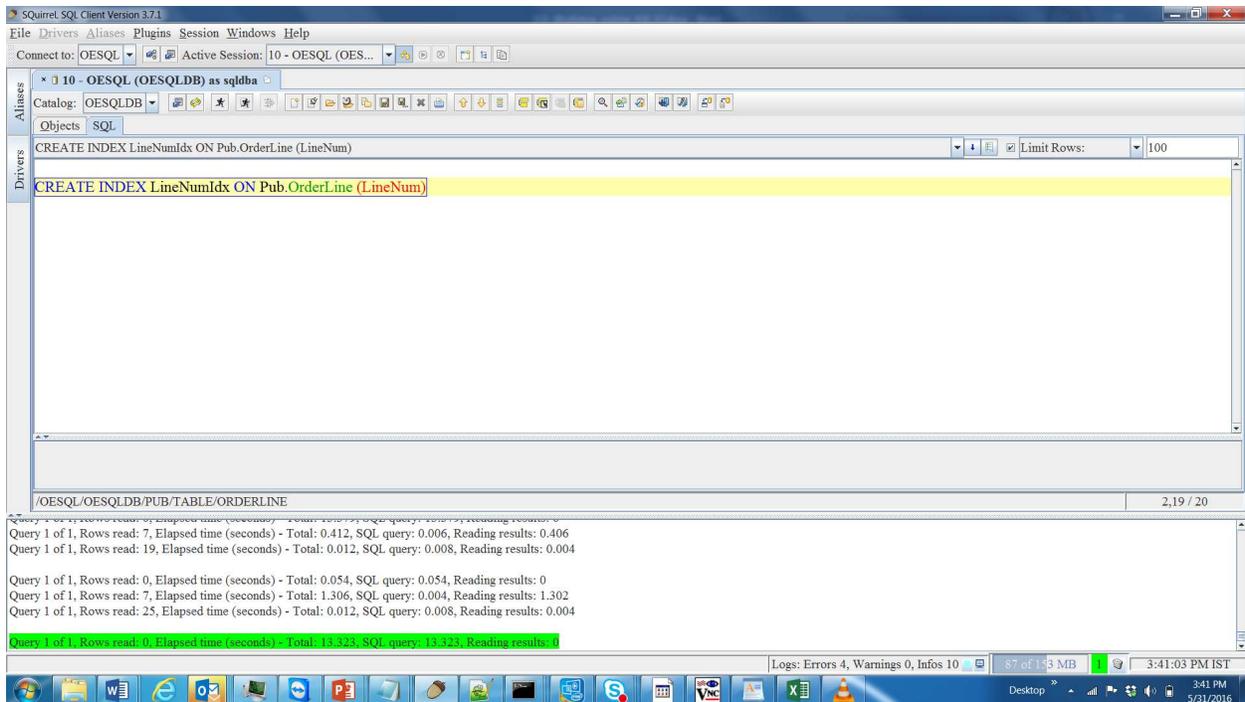
Now, check the query plan.

The screenshot shows the Squirrel SQL Client interface. The query editor contains the following SQL statement:

```
SELECT "_Description" FROM QPlan
```

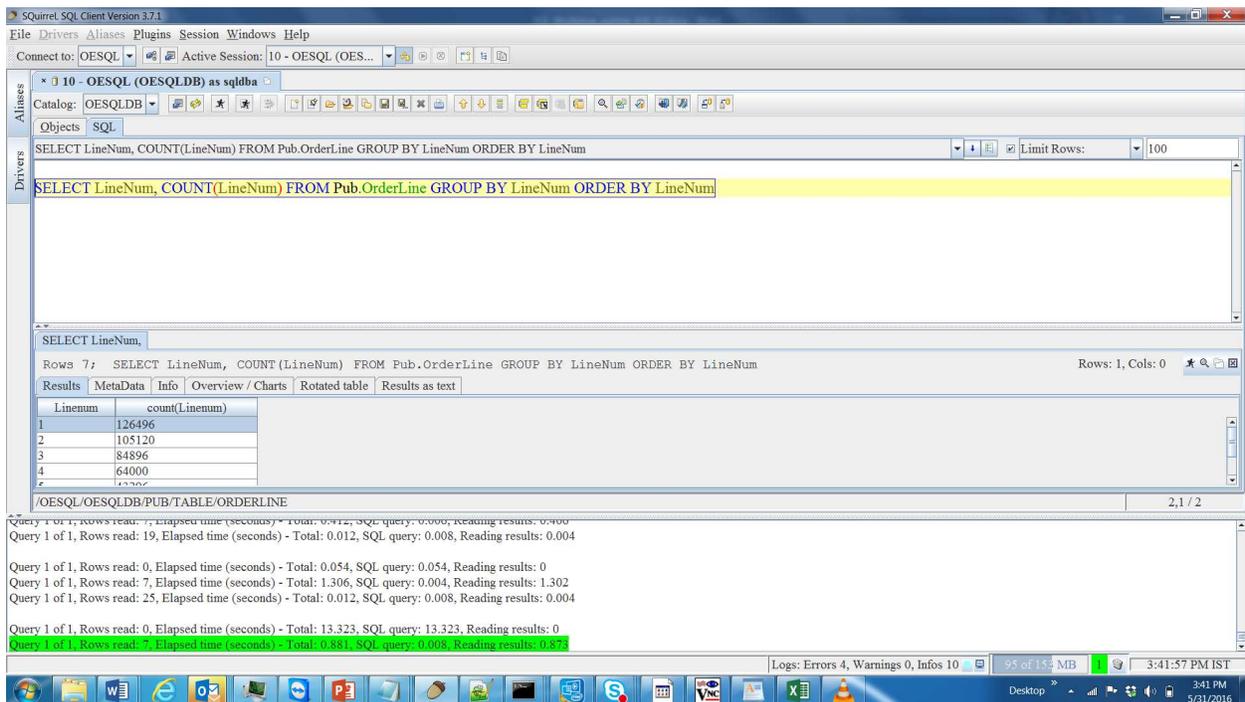
The query plan is displayed in the lower pane, showing the execution plan for the query. The plan includes a table scan for the QPlan table, followed by a group by operation and a sort operation.

```
SELECT COMMAND.
PROJECT [27] (
  SORT [26] IN TMP_TBL00000319(
    HASH_AGGREGATE [20] [ INTO TMP_TBL00000318 ] (
      PROJECT [16] (
        PUB_ORDERLINE. [3] (
          TABLE_SCAN
        )
        , PUB_ORDERLINE.Linenum
      )
      , PEXPR1
      , count ( PEXPR1 )
      , GROUP BY (
        , PEXPR1
      )
    )
    , SORT BY (
      , Sort project expression #0 == PEXPR1
    )
  )
)
/ OESQL/OESQLDB/PUB/TABLE/ORDERLINE
```

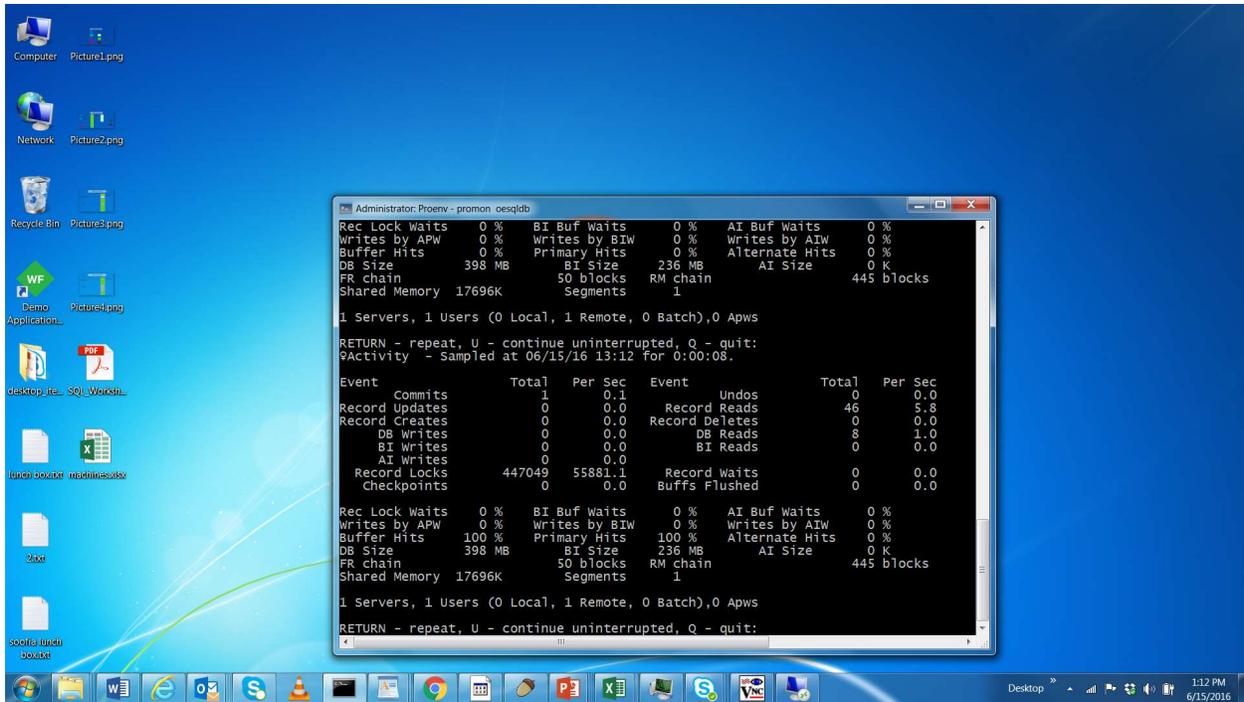



Now, run the same query again.

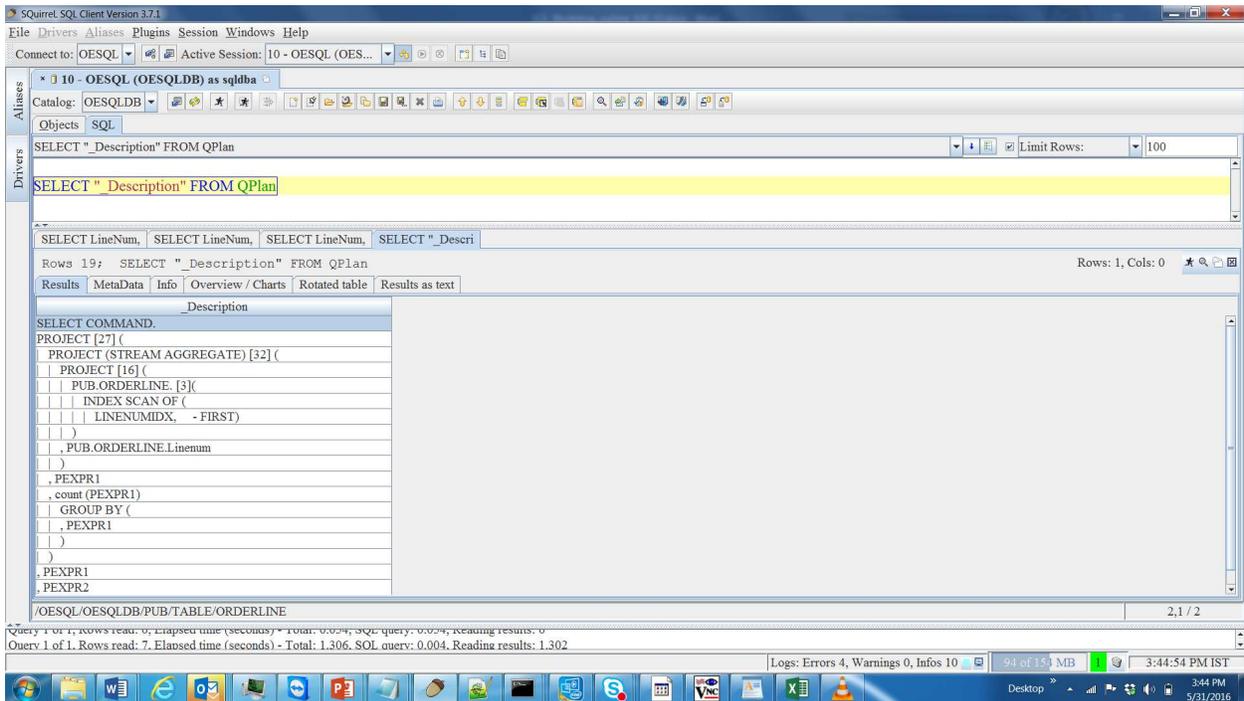
SELECT LineNum, COUNT(LineNum) FROM Pub.OrderLine GROUP BY LineNum ORDER BY LineNum ;



Now, check the number of records read.



Now, check the query plan



_Description

```
-----  
SELECT COMMAND.  
PROJECT [27] (  
| PROJECT (STREAM AGGREGATE) [32] (  
| | PROJECT [16] (  
| | | PUB.ORDERLINE. [3](  
| | | | INDEX SCAN OF (  
| | | | | LINENUMIDX, - FIRST)  
| | | | )  
| | | , PUB.ORDERLINE.Linum  
| | | )  
| | , PEXPR1  
| | , count (PEXP1)  
| | GROUP BY (  
| | | , PEXPR1  
| | | )  
| | )  
| )  
| , PEXPR1  
| , PEXPR2  
| )
```

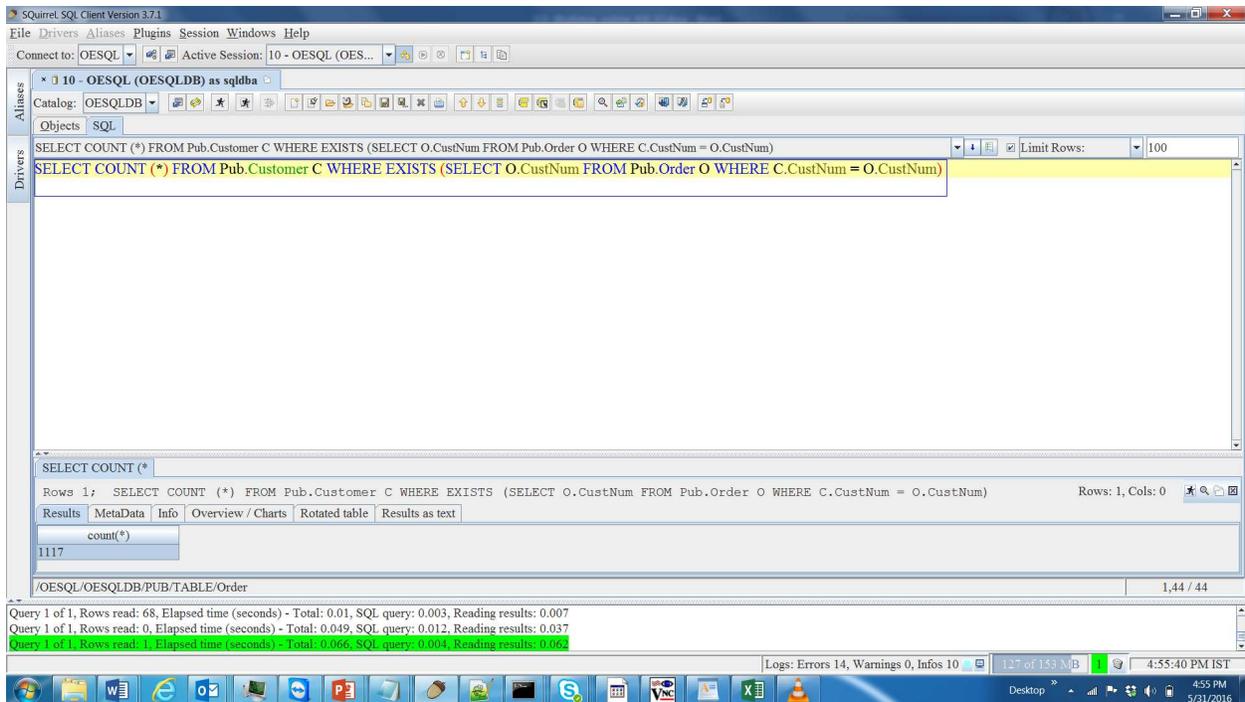
Observation: Did you notice the drastic difference in number of records read? In fact, in case of Stream Aggregation, it did not even read a single record. It is because, all the details which were required by query were already present in Index itself.

Chapter 7 - What happened to my subquery?

In this chapter you will learn how subqueries are transformed into other basic query operations by running below two queries containing subqueries and analyzing query plans generated. Knowing query transformations is important in troubleshooting the performance problems.

Let us start with simple co-related subquery

```
SELECT COUNT (*) FROM Pub.Customer C WHERE EXISTS (SELECT O.CustNum FROM Pub.Order O WHERE  
C.CustNum = O.CustNum);
```



```

SELECT COMMAND.
PROJECT [55] (
| PROJECT [54] (
| | PROJECT [53] distinct (
| | | JOIN [48][AUG_NESTED_LOOP-JOIN]
| | | [RHS-SORTED(-ASC-DUPS) ]{(
| | | | RESTRICT [10] (
| | | | | PROJECT [40] (
| | | | | | HASH AGGREGATE [36] [ INTO TMPTBL00000387 ] (
| | | | | | | JOIN [31][AUG_NESTED_LOOP-JOIN]
| | | | | | | [RHS-SORTED(-ASC-DUPS) ]{(
| | | | | | | | PROJECT [26] (
| | | | | | | | | PUB.O. [3](
| | | | | | | | | | INDEX SCAN OF (
| | | | | | | | | | | CustOrder, - FIRST)
| | | | | | | | | | | )

```

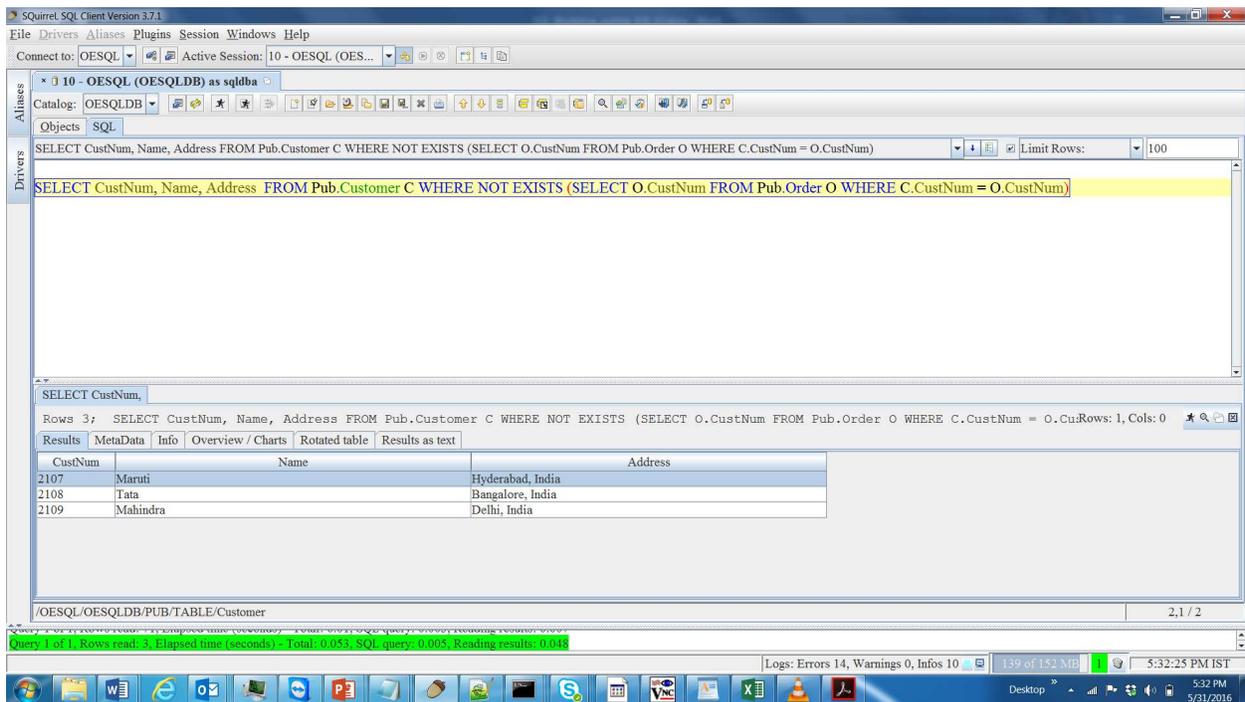
```

| | | | | | | | , PUB.O.CustNum
| | | | | | | | , PUB.O.rowid
| | | | | | | | )
| | | | | | | | ,
| | | | | | | | | (PEXPR1) = (PEXPR3)
| | | | | | | | -- above defines ANL left side keys <relop> right side keys.
| | | | | | | | ,
| | | | | | | | PROJECT [30] distinct (
| | | | | | | | | PROJECT [22] (
| | | | | | | | | | PUB.C. [23] (
| | | | | | | | | | | INDEX SCAN OF (
| | | | | | | | | | | CustNum,
| | | | | | | | | | | | (PUB.C.CustNum) = (null))
| | | | | | | | | | )
| | | | | | | | | | , PUB.C.CustNum
| | | | | | | | | | )
| | | | | | | | | | , PEXPR1
| | | | | | | | | | )
| | | | | | | | | | )
| | | | | | | | | | , pro_nonempty (*)
| | | | | | | | | | , PEXPR3
| | | | | | | | | | GROUP BY (
| | | | | | | | | | | , PEXPR3
| | | | | | | | | | )
| | | | | | | | | | )
| | | | | | | | | | , PEXPR1
| | | | | | | | | | , PEXPR2
| | | | | | | | | | )
| | | | | | | | | | )
| | | | | | | | | | (0) < (PEXPR1)
| | | | | | | | | | )
| | | | | | | | | | ,
| | | | | | | | | | (PEXPR2) = (PEXPR3)
| | | | | | | | | | -- above defines ANL left side keys <relop> right side keys.
| | | | | | | | | | ,
| | | | | | | | | | PROJECT [46] (
| | | | | | | | | | | PUB.C. [1] (
| | | | | | | | | | | | INDEX SCAN OF (
| | | | | | | | | | | | CustNum,
| | | | | | | | | | | | | (PUB.C.CustNum) = (null))
| | | | | | | | | | | )
| | | | | | | | | | | , PUB.C.CustNum
| | | | | | | | | | | , PUB.C.rowid
| | | | | | | | | | | )
| | | | | | | | | | | )
| | | | | | | | | | | , PEXPR3
| | | | | | | | | | | , PEXPR4
| | | | | | | | | | | )
| | | | | | | | | | | , count (*)
| | | | | | | | | | | )
| | | | | | | | | | | , PEXPR1
| | | | | | | | | | | )

```

Let us look at second subquery transformation

SELECT CustNum, Name, Address FROM Pub.Customer C WHERE NOT EXISTS (SELECT O.CustNum FROM Pub.Order O WHERE C.CustNum = O.CustNum) ;



```

SELECT COMMAND.
PROJECT [63] (
| PROJECT [62] distinct (
| | JOIN [55][AUG_NESTED_LOOP-JOIN]
| | | [RHS-SORTED(-ASC-DUPS) ](
| | | RESTRICT [12] (
| | | | PROJECT [43] (
| | | | | PROJECT (STREAM AGGREGATE) [251] (
| | | | | | OUTER JOIN [33][AUG_NESTED_LOOP-JOIN][OJ SQL Gen]
| | | | | | | [RHS-SORTED(-ASC-DUPS) ](
| | | | | | | PROJECT [32] distinct (
| | | | | | | | PROJECT [24] (
| | | | | | | | | PUB.C. [25](
| | | | | | | | | | INDEX SCAN OF (
| | | | | | | | | | CustNum, - FIRST)

```

```

| | | | | | | | )
| | | | | | | | , PUB.C.CustNum
| | | | | | | | )
| | | | | | | | , PEXPR1
| | | | | | | | )
| | | | | | | | ,
| | | | | | | | ( PEXPR1 ) = ( PEXPR2 )
| | | | | | | | -- above defines ANL left side keys <relop> right side keys.
| | | | | | | | ,
| | | | | | | | PROJECT [28] (
| | | | | | | | PUB.O. [5](
| | | | | | | | INDEX SCAN OF (
| | | | | | | | CustOrder,
| | | | | | | | | (PUB.O.CustNum) = (null))
| | | | | | | | )
| | | | | | | | , PUB.O.CustNum
| | | | | | | | , PUB.O.rowid
| | | | | | | | )
| | | | | | | | )
| | | | | | | | , count (PEXP3)
| | | | | | | | , PEXPR1
| | | | | | | | GROUP BY (
| | | | | | | | , PEXPR1
| | | | | | | | )
| | | | | | | | )
| | | | | | | | , PEXPR1
| | | | | | | | , PEXPR2
| | | | | | | | )
| | | |
| | | | (0) = (PEXP1) [Null-Null EQ]
| | | | )
| | | | ,
| | | | ( PEXPR2 ) = ( PEXPR3 )
| | | | -- above defines ANL left side keys <relop> right side keys.
| | | | ,
| | | | PROJECT [53] (
| | | | | PUB.C. [3](
| | | | | | INDEX SCAN OF (
| | | | | | CustNum,
| | | | | | | (PUB.C.CustNum) = (null) [Null-Null EQ])
| | | | | | )
| | | | | , PUB.C.CustNum
| | | | | , PUB.C.Name
| | | | | , PUB.C.Address
| | | | | , PUB.C.rowid
| | | | | )
| | | | )
| | | | , PEXPR3
| | | | , PEXPR4
| | | | , PEXPR5
| | | | , PEXPR6
| | | | )
, PEXPR1
, PEXPR2
, PEXPR3

```

)

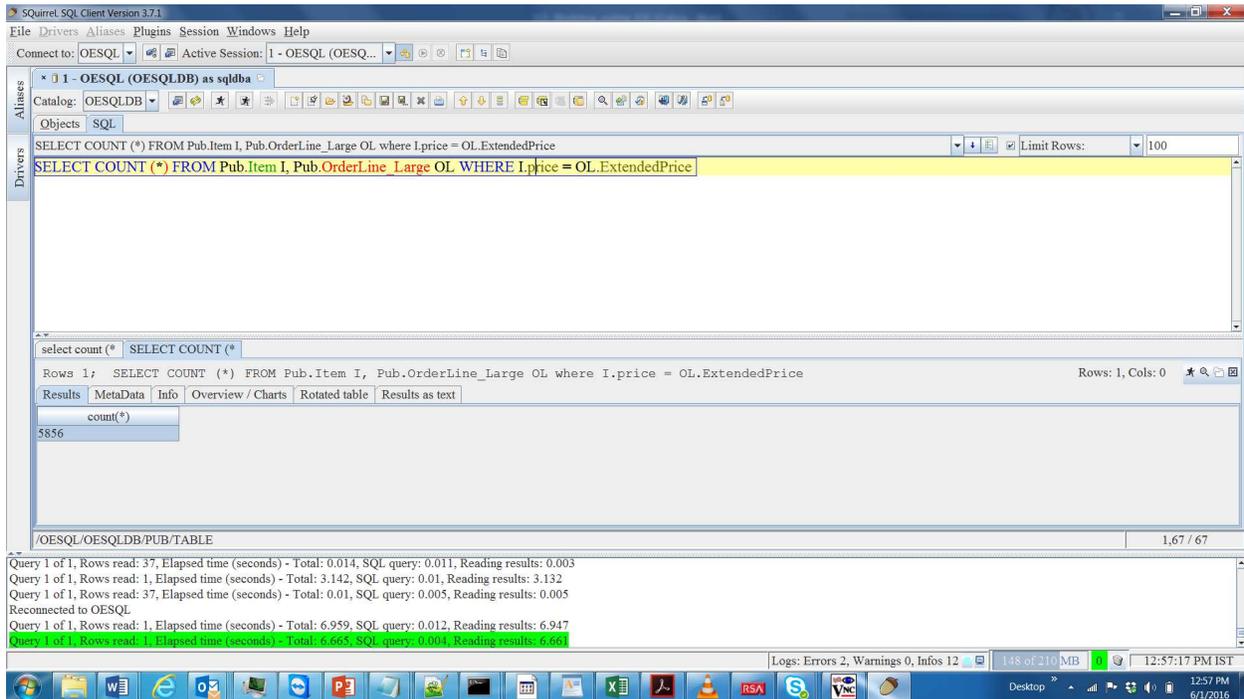
Chapter 8 - Importance of Statistics

In this chapter, you will learn the different types of Statistics available in OE-SQL and their importance in generating good query plans to have better performance. Basically, statistics provides insights on table, index and column data which will be used by OE-SQL optimizer to come up with optimal query plans.

- a. **Importance of Table statistics** – To explain the importance of these statistics, you need to run provided join query and check the number of IOs using promon utility. And then, you need to run the table statistics command on tables used by the provided join query and run the same query again to verify the difference between number of IOs took place with former case.

Execute the following query and check the time taken for execution.

SELECT COUNT () FROM Pub.Item I, Pub.OrderLine_Large OL where I.price = OL.ExtendedPrice;*



The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

```
SELECT COUNT (*) FROM Pub.Item I, Pub.OrderLine_Large OL WHERE I.price = OL.ExtendedPrice
```

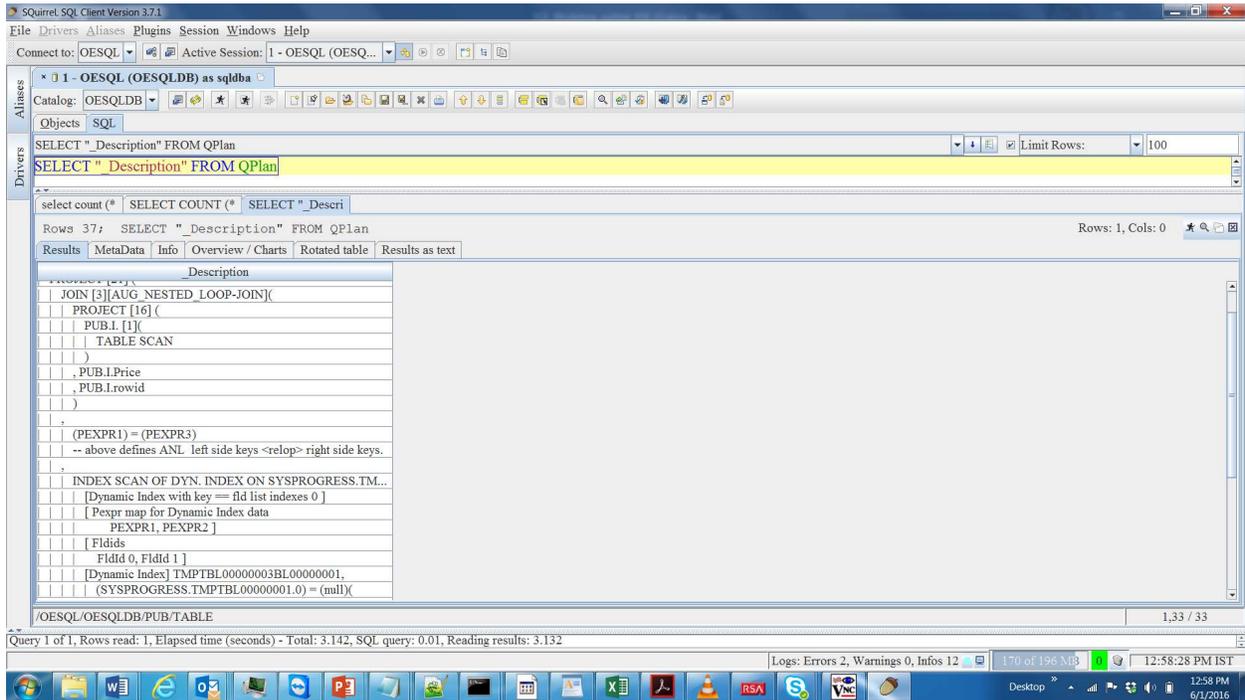
The query results are shown in a table with the following data:

count(*)
5856

The bottom status bar shows the following query execution statistics:

```
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.014, SQL query: 0.011, Reading results: 0.003  
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 3.142, SQL query: 0.01, Reading results: 3.132  
Query 1 of 1, Rows read: 37, Elapsed time (seconds) - Total: 0.01, SQL query: 0.005, Reading results: 0.005  
Reconnected to OESQL  
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 6.959, SQL query: 0.012, Reading results: 6.947  
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 6.665, SQL query: 0.004, Reading results: 6.661
```

Now, check the query plan. You can notice that, DIDX is created on the large table and this caused query to run slower.



_Description

```

-----
SELECT COMMAND.
PROJECT [22] (
| PROJECT [21] (
| | JOIN [3][AUG_NESTED_LOOP-JOIN](
| | | PROJECT [16] (
| | | | PUBL.I. [1](
| | | | | TABLE SCAN
| | | | )
| | | | , PUBL.I.Price
| | | | , PUBL.I.rowid
| | | )
| | | ,
| | | (PEXPR1) = (PEXPR3)
| | | -- above defines ANL left side keys <
| | | ,
| | | INDEX SCAN OF DYN. INDEX ON SYSPROGRES
| | | | [Dynamic Index with key == fld lis
| | | | [ Pexpr map for Dynamic Index data
| | | | PEXPR1, PEXPR2 ]
| | | | [Fldids
| | | | Fldid 0, Fldid 1]
| | | | [Dynamic Index] TMPTBL00000003BL00
| | | | (SYSPROGRESS.TMPTBL00000001.0)
| | | | PROJECT [19] (
| | | | | PUBL.OL. [2](
| | | | | | TABLE SCAN

```

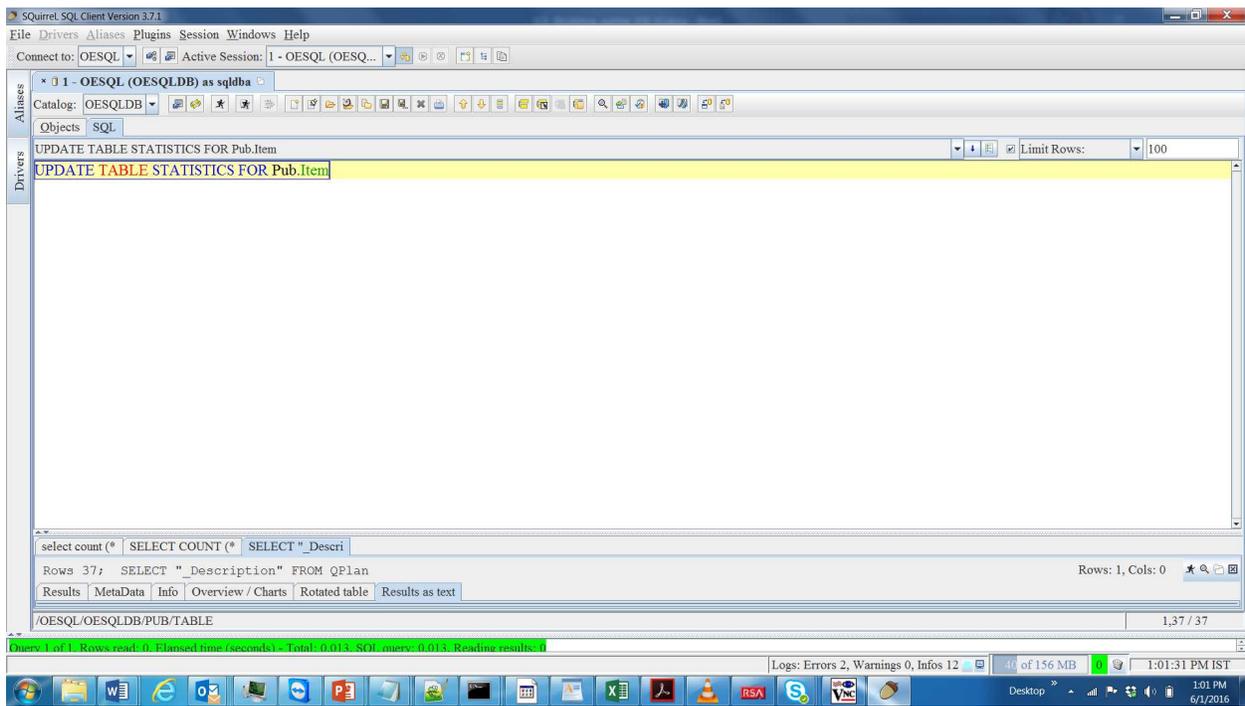
```

/ / / / )
/ / / / , PUB.OL.ExtendedPrice
/ / / / , PUB.OL.rowid
/ / / / )
/ / /
/ / / )
/ / )
/ , count (*)
/ )
, PEXPR1
)

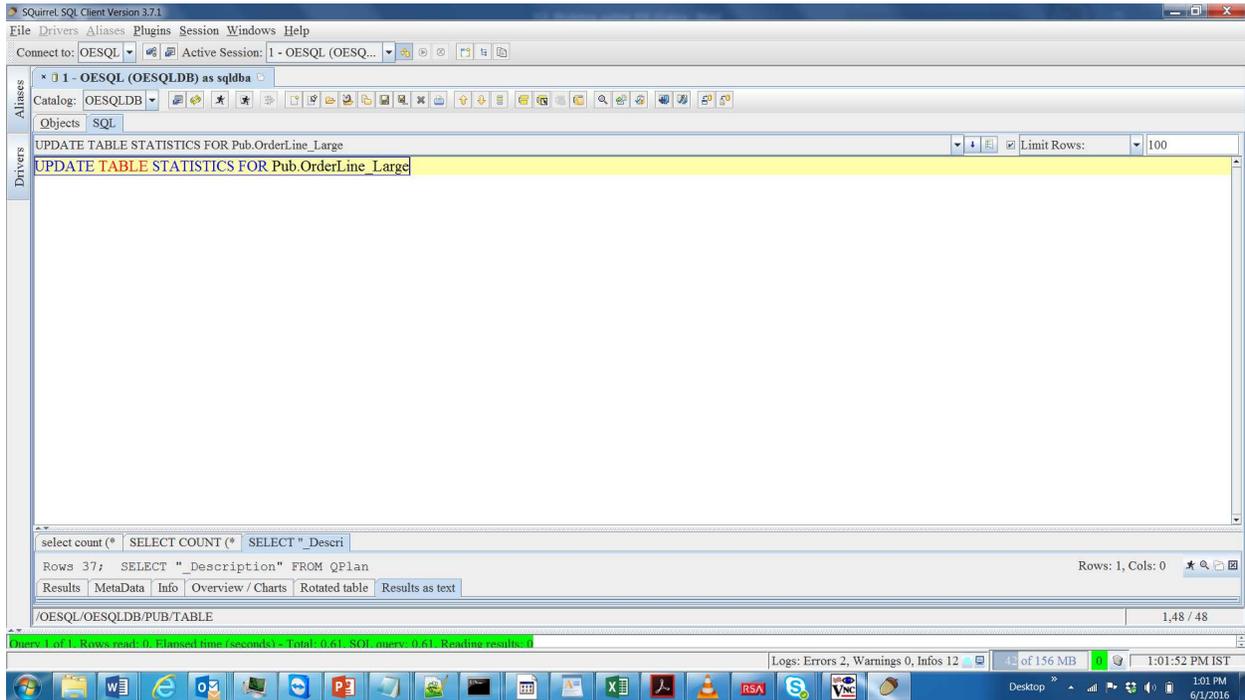
```

Now, let us update the table statistics for the 2 tables involved in the join.

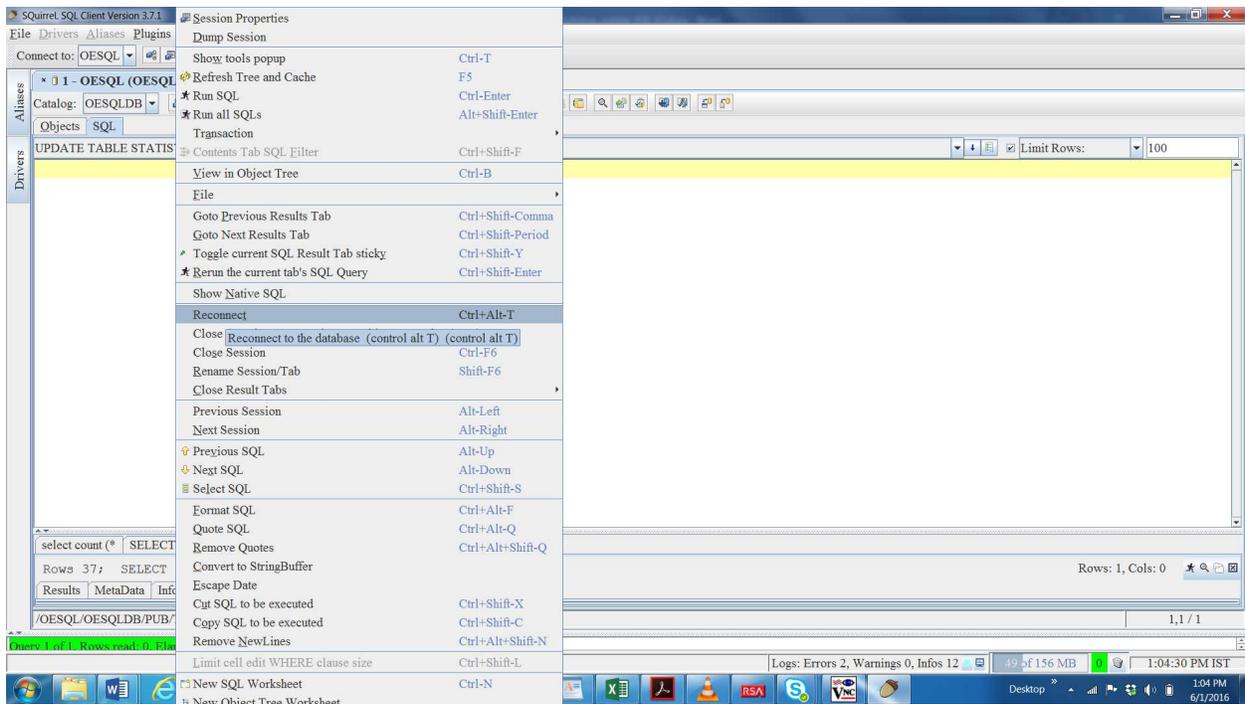
UPDATE TABLE STATISTICS FOR Pub.Item;



UPDATE TABLE STATISTICS FOR Pub.OrderLine_Large;

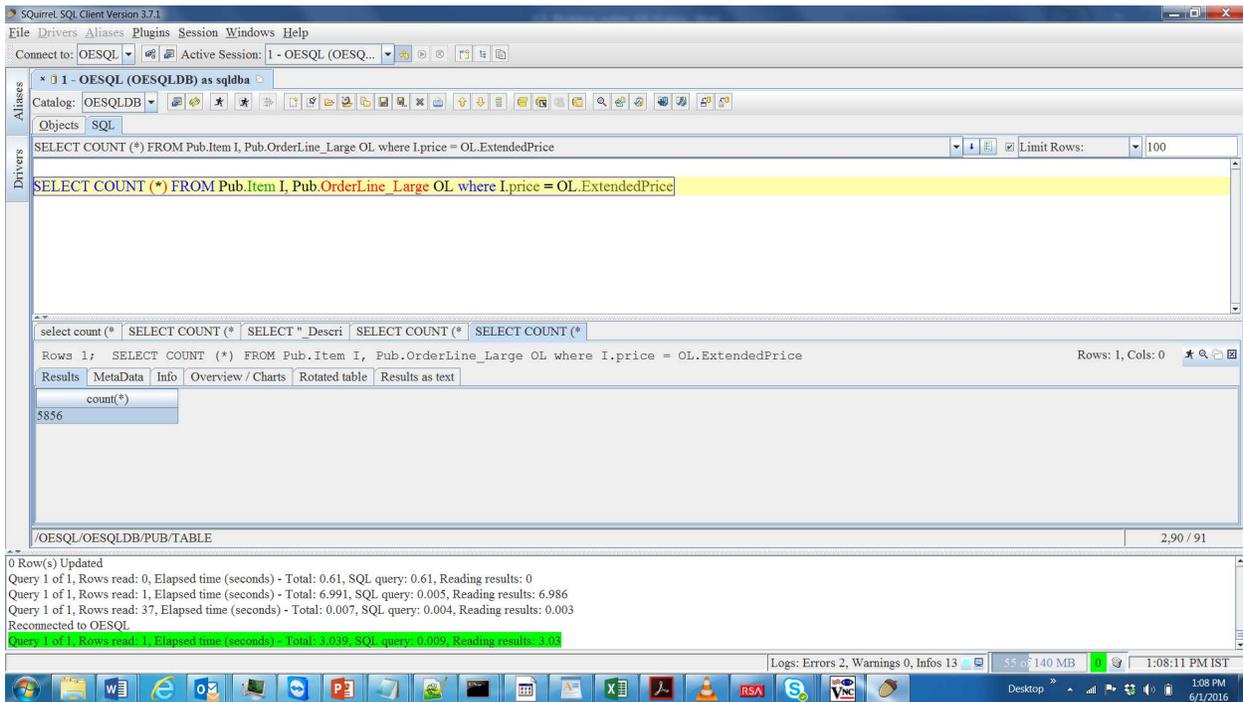


Need to Reconnect to the database in order to see the effect of Table Statistics. For this, go to the "Session" tab of Squirrel client and click on "Reconnect".

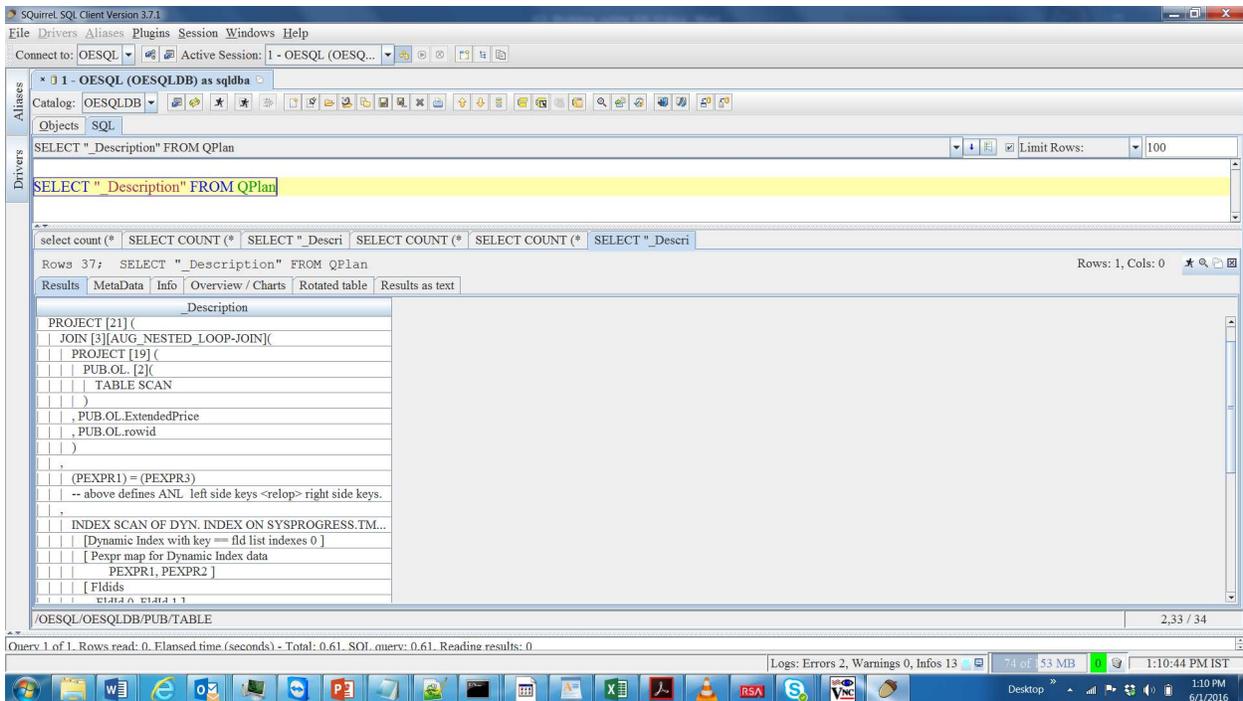


Now, run the same query again and notice the time difference.

SELECT COUNT () FROM Pub.Item I, Pub.OrderLine_Large OL where I.price = OL.ExtendedPrice;*



Now, check what caused the change in execution time by almost half!! For that, let us check the query plan this time for this same query.



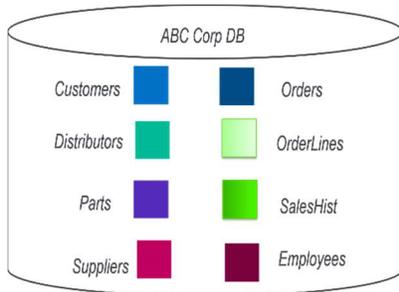
_Description

```
-----
SELECT COMMAND.
PROJECT [22] (
| PROJECT [21] (
| | JOIN [3][AUG_NESTED_LOOP-JOIN](
| | | PROJECT [19] (
| | | | PUB.OL. [2](
| | | | | TABLE SCAN
| | | | )
| | | , PUB.OL.ExtendedPrice
| | | , PUB.OL.rowid
| | | )
| | ,
| | | (PEXPR1) = (PEXPR3)
| | | -- above defines ANL left side keys <
| | ,
| | | INDEX SCAN OF DYN. INDEX ON SYSPROGRES
| | | | [Dynamic Index with key == fld lis
| | | | [ Pexpr map for Dynamic Index data
| | | | PEXPR1, PEXPR2 ]
| | | | [ Fldids
| | | | FldId 0, FldId 1 ]
| | | | [Dynamic Index] TMPTBL00000002BL00
| | | | | (SYSPROGRESS.TMPTBL00000001.0)
| | | | PROJECT [16] (
| | | | | PUB.I. [1](
| | | | | | TABLE SCAN
| | | | | )
| | | | , PUB.I.Price
| | | | , PUB.I.rowid
| | | | )
| | |
| | | )
| | )
| , count (*)
| )
, PEXPR1
```

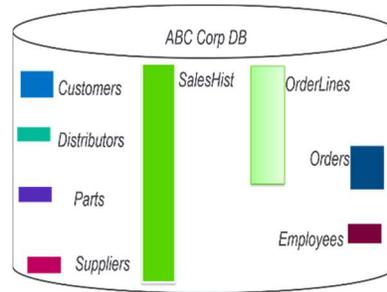
Observation:

We can clearly notice that, having table statistics helped SQL Optimizer to select the join order which is more optimal. That is, SQL optimizer, now identified that creation of Dynamic Index on the Large table amounts to excessive cost, hence, it selected the join order where the Large table is kept on left side and Dynamic Index is created on smaller table.

Before Table Statistics:



After Table Statistics:



- b. **Importance of Index statistics** - To explain the importance of these statistics, you need to run provided join query and check the number of IOs using promon utility. And then, you need to run the index statistics command on tables used by the provided join query and run the same query again to verify the difference between number of IOs took place with former case.

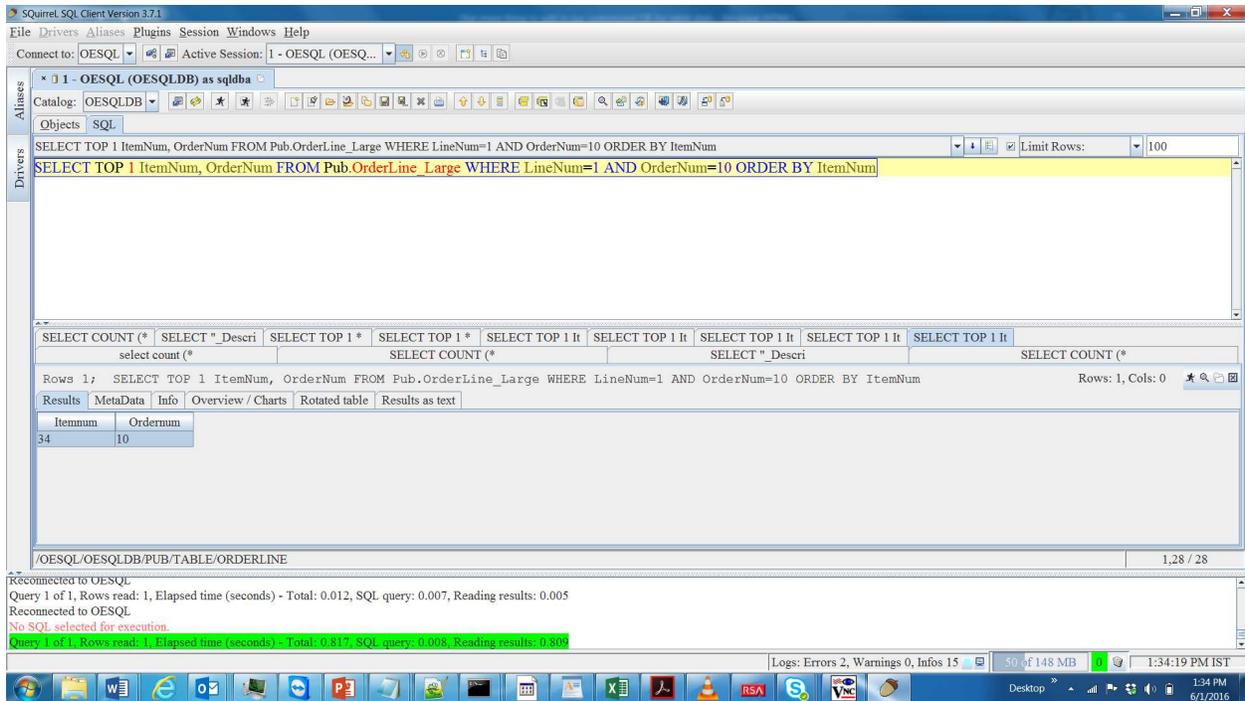
Now, let us see how index stats effects the performance of a query.

Table Pub.OrderLine_Large already has following 2 indexes.

- i) LineNum_Large created on LineNum column
- ii) OrderNum_Large created on OrderNum colum.

Now, run the below query.

SELECT TOP 1 ItemNum, OrderNum FROM Pub.OrderLine_Large WHERE LineNum=1 AND OrderNum=10 ORDER BY ItemNum;



Now, check the number of Records read

The screenshot shows the Squirrel SQL Client interface. A performance monitor window titled 'Administrator: Proenv - promon oesqldb' is open, displaying various system metrics. The 'Event' section shows 'Record Reads' with a total of 379488 and a per-second rate of 23718.0. Other metrics include Record Updates, Record Creates, DB Writes, BI Writes, AI Writes, Record Locks, Checkpoints, Record Waits, and Buffs Flushed.

In the background, the SQL client shows a query result for 'SELECT TOP 1 ItemNum, OrderNum FROM Pub.OrderLine_Large'. The result table has two columns: 'Itemnum' with value 34 and 'Ordernum' with value 10.

The status bar at the bottom indicates 'Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.012, SQL query: 0.008, Reading results: 0.809'.

Now, check the query plan. We can notice that it selected index which is created on LineNum column of Pub.OrderLine_Large table.

The screenshot shows the Squirrel SQL Client interface with a query plan window open. The query is 'SELECT "Description" FROM QPlan'. The query plan details show an 'INDEX SCAN OF (LINENUM_LARGE, (PUB.ORDERLINE_LARGE.LineNum) = (1))' operation. The plan also shows the table 'PUB.ORDERLINE_LARGE' and the columns 'Itemnum' and 'Ordernum'.

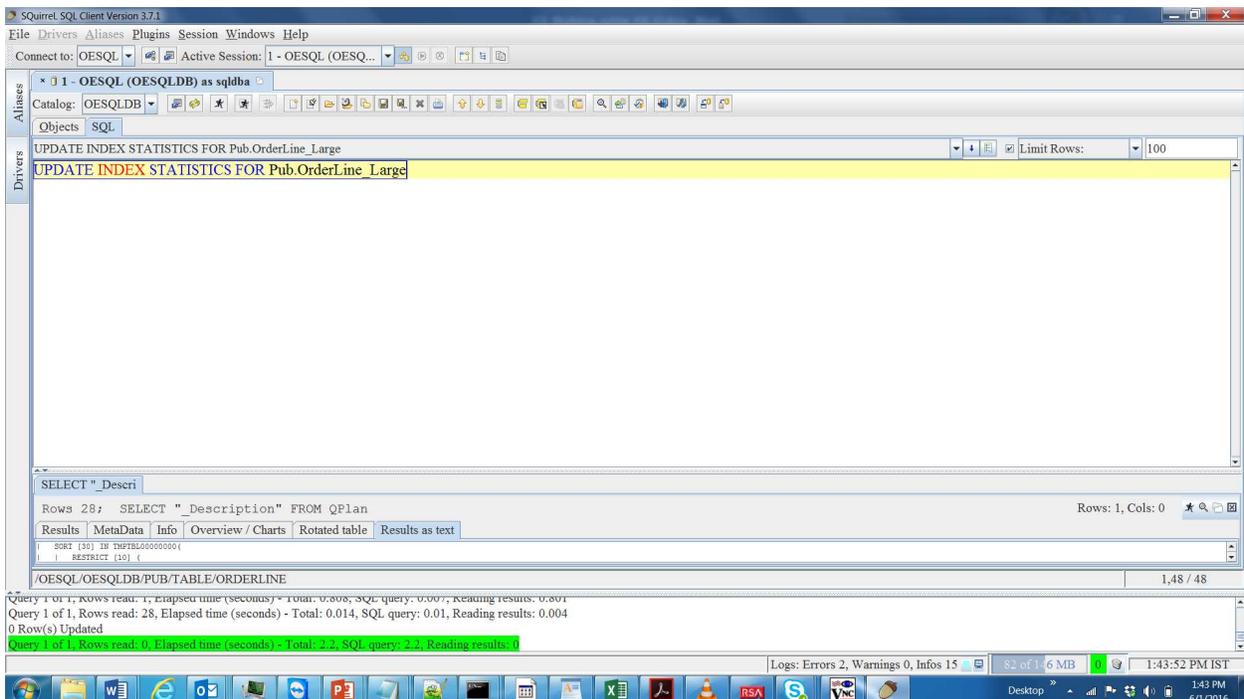
The status bar at the bottom indicates 'Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.817, SQL query: 0.008, Reading results: 0.809'.

_Description

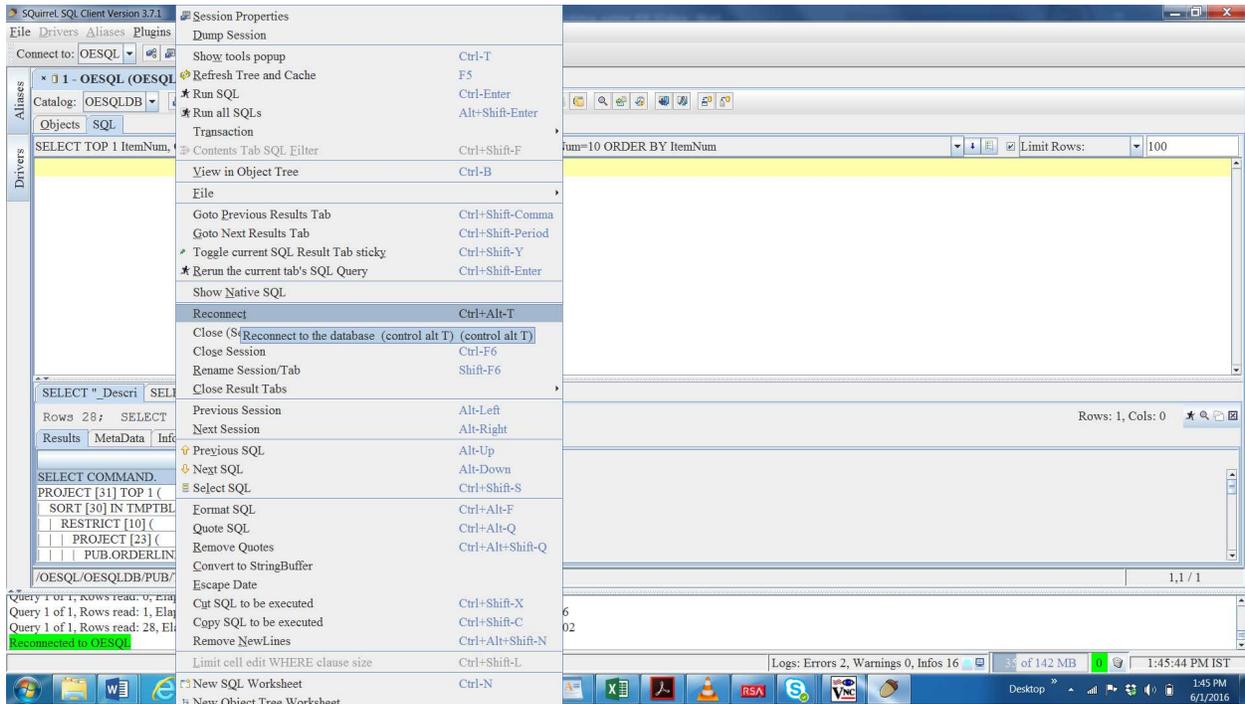
```
-----  
SELECT COMMAND.  
PROJECT [31] TOP 1 (  
  | SORT [30] IN TMPTBL0000000(  
  | | RESTRICT [10] (  
  | | | PROJECT [23] (  
  | | | | PUB.ORDERLINE_LARGE. [2](  
  | | | | | INDEX SCAN OF (  
  | | | | | | LINENUM_LARGE,  
  | | | | | | (PUB.ORDERLINE_LARGE.L  
  | | | | | )  
  | | | | ,PUB.ORDERLINE_LARGE.Itemnum  
  | | | | ,PUB.ORDERLINE_LARGE.Ordernum  
  | | | )  
  | | )  
  | | | (PEXPR2) = (10)  
  | | | Evaluation callback list(  
  | | | | col id# 2  
  | | | )  
  | | )  
  | | SORT BY (  
  | | | , Sort project expression #0 == PEXPR1  
  | | )  
  | | , PEXPR1  
  | | , PEXPR2  
  | | )  
  | | , PEXPR1  
  | | , PEXPR2  
  | )  
)
```

Now, update the Index Statistics for table Pub.OrderLine_Large.

UPDATE INDEX STATISTICS FOR Pub.OrderLine_Large;

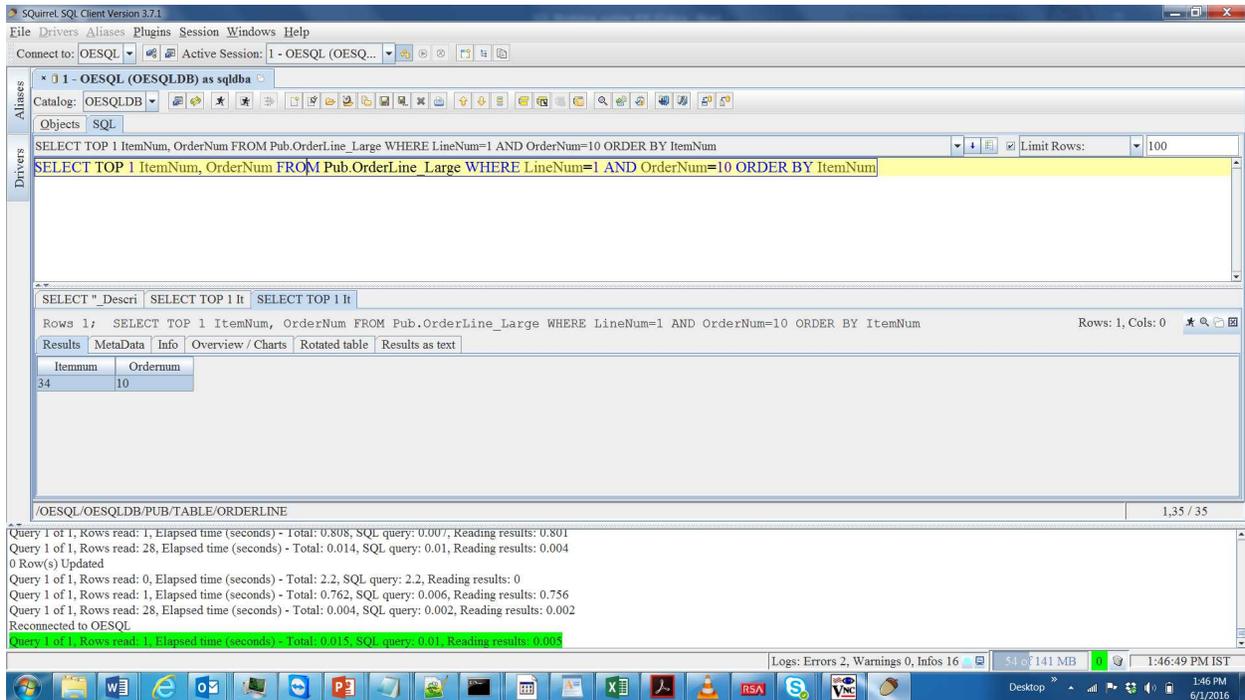


Reconnect to Database in order to get the effects of Index Statistics

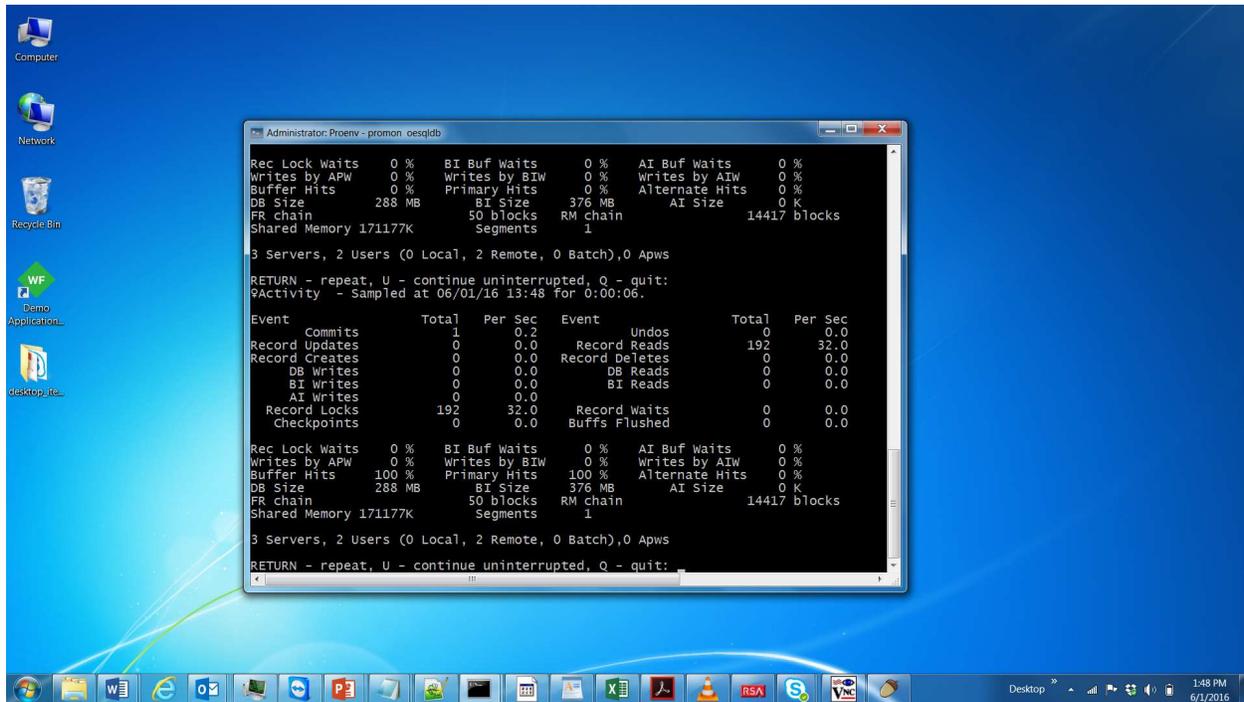


Now, run the same query again.

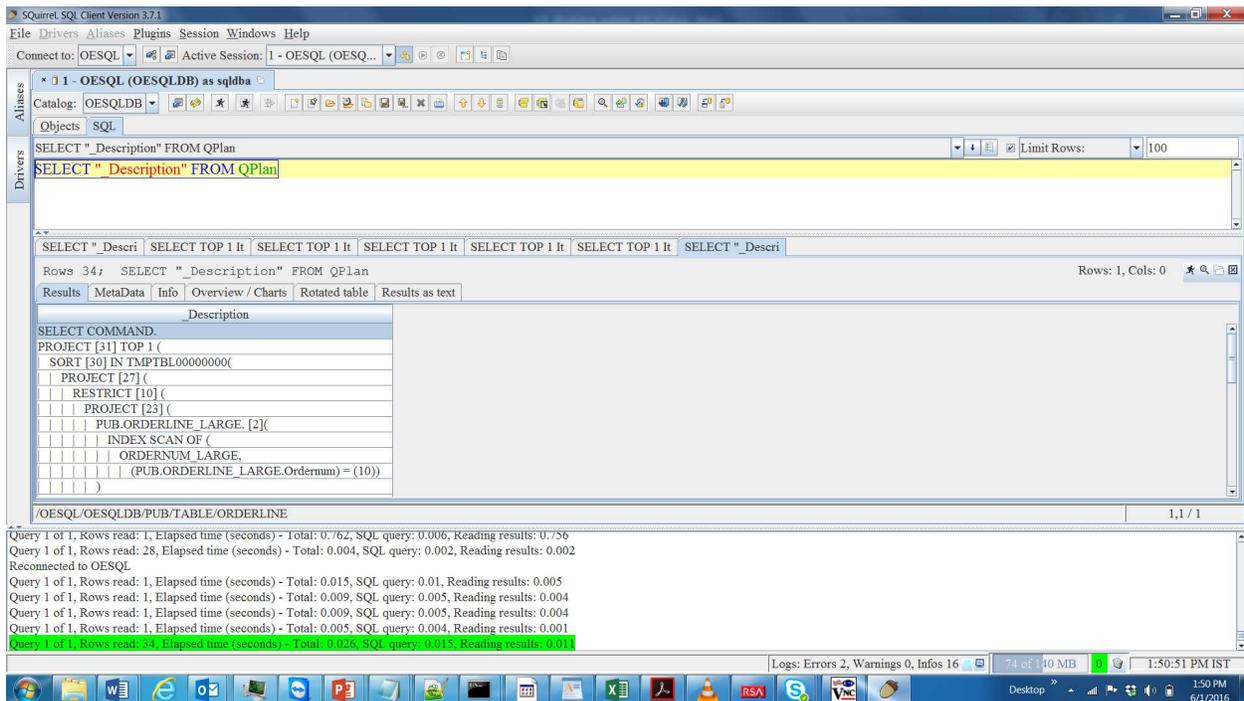
SELECT TOP 1 ItemNum, OrderNum FROM Pub.OrderLine_Large WHERE LineNum=1 AND OrderNum=10 ORDER BY ItemNum;



Now, check the number of Records Read in promon Window.



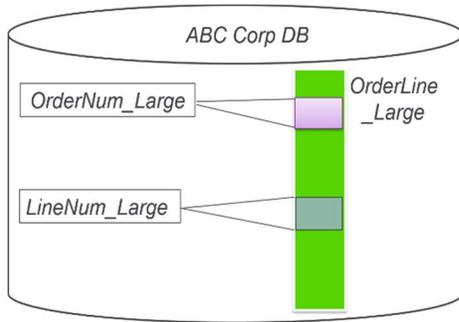
Now, check the query plan for this same query.



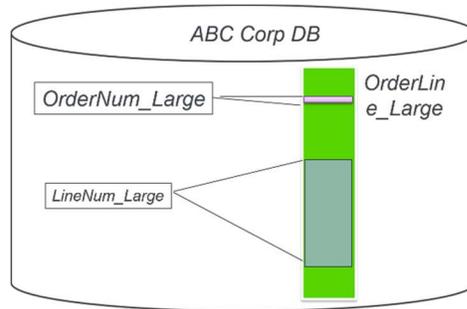
Observation:

What caused the dramatic improvement in query execution time and number of records read!!! With the help of index statistics SQL Optimizer could differentiate and compare the costs of two different indexes and selected the index on OrderNum (OrderNum_Large) as it contains more number of unique values and hence filters more rows.

Before Index Statistics



After Index Statistics

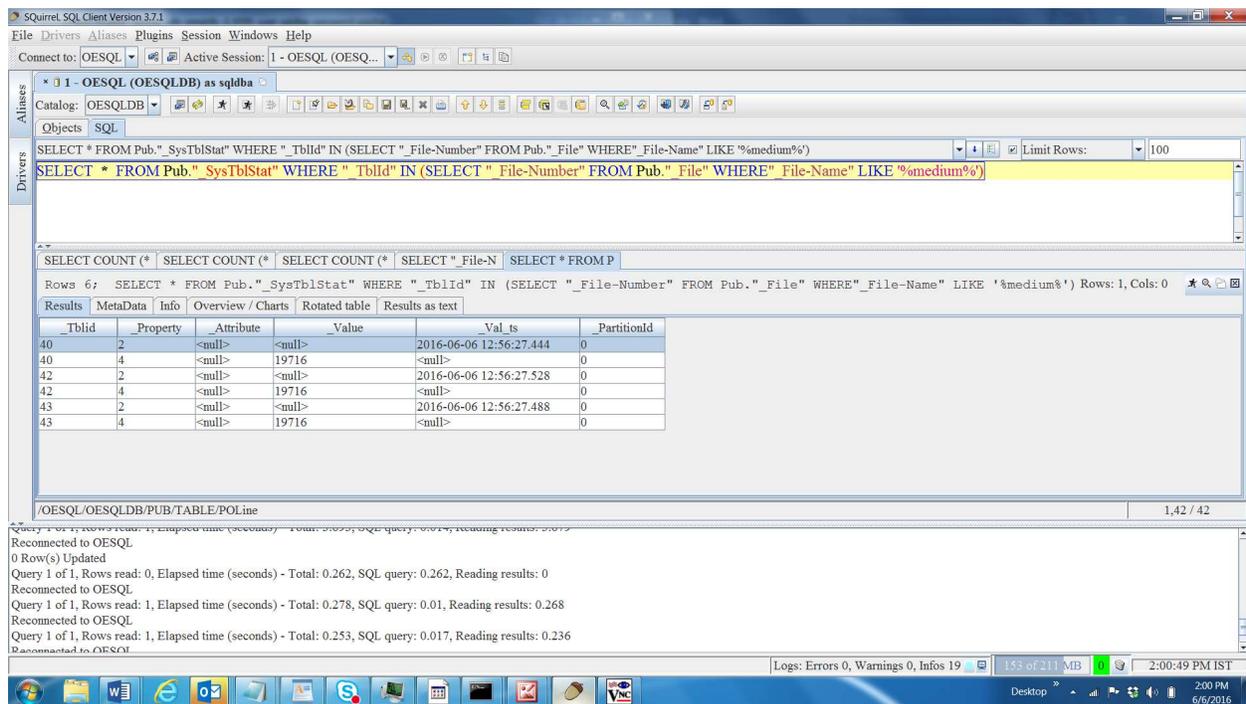


- c. **Importance of Column statistics** – There are 2 flavors of column statistics, one which just updates column statistics for columns which has index on them and the other one, which collects column statistics for all the columns of the table. We will use the flavor which collects statistics for all the columns to explain the importance of these statistics. You need to run provided join query and check the number of IOs using promon utility. And then, you need to run the column statistics command on tables used by the provided join query and run the same query again to verify the difference between number of IOs took place with former case.

First, let us check if the Table Statistics exists for the tables which we will be using in our example query to demonstrate the importance of column statistics. If you think, table and index statistics are enough, then you may need to re-think about this perspective after this illustration.

Below query checks if the table statistics are present for tables, Pub.Order_Medium, Pub.OrderLine_Medium and Pub.POLine_Medium.

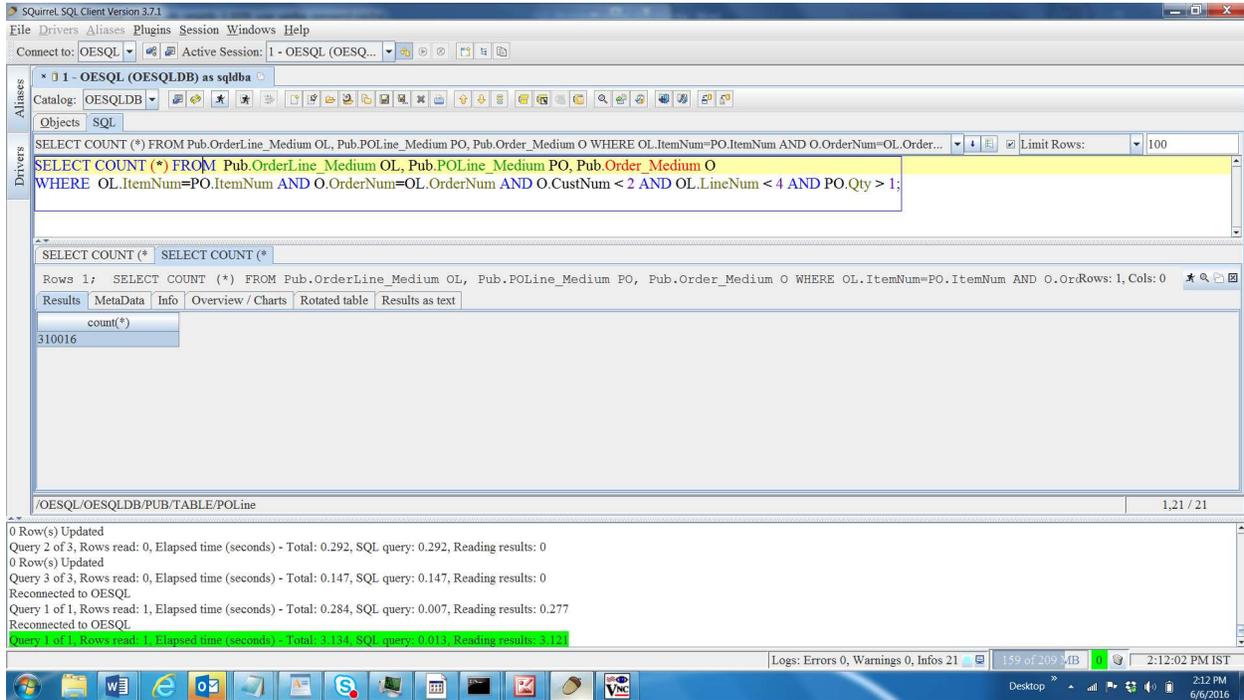
```
SELECT * FROM Pub."_SysTblStat" WHERE "_TblId" IN (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" LIKE '%medium%');
```



We can notice that each table has same number of rows (19716).

Now, let us run the below query to see the effect of column statistics.

```
SELECT COUNT (*) FROM Pub.OrderLine_Medium OL, Pub.POLine_Medium PO, Pub.Order_Medium O
WHERE OL.ItemNum=PO.ItemNum AND O.OrderNum=OL.OrderNum AND O.CustNum < 2 AND OL.LineNum <
4 AND PO.Qty > 1;
```



Now, check the query plan.

```

    _Description
    -----
SELECT COMMAND.
PROJECT [53] (
| PROJECT [52] (
| | JOIN [5][AUG_NESTED_LOOP-JOIN](
| | | JOIN [3][AUG_NESTED_LOOP-JOIN](
| | | | RESTRICT [55] (
| | | | | PROJECT [42] (
| | | | | | PUB.OL. [1](
| | | | | | | TABLE SCAN
| | | | | | )
| | | | | , PUB.OL.Itemnum
| | | | | , PUB.OL.Ordernum
| | | | | , PUB.OL.Lineum
| | | | | , PUB.OL.rowid
| | | | | )
| | | | | )
| | | | | (PEXPR3) < (4)
| | | | | Evaluation callback list(
| | | | | | col id# 3
| | | | | )
| | | | | )
| | | | | ,

```



```

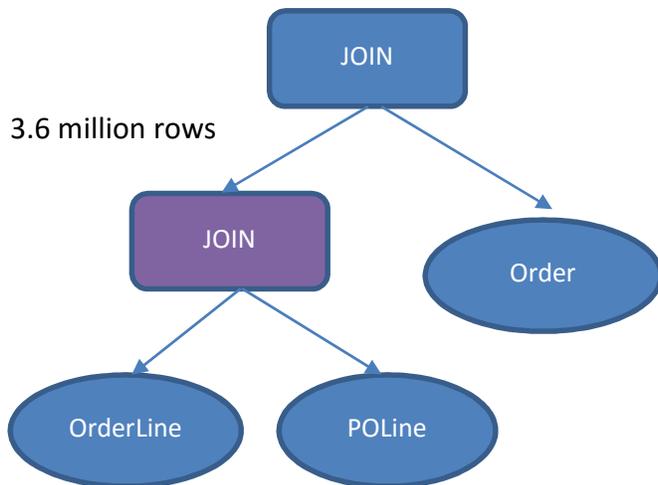
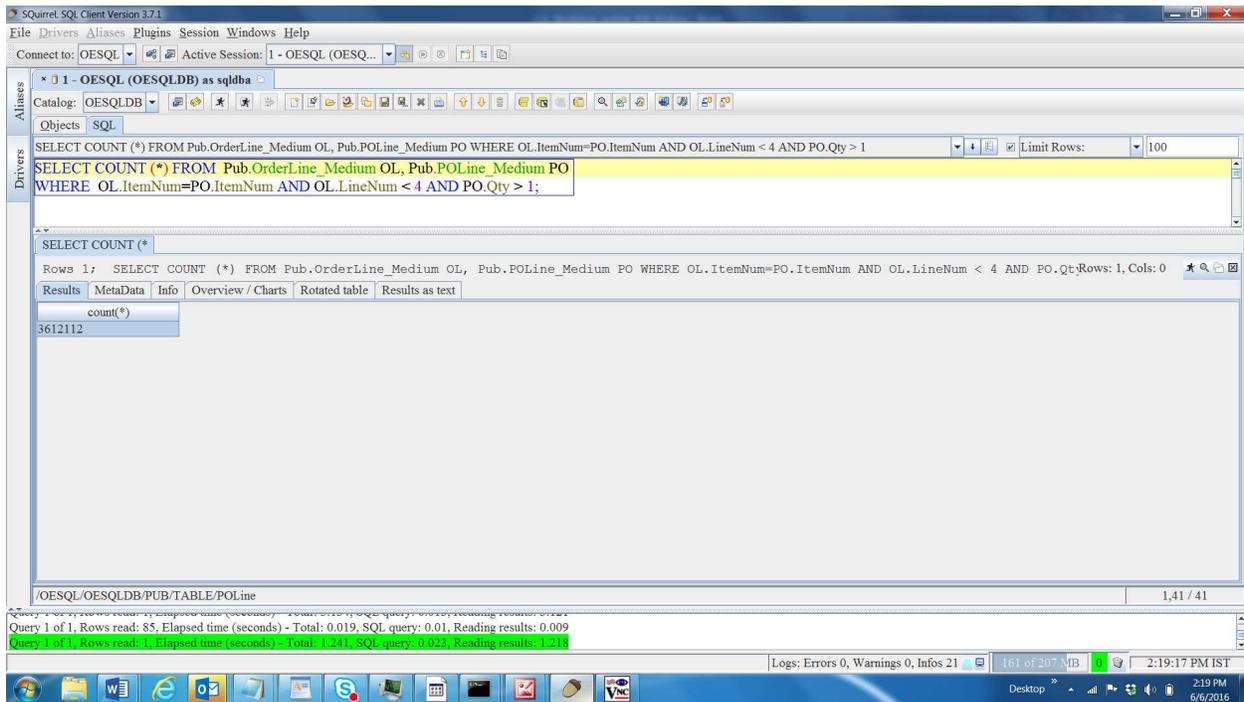
/ / / / )
/ / /
/ / / )
/ / )
/ , count (*)
/ )
, PEXPR1

```

Reason for Slower execution:

If we analyze, the intermediate join (between Pub.OrderLine_Medium and Pub.POLine_Medium) produced around 3.6 million rows!!!

SELECT COUNT() FROM Pub.OrderLine_Medium OL, Pub.POLine_Medium PO
WHERE OL.ItemNum=PO.ItemNum AND OL.LineNum<4 AND PO.Qty>1;*

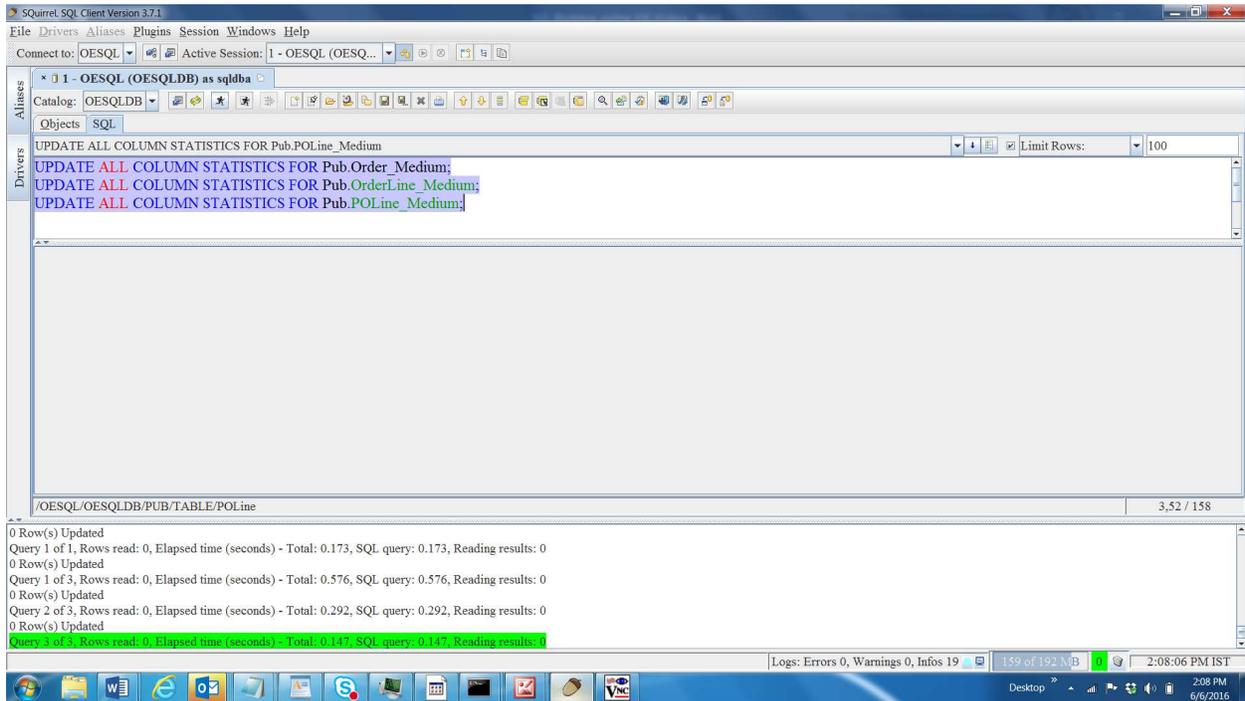


Now, run the column statistics for the 3 tables involved in the query.

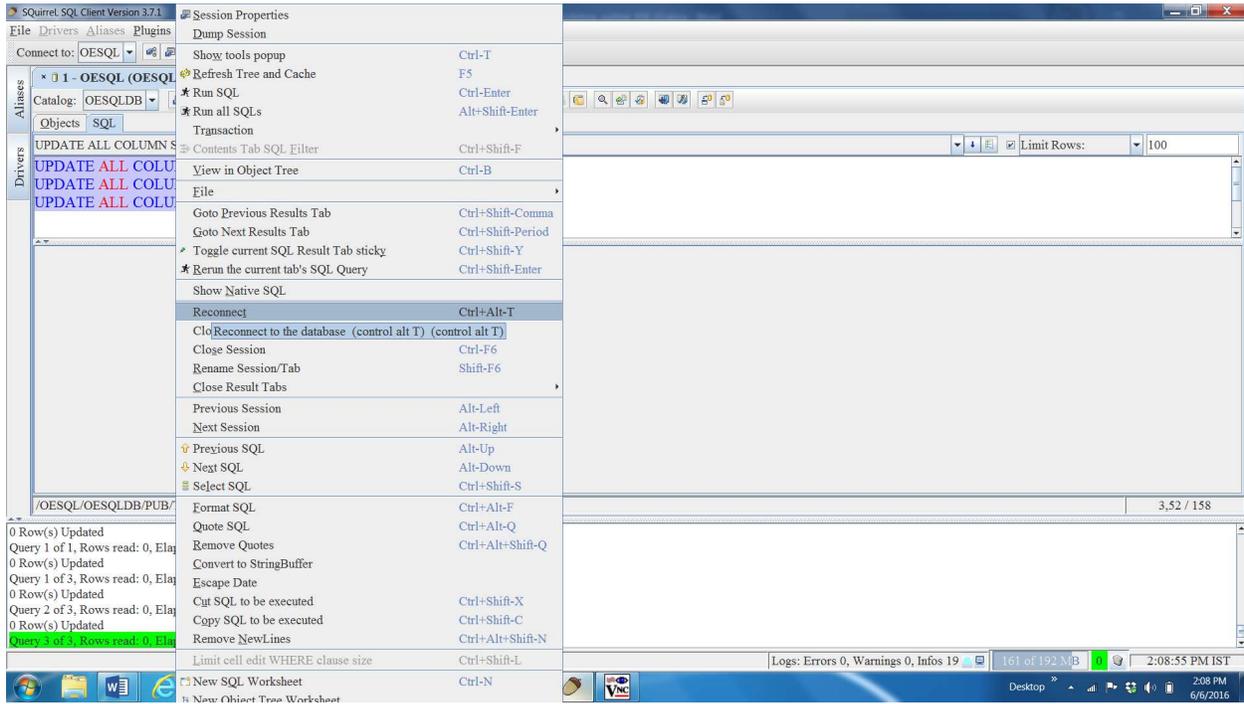
UPDATE ALL COLUMN STATISTICS FOR Pub.Order_Medium;

UPDATE ALL COLUMN STATISTICS FOR Pub.OrderLine_Medium;

UPDATE ALL COLUMN STATISTICS FOR Pub.POLine_Medium;

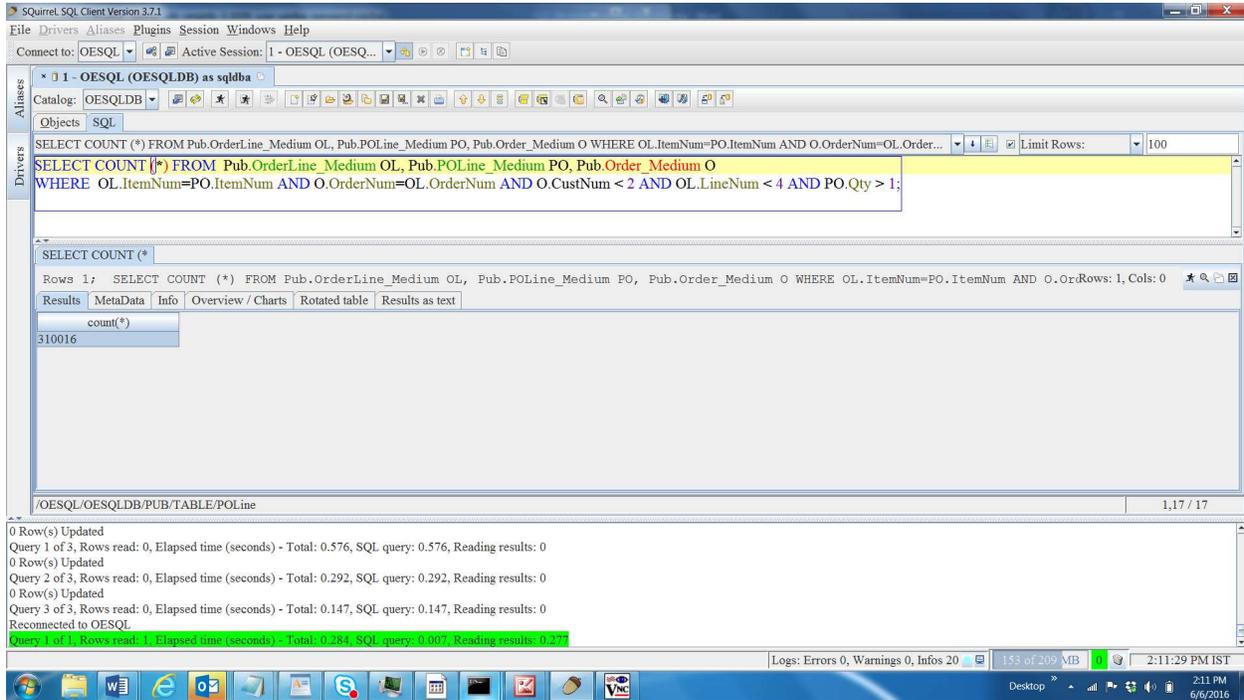


You need to Reconnect to see the effect of Column Statistics.



Now, run the same query again.

```
SELECT COUNT (*) FROM Pub.OrderLine_Medium OL, Pub.POLine_Medium PO, Pub.Order_Medium O
WHERE OL.ItemNum=PO.ItemNum AND O.OrderNum=OL.OrderNum AND O.CustNum < 2 AND OL.LineNum <
4 AND PO.Qty > 1;
```



Now, check the query plan.

```

    _Description
    -----
    SELECT COMMAND.
    PROJECT [53] (
    | PROJECT [52] (
    | | JOIN [66][AUG_NESTED_LOOP-JOIN](
    | | | JOIN [65][AUG_NESTED_LOOP-JOIN](
    | | | | RESTRICT [55] (
    | | | | | PROJECT [42] (
    | | | | | | PUB.OL. [1](
    | | | | | | | TABLE SCAN
    | | | | | | )
    | | | | | , PUB.OL.Itemnum
    | | | | | , PUB.OL.Ordernum
    | | | | | , PUB.OL.Lineum
    | | | | | , PUB.OL.rowid
    | | | | | )
    | | | | )
    | | | | (PEXP3) < (4)
    | | | | Evaluation callback list(
    | | | | | col id# 3
    | | | | )
    | | | )
    )

```



```

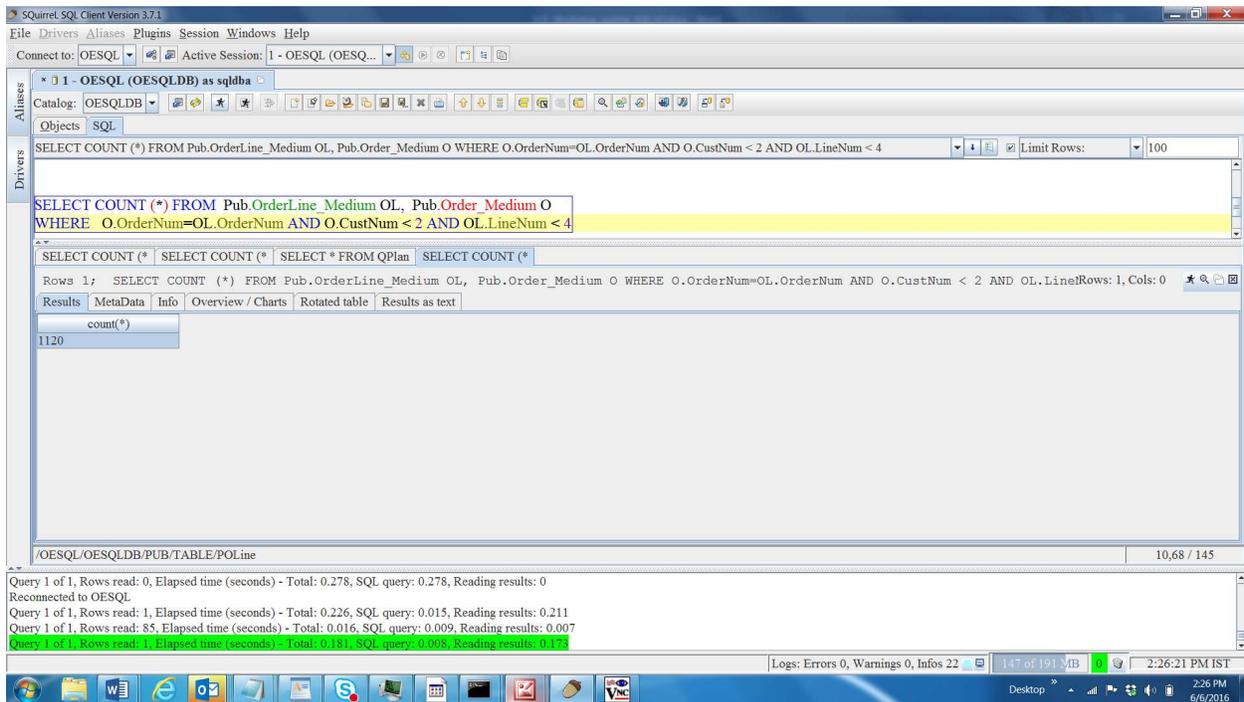
/ / / / )
/ / / / )
/ / /
/ / / )
/ / )
/ , count (*)
/ )
, PEXPR1
)

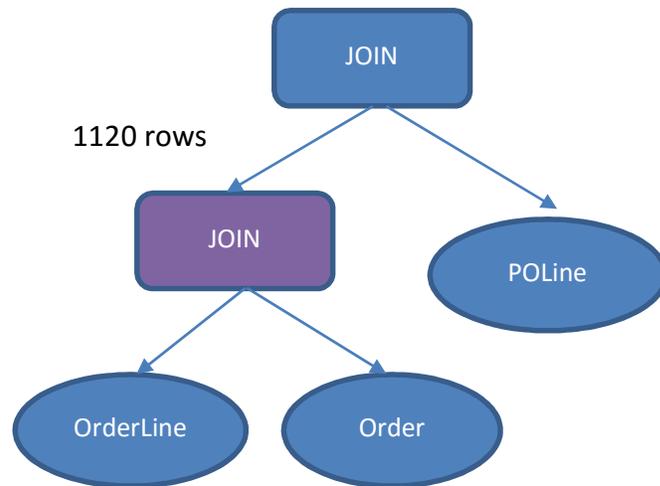
```

Reason for faster execution:

We can notice that, due to column statistics in place intermediate join is now between Pub.Order_Medium and pub.OrderLine_Medium which produces 1120 rows and that is why it is much faster.

SELECT COUNT() FROM Pub.OrderLine_Medium OL, Pub.Order_Medium O
WHERE O.OrderNum=OL.OrderNum AND O.CustNum < 2 AND OL.LineNum<4;*





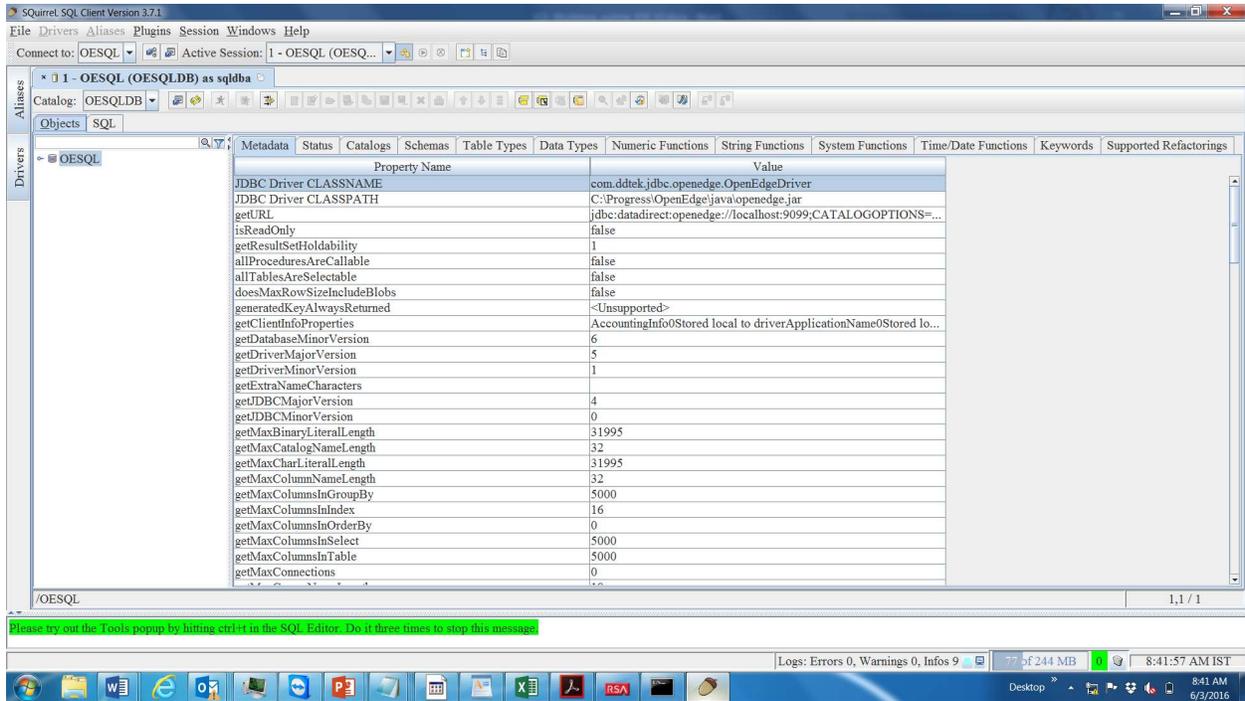
Observation: Having column statistics helped SQL Optimizer to come up with more optimal join order which made the overall query execution much faster. For instance, with the help of column statistics in place, SQL Optimizer could identify that, filter on OrderNum of table Pub.Order can filter more number of rows, because it has more number of unique values and selected this table as part of the intermediate join.

Chapter 9 – What goes into Statistics?

In this chapter you will learn what is inside the statistics and how this information helps SQL Optimizer to come up with optimal query plans.

First let us see about Table Statistics. The System Table involved in this is, Pub.”_SysTblStat”. Let us see what goes into this table when we update the Table statistics. Go to Squirrel window, and check for this table structure.

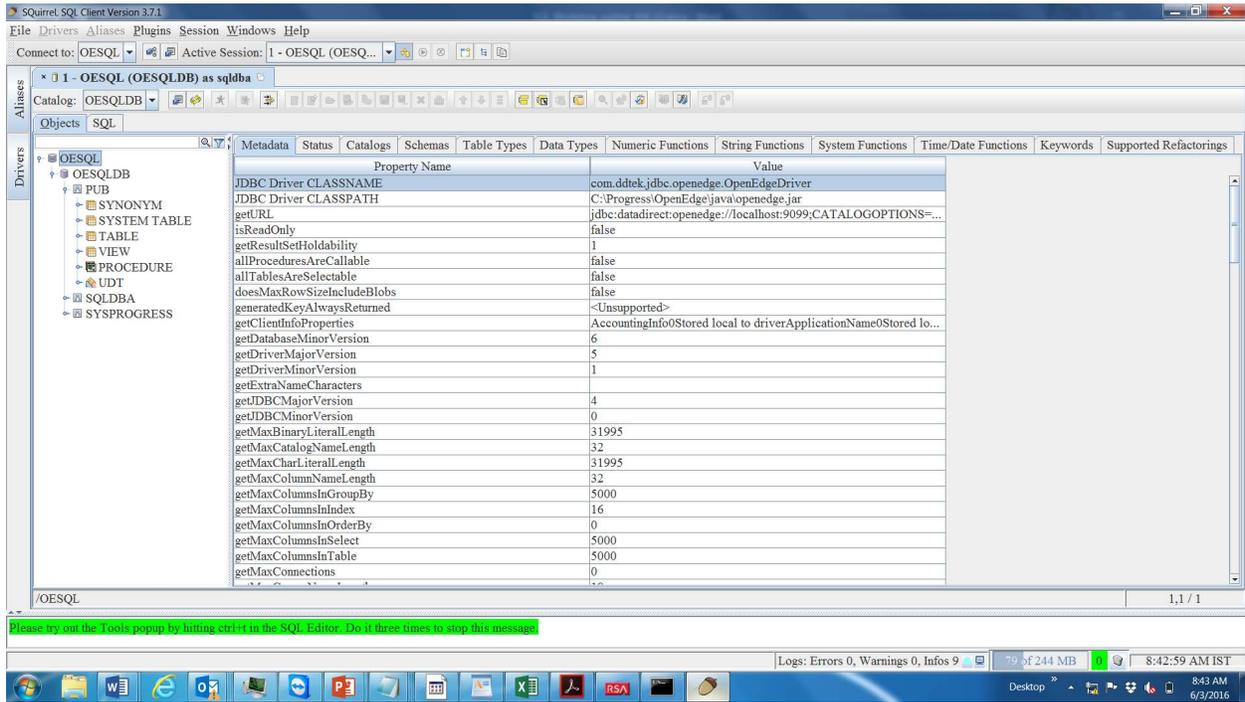
Click on “Objects” Tab (just beside the SQL Tab).



The screenshot shows the Squirrel SQL Client interface. The main window displays the metadata for the OESQL driver. The table below represents the data shown in the screenshot.

Property Name	Value
JDBC Driver CLASSNAME	com.ddtek.jdbc.openedge.OpenEdgeDriver
JDBC Driver CLASSPATH	C:\Progress\OpenEdge\java\openedge.jar
getURL	jdbc:datadirect:openedge://localhost:9099;CATALOGOPTIONS=...
isReadOnly	false
getResultSetHoldability	1
allProceduresAreCallable	false
allTablesAreSelectable	false
doesMaxRowSizeIncludeBlobs	false
generatedKeyAlwaysReturned	<Unsupported>
getClientInfoProperties	AccountingInfoStored local to driverApplicationNameStored lo...
getDatabaseMinorVersion	6
getDriverMajorVersion	5
getDriverMinorVersion	1
getExtraNameCharacters	
getJDBCMajorVersion	4
getJDBCMinorVersion	0
getMaxBinaryLiteralLength	31995
getMaxCatalogNameLength	32
getMaxCharLiteralLength	31995
getMaxColumnNameLength	32
getMaxColumnsInGroupBy	5000
getMaxColumnsInIndex	16
getMaxColumnsInOrderBy	0
getMaxColumnsInSelect	5000
getMaxColumnsInTable	5000
getMaxConnections	0

Expand the alias "OESQL"



Squirrel SQL Client Version 3.7.1

Connect to: OESQL Active Session: 1 - OESQL (OESQ...)

Catalog: OESQLDB

Objects SQL

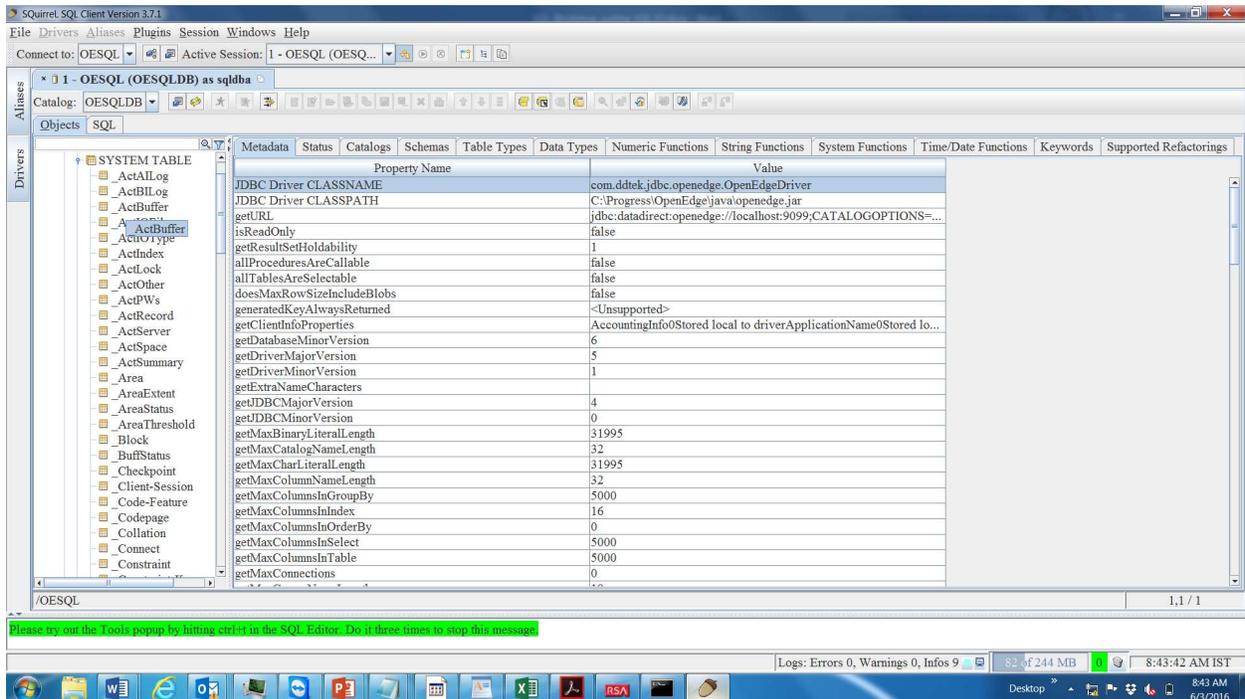
Property Name	Value
JDBC Driver CLASSNAME	com.dteck.jdbc.openedge.OpenEdgeDriver
JDBC Driver CLASSPATH	C:\Progress\OpenEdge\java\openedge.jar
getURL	jdbc:datadirect:openedge://localhost:9099;CATALGOPTIONS=...
isReadOnly	false
getResultSetHoldability	1
allProceduresAreCallable	false
allTablesAreSelectable	false
doesMaxRowSizeIncludeBlobs	false
generatedKeyAlwaysReturned	<Unsupported>
getClientInfoProperties	AccountingInfoStored local to driverApplicationNameStored lo...
getDatabaseMinorVersion	6
getDriverMajorVersion	5
getDriverMinorVersion	1
getExtraNameCharacters	
getJDBCMinorVersion	4
getJDBCMajorVersion	0
getMaxBinaryLiteralLength	31995
getMaxCatalogNameLength	32
getMaxCharLiteralLength	31995
getMaxColumnNameLength	32
getMaxColumnsInGroupBy	5000
getMaxColumnsInIndex	16
getMaxColumnsInOrderBy	0
getMaxColumnsInSelect	5000
getMaxColumnsInTable	5000
getMaxConnections	0

1,1 / 1

Please try out the Tools popup by hitting ctrl+T in the SQL Editor. Do it three times to stop this message.

Logs: Errors 0, Warnings 0, Infos 9 79 of 244 MB 8:42:59 AM IST 6/3/2016

Expand the "SYSTEM TABLE" Item.



Squirrel SQL Client Version 3.7.1

Connect to: OESQL Active Session: 1 - OESQL (OESQ...)

Catalog: OESQLDB

Objects SQL

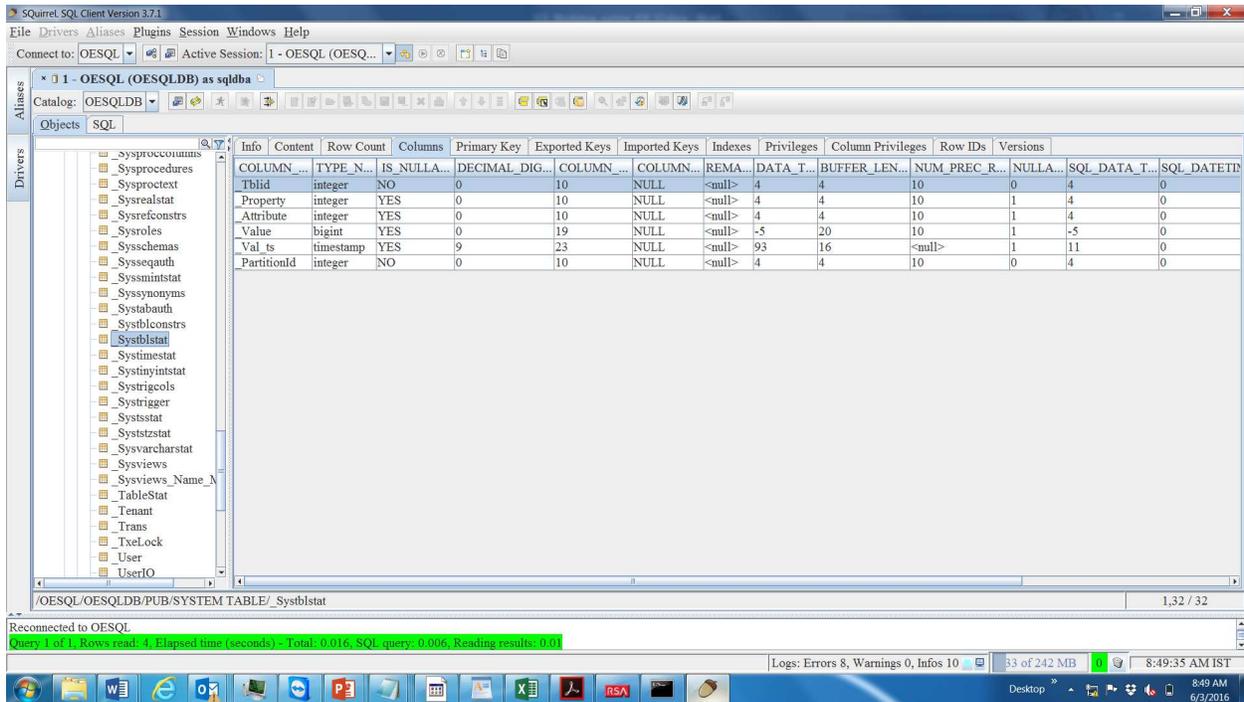
Property Name	Value
JDBC Driver CLASSNAME	com.dteck.jdbc.openedge.OpenEdgeDriver
JDBC Driver CLASSPATH	C:\Progress\OpenEdge\java\openedge.jar
getURL	jdbc:datadirect:openedge://localhost:9099;CATALGOPTIONS=...
isReadOnly	false
getResultSetHoldability	1
allProceduresAreCallable	false
allTablesAreSelectable	false
doesMaxRowSizeIncludeBlobs	false
generatedKeyAlwaysReturned	<Unsupported>
getClientInfoProperties	AccountingInfoStored local to driverApplicationNameStored lo...
getDatabaseMinorVersion	6
getDriverMajorVersion	5
getDriverMinorVersion	1
getExtraNameCharacters	
getJDBCMinorVersion	4
getJDBCMajorVersion	0
getMaxBinaryLiteralLength	31995
getMaxCatalogNameLength	32
getMaxCharLiteralLength	31995
getMaxColumnNameLength	32
getMaxColumnsInGroupBy	5000
getMaxColumnsInIndex	16
getMaxColumnsInOrderBy	0
getMaxColumnsInSelect	5000
getMaxColumnsInTable	5000
getMaxConnections	0

1,1 / 1

Please try out the Tools popup by hitting ctrl+T in the SQL Editor. Do it three times to stop this message.

Logs: Errors 0, Warnings 0, Infos 9 82 of 244 MB 8:43:42 AM IST 6/3/2016

Look for Pub.”_SysTblStat” and click on it. Notice the columns of this table.



Columns we are interested in are “_TblId” & “_Value”. Column “_TblId” denotes the Table Identifier for which Table Statistics are collected and column “_Value” denotes the number of rows the table contains.

Now, go back to "SQL" Tab and run the below query to see table statistics information of table Pub.Order_Medium. The number 19716 denotes the table cardinality and other value denotes the timestamp when table statistics ran last time.

*SELECT * FROM Pub."_SysTblStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Order_Medium');*

The screenshot shows the Squirrel SQL Client interface. The SQL editor contains the query: `SELECT * FROM Pub."_SysTblStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Order_Medium');`. The results pane displays a table with the following data:

Tblid	_Property	_Attribute	Value	_Val_ts	_PartitionId
40	2	<null>	<null>	2016-06-06 12:56:27.444	0
40	4	<null>	19716	<null>	0

The status bar at the bottom indicates the connection is to OESQL, and the query execution log shows: `Query 1 of 1, Rows read: 2, Elapsed time (seconds) - Total: 0.012, SQL query: 0.009, Reading results: 0.006`.

Now, let us see what goes into Index Statistics. The System Table which is used in this is, Pub."_SysIdxStat". Check what goes into this system table when we update Index Statistics.

First, see that the index statistics are not present for table Pub.Customer

*SELECT * FROM Pub."_SysIdxStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*

The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

```
SELECT * FROM Pub."_SysIdxStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer')
```

The query results are shown in a table with the following columns:

_Tblid	_Idxid	_Property	_Attribute	_Value	_Val_ts	_Partitionid

The status bar at the bottom indicates the current location as `/OESQL/OESQLDB/SYSPROGRESS/VIEW` and shows 1,122 / 122 rows. The query log at the bottom displays the following information:

```
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.226, SQL query: 0.015, Reading results: 0.211
Query 1 of 1, Rows read: 85, Elapsed time (seconds) - Total: 0.016, SQL query: 0.009, Reading results: 0.007
Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.181, SQL query: 0.008, Reading results: 0.173
Query 1 of 1, Rows read: 2, Elapsed time (seconds) - Total: 0.012, SQL query: 0.009, Reading results: 0.003
Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.015, SQL query: 0.012, Reading results: 0.008
```

Now, let us run the index statistics for this table, Pub.Order.

UPDATE INDEX STATISTICS FOR Pub.Customer;

The screenshot shows the Squirrel SQL Client interface. The SQL editor contains the command: `UPDATE INDEX STATISTICS FOR Pub.Customer;`. The results pane is empty, indicating that the command was executed successfully but did not return any data rows. The status bar at the bottom shows 'Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.028, SQL query: 0.028, Reading results: 0'.

Now, run the same query again to see what went into Index Statistics System Table, Pub."_SysIdxStat".

*SELECT * FROM Pub."_SysIdxStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*

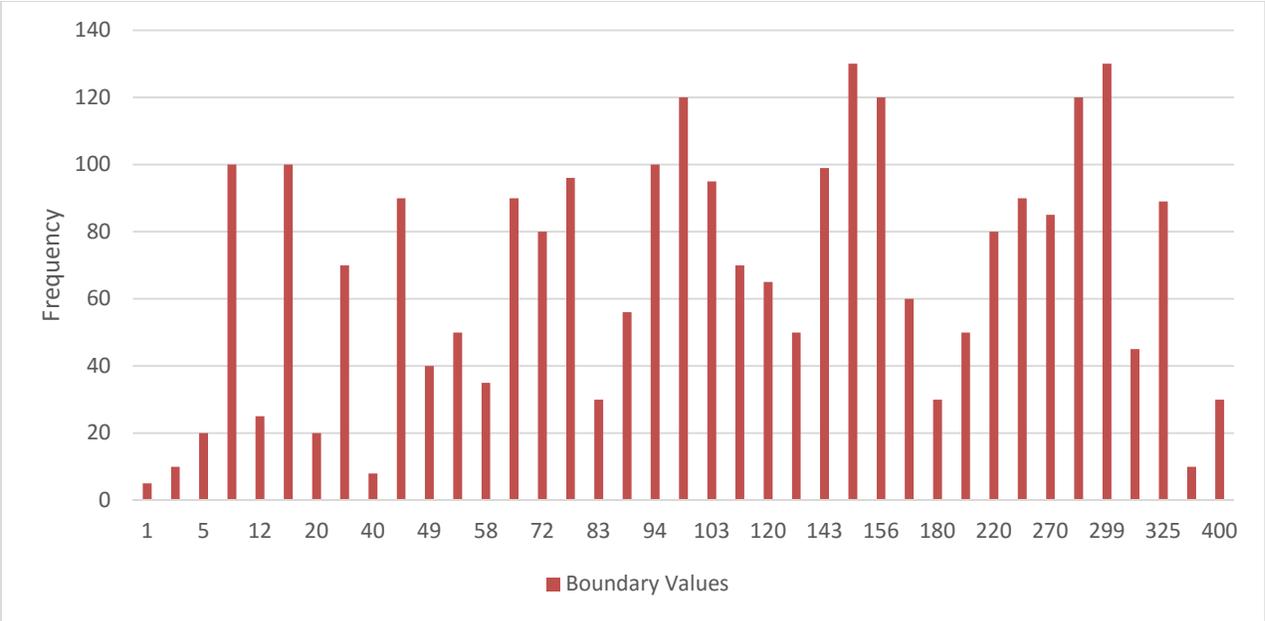
The screenshot shows the Squirrel SQL Client interface with the query: `SELECT * FROM Pub."_SysIdxStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');`. The results pane displays 23 rows of data. The status bar at the bottom shows 'Query 1 of 1, Rows read: 23, Elapsed time (seconds) - Total: 0.008, SQL query: 0.004, Reading results: 0.004'.

_Tblid	_Idxid	_Property	_Attribute	Value	_Val_ts	_Partitionid
2	13	7	0	0	<null>	0
2	13	8	0	0	<null>	0
2	13	11	0	0	<null>	0
2	14	5	0	9	<null>	0
2	14	5	1	796	<null>	0
2	14	6	0	2	<null>	0
2	14	7	0	4	<null>	0
2	14	8	0	9133	<null>	0
2	14	11	0	1061	<null>	0
2	14	11	1	7	<null>	0
2	15	5	0	1017	<null>	0
2	15	6	0	2	<null>	0
2	15	7	0	7	<null>	0
2	14	8	0	177925	<null>	0

We can notice that, there are 3 indexes, each containing 5 or more than 5 properties. Property value “5” denotes number of unique values, property “6” denotes the number of levels in index, property “7” denotes the number of blocks this index occupied, property “8” denotes size of the index in bytes and property “11” denotes the max dup (highest number of duplicate values that SQL found).

Now, let us check what goes into Column Statistics. The system table involved in this is, Pub.”_SysColStat” and various system tables specific to data type of column, such as Pub.”_SysIntStat” or Pub.”_SysNumStat”. These System tables basically represents Histogram which stores information about column data. Table, Pub.”_SysColStat” stores the number of unique values in each bucket (there will be total 40 buckets as of OpenEdge 11.6) and systable pub.”_Sys<data_Type>Stat” stores the boundary values of each bucket.

Sample histogram might look like below



Bucket Number	1	2	3	4	5	6	7	...	39	40
Unique count	1	3	5	8	12	15	20	...	380	400
Boundary value	5	10	20	100	25	100	20	...	10	30

First, check that the column statistics are not present for table Pub.Customer.

```
SELECT * FROM Pub."_SysColStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');
```

The screenshot shows the Squirrel SQL Client interface. The main window displays the following SQL query:

```
SELECT * FROM Pub."_SysColStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer')
```

The query is highlighted in yellow. Below the query, the results pane shows a table with the following columns: `_Tblid`, `_Colid`, `Property`, `Attribute`, `Value`, `Val_ts`, and `Partitionid`. The table is currently empty.

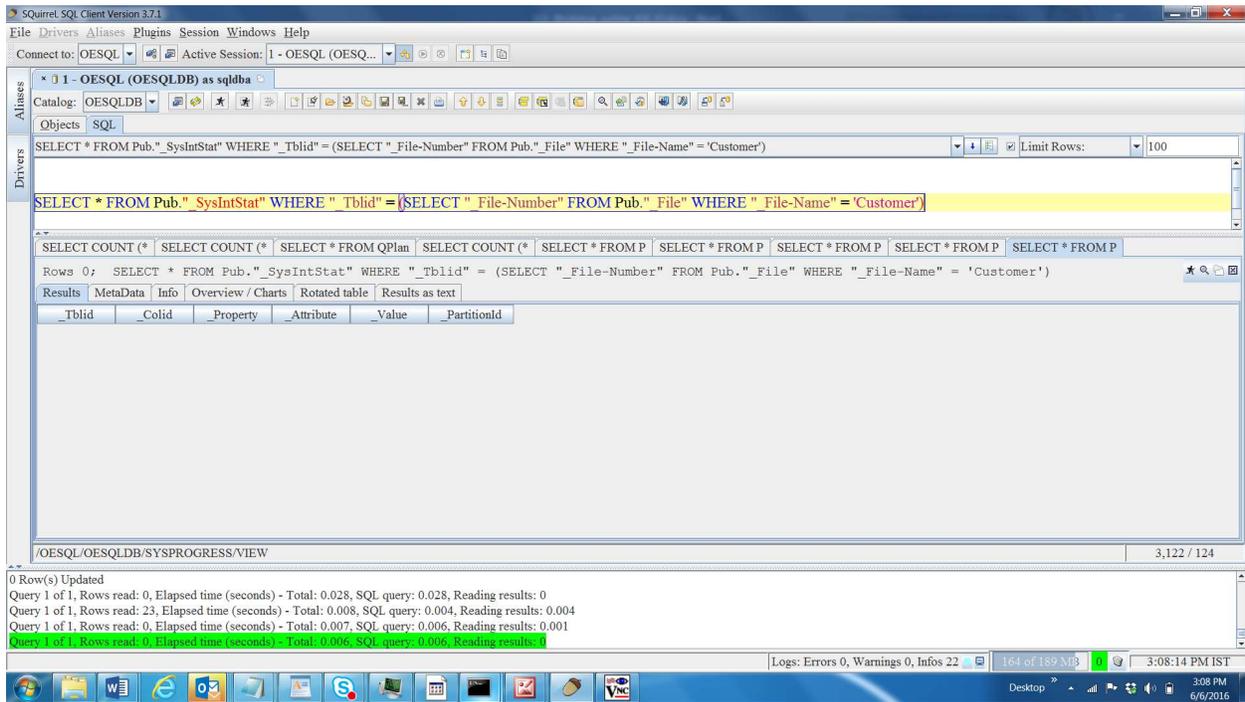
The status bar at the bottom of the window displays the following information:

- Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.015, SQL query: 0.012, Reading results: 0.003
- 0 Row(s) Updated
- Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.028, SQL query: 0.028, Reading results: 0
- Query 1 of 1, Rows read: 23, Elapsed time (seconds) - Total: 0.008, SQL query: 0.004, Reading results: 0.004
- Query 1 of 1, Rows read: 0, Elapsed time (seconds) - Total: 0.007, SQL query: 0.006, Reading results: 0.000

The system tray at the bottom right shows the date and time: 3:06 PM 6/6/2016.

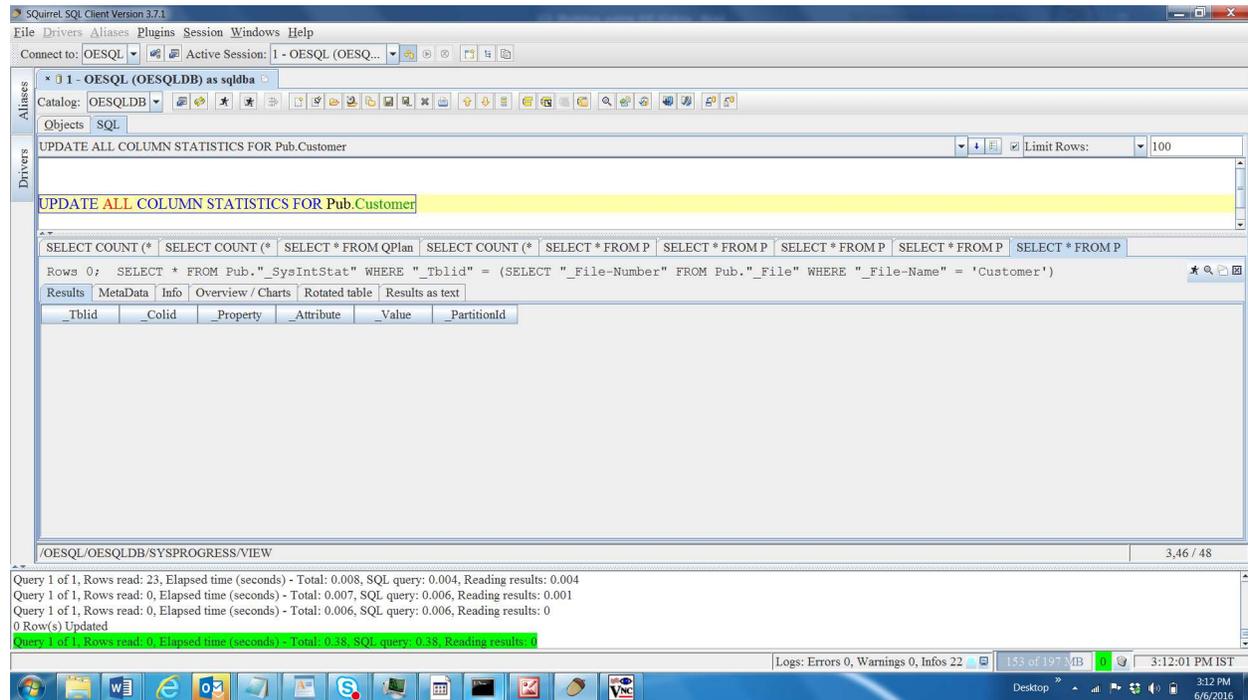
And check the table Pub."_SysIntStat" for integer columns of Pub.Customer table.

*SELECT * FROM Pub."_SysIntStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*



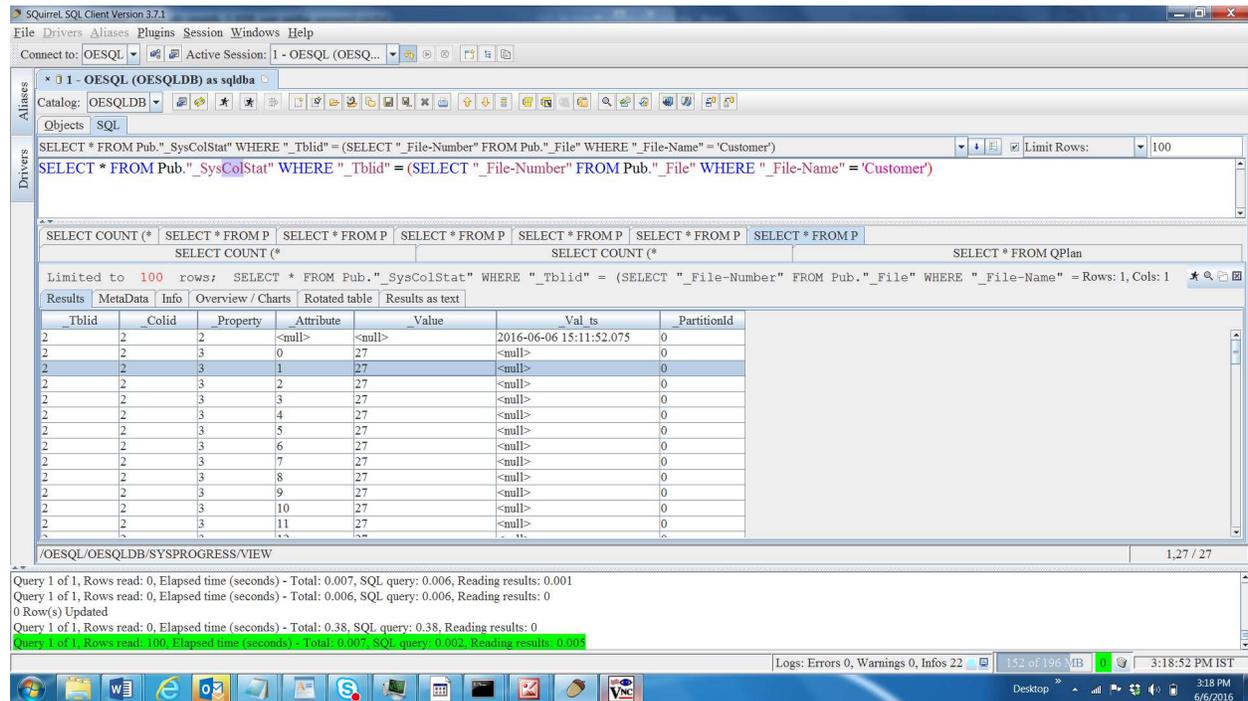
Now, update the column statistics for table Pub.Customer.

UPDATE ALL COLUMN STATISTICS FOR Pub.Customer;



Now, run the same query to check what went into Pub."_SysColStat" and Pub."_SysIntStat" tables.

SELECT * FROM Pub."_SysColStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');



*SELECT * FROM Pub."_SysIntStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*

The screenshot shows the Squirrel SQL Client interface. The query window contains the SQL statement: `SELECT * FROM Pub."_SysIntStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');`. The results pane displays a table with 12 rows and 7 columns: `_Tblid`, `Colid`, `_Property`, `_Attribute`, `_Value`, and `PartitionId`. The values for `_Value` range from 30 to 1354. The status bar at the bottom indicates: "Query 1 of 1, Rows read: 100, Elapsed time (seconds) - Total: 0.007, SQL query: 0.002, Reading results: 0.005".

_Tblid	Colid	_Property	_Attribute	_Value	PartitionId
2	2	1	0	30	0
2	2	1	1	58	0
2	2	1	2	1015	0
2	2	1	3	1102	0
2	2	1	4	1130	0
2	2	1	5	1158	0
2	2	1	6	1186	0
2	2	1	7	1214	0
2	2	1	8	1242	0
2	2	1	9	1270	0
2	2	1	10	1298	0
2	2	1	11	1326	0
2	2	1	12	1354	0

*SELECT * FROM Pub."_SysVarCharStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*

The screenshot shows the Squirrel SQL Client interface. The query window contains the SQL statement: `SELECT * FROM Pub."_SysVarCharStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');`. The results pane displays a table with 12 rows and 7 columns: `_Tblid`, `Colid`, `_Property`, `_Attribute`, `_Value`, and `PartitionId`. The values for `_Value` are categorical, such as "Ais Sports", "Ais Sport n Shop", "Armadillo Sport", etc. The status bar at the bottom indicates: "Query 1 of 1, Rows read: 100, Elapsed time (seconds) - Total: 0.017, SQL query: 0.007, Reading results: 0.009".

_Tblid	Colid	_Property	_Attribute	_Value	PartitionId
2	3	1	0	Ais Sports	0
2	3	1	1	Ais Sport n Shop	0
2	3	1	2	Armadillo Sport	0
2	3	1	3	Atkilak's Hunting & Fishing	0
2	3	1	4	Bedlan's Sporting Goods Inc	0
2	3	1	5	Bill Williams Sporting Goods	0
2	3	1	6	Brendamour's Sporting Goods	0
2	3	1	7	Cahill's Sporting Goods	0
2	3	1	8	Champs Sports	0
2	3	1	9	Columbia Sporting Goods	0
2	3	1	10	Damon's Toys	0
2	3	1	11	Dmk Sports Pro Shop	0
2	3	1	12	Epic Sport Works	0

*SELECT * FROM Pub."_SysnumStat" WHERE "_Tblid" = (SELECT "_File-Number" FROM Pub."_File" WHERE "_File-Name" = 'Customer');*

Query 1 of 1, Rows read: 100, Elapsed time (seconds) - Total: 0.017, SQL query: 0.007, Reading results: 0.01
 Error: [DataDirect][OpenEdge JDBC Driver][OpenEdge] Table/view/synonym "PUB._SysnumStat" cannot be found. (15814)
 SQLState: 42S02
 ErrorCode: -210083
 Query 1 of 1, Rows read: 80, Elapsed time (seconds) - Total: 0.033, SQL query: 0.013, Reading results: 0.01

_Tblid	_Colid	_Property	_Attribute	_Value	_PartitionId
2	13	1	0	2,300	0
2	13	1	1	3,300	0
2	13	1	2	4,400	0
2	13	1	3	5,800	0
2	13	1	4	7,300	0
2	13	1	5	8,400	0
2	13	1	6	9,400	0
2	13	1	7	10,600	0
2	13	1	8	11,500	0
2	13	1	9	12,500	0
2	13	1	10	13,700	0
2	13	1	11	15,000	0
2	13	1	12	16,300	0

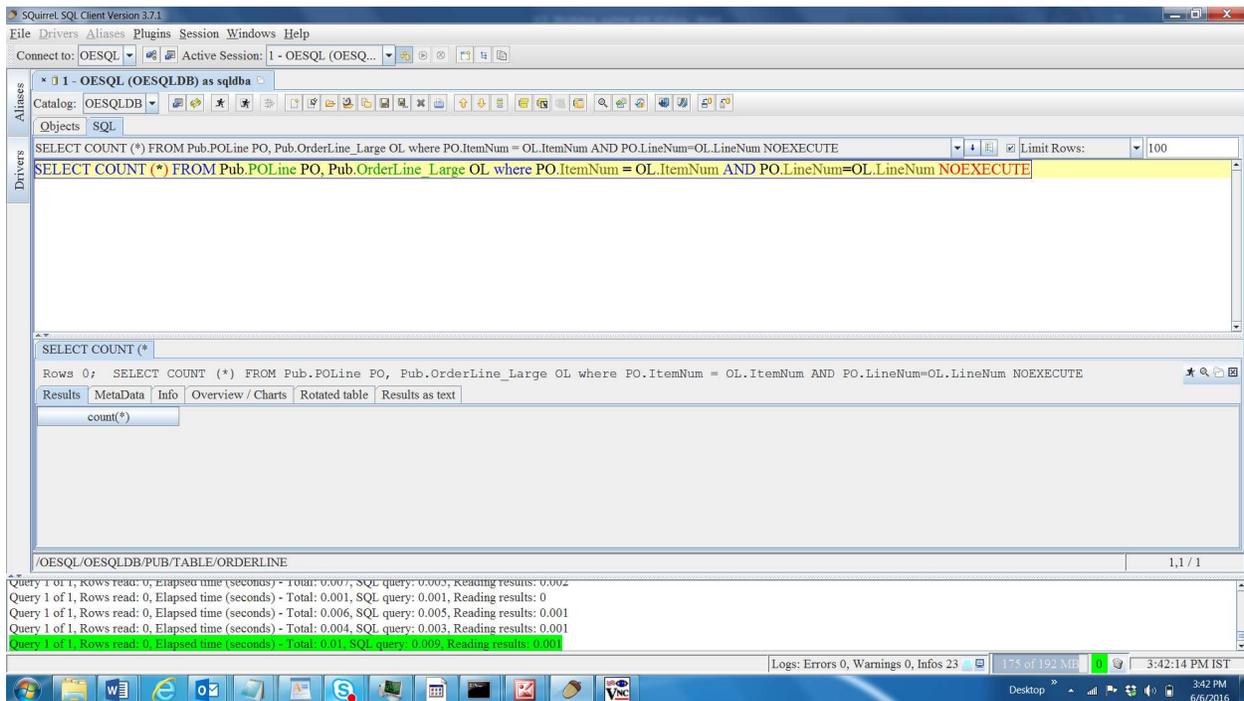
Here, “_Attribute’ represents the bucket number of histogram and “_Value” represents the boundary value of that bucket.

Chapter 10 - SQL hints

In this chapter you will learn different hints which are supported by OE-SQL and their importance in troubleshooting the performance problems.

- a. NoExecute – Useful in cases, where you just want to check the query plan and do not want to execute the actual query. For example, you know some queries are running very slow and you want to know what is the plan used by such queries quickly. Below example will show the usage of “NoExecute”.

SELECT COUNT () FROM Pub.POLine PO, Pub.OrderLine_Large OL where PO.ItemNum = OL.ItemNum AND PO.LineNum=OL.LineNum NOEXECUTE;*



Now, you can check the query plan.

The screenshot shows the Squirrel SQL Client interface. The main window displays the query plan for the query 'SELECT * FROM QPlan'. The query is highlighted in yellow. Below the query, the execution plan is shown in a table format. The table has a 'Description' column and contains the following entries:

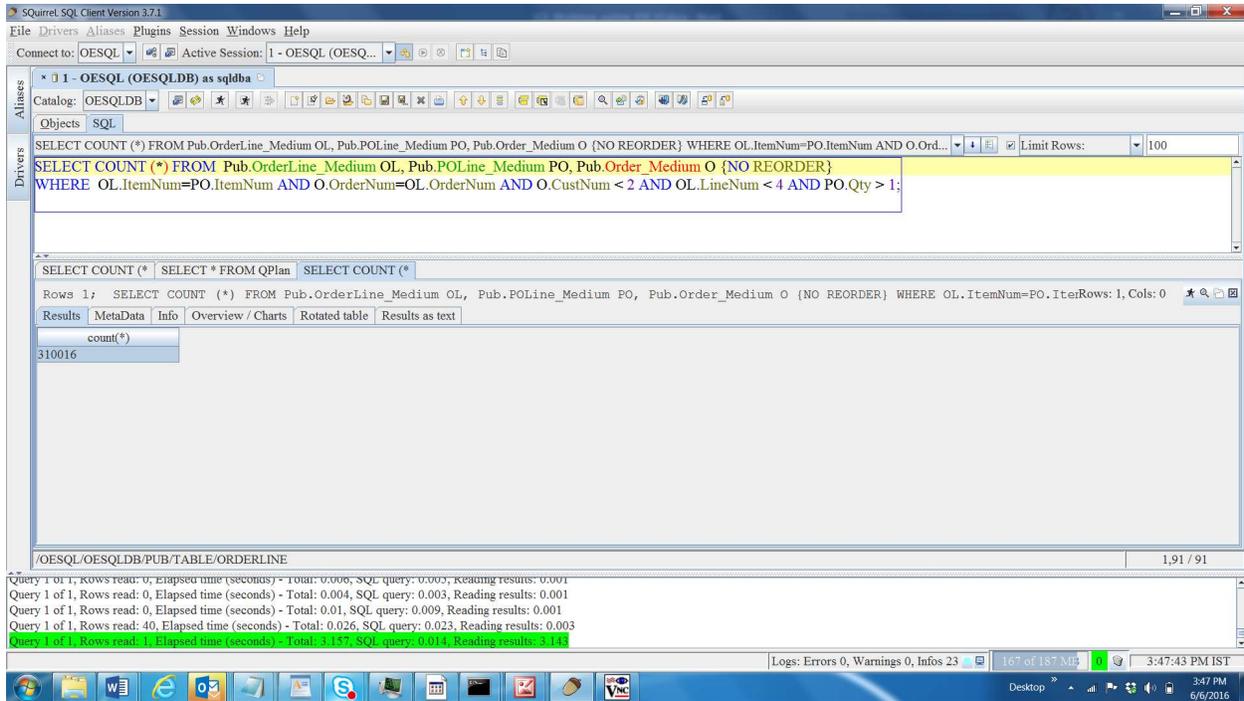
Description
PROJECT [27] (
PUB.OL [2](
TABLE SCAN
)
,PUB.OL.Itemnum
,PUB.OL.Lineum
,PUB.OL.rowid
)
,
(PEXPR2, PEXPR1) = (PEXPR5, PEXPR4)
-- above defines ANL left side keys <relop> right side keys.
,
INDEX SCAN OF DYN. INDEX ON SYSPROGRESS.TM...
[Dynamic Index with key == fld list indexes 1, 0]
[Pexpr map for Dynamic Index data
PEXPR1 PEXPR2 PEXPR3 PEXPR4

The status bar at the bottom of the window shows 'Logs: Errors 0, Warnings 0, Infos 23' and the system clock '3:43:17 PM IST 6/6/2016'.

- b. No reorder – Useful when you suspect that the join order used by SQL optimizer is not efficient and want to validate the query plan/execution time by forcing SQL optimizer to choose specified join order. Below example will show the usage of “No reorder”.

Let us use the same query as the one we used to illustrate the effect of column statistics

```
SELECT COUNT (*) FROM Pub.OrderLine_Medium OL, Pub.POLine_Medium PO, Pub.Order_Medium O {NO REORDER}
WHERE OL.ItemNum=PO.ItemNum AND O.OrderNum=OL.OrderNum AND O.CustNum < 2 AND OL.LineNum < 4 AND PO.Qty > 1;
```



Now, if you check the query plan, it uses the given join order instead of re-ordering to create the best possible plan.

```

    _Description
    -----
    SELECT COMMAND.
    PROJECT [53] (
    | PROJECT [52] (
    | | JOIN [5][AUG_NESTED_LOOP-JOIN](
    | | | JOIN [3][AUG_NESTED_LOOP-JOIN](
    | | | | RESTRICT [55] (
    | | | | | PROJECT [42] (
    | | | | | | PUB.OL. [1](
    | | | | | | | TABLE SCAN
    | | | | | | )
    | | | | | , PUB.OL.Itemnum
    | | | | | , PUB.OL.Ordernum
    | | | | | , PUB.OL.Lineum
    | | | | | , PUB.OL.rowid
  
```

```

| | | | )
| | | |
| | | | | (PEXPR3) < (4)
| | | | | Evaluation callback list(
| | | | | | col id# 3
| | | | | )
| | | | )
| | | | ,
| | | | | (PEXPR1) = (PEXPR5)
| | | | | -- above defines ANL left side ke
| | | | ,
| | | | | INDEX SCAN OF DYN. INDEX ON SYSPRO
| | | | | [Dynamic Index with key == fld
| | | | | [ Pexpr map for Dynamic Index
| | | | | | PEXPR1, PEXPR2, PEXPR3
| | | | | [ Fldids
| | | | | | Fldld 0, Fldld 1, Fldld 2
| | | | | [Dynamic Index] TMPTBL00000026
| | | | | | (SYSPROGRESS.TMPTBL0000002
| | | | | | RESTRICT [54] (
| | | | | | | PROJECT [46] (
| | | | | | | | PUB.PO. [2](
| | | | | | | | | TABLE SCAN
| | | | | | | | )
| | | | | | | , PUB.PO.Itemnum
| | | | | | | , PUB.PO.Qty
| | | | | | | , PUB.PO.rowid
| | | | | | )
| | | | |
| | | | | | (PEXPR2) > (1)
| | | | | | Evaluation callback list(
| | | | | | | col id# 5
| | | | | | )
| | | | | )
| | | |
| | | | )
| | | )
| | ,
| | | (PEXPR2) = (PEXPR8)
| | | -- above defines ANL left side keys <
| | | ,
| | | INDEX SCAN OF DYN. INDEX ON SYSPROGRES
| | | | [Dynamic Index with key == fld lis
| | | | [ Pexpr map for Dynamic Index data
| | | | | PEXPR1, PEXPR2, PEXPR3 ]
| | | | [ Fldids
| | | | | Fldld 0, Fldld 1, Fldld 2 ]
| | | | [Dynamic Index] TMPTBL00000027BL00
| | | | | (SYSPROGRESS.TMPTBL00000025.0)
| | | | RESTRICT [56] (
| | | | | PROJECT [50] (
| | | | | | PUB.O. [4](
| | | | | | | TABLE SCAN
| | | | | | )
| | | | | , PUB.O.Ordernum

```


Now, check the query plan. It selected index, OrderNum_Large.

The screenshot shows the Squirrel SQL Client interface. The main window displays the query plan for the query `SELECT * FROM QPlan`. The query plan is shown in a tree view with the following structure:

```
SELECT COUNT (*)
SELECT COUNT (*)
SELECT COUNT (*)
SELECT * FROM QPlan
SELECT COUNT (*)
SELECT * FROM QPlan
```

The description of the query plan is as follows:

```
SELECT COMMAND.
PROJECT [21] (
  PROJECT [20] (
    RESTRICT [9] (
      PROJECT [18] (
        PUB.ORDERLINE_LARGE [1] (
          INDEX SCAN OF (
            ORDERNUM_LARGE.
            (PUB.ORDERLINE_LARGE.Ordernum) = (10))
          )
        ,PUB.ORDERLINE_LARGE.Lineum
      )
      (PEXPR1) = (3)
      Evaluation callback list(
        col id# 3
      )
    )
  )
)
```

The status bar at the bottom indicates: Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.008, SQL query: 0.005, Reading results: 0.003.

Now, run the same query with force index hints.

SELECT COUNT () FROM Pub.OrderLine_Large FORCE (INDEX (LineNum_Large)) WHERE LineNum=3 AND OrderNum=10;*

The screenshot shows the Squirrel SQL Client interface. The main window displays the query `SELECT COUNT (*) FROM Pub.OrderLine_Large FORCE (INDEX (LineNum_Large)) WHERE LineNum=3 AND OrderNum=10`. The query plan is shown in a tree view with the following structure:

```
SELECT COUNT (*) FROM Pub.OrderLine_Large FORCE (INDEX (LineNum_Large)) WHERE LineNum=3 AND OrderNum=10
```

The description of the query plan is as follows:

```
SELECT COUNT (*) FROM Pub.OrderLine_Large FORCE (INDEX (LineNum_Large)) WHERE LineNum=3 AND OrderNum=10
```

The status bar at the bottom indicates: Query 1 of 1, Rows read: 22, Elapsed time (seconds) - Total: 0.016, SQL query: 0.005, Reading results: 0.011.

Now, check the query plan.

The screenshot shows the Squirrel SQL Client interface. The main window displays the query plan for the command 'SELECT * FROM QPlan'. The query plan is shown in a tree view with the following structure:

```
SELECT COMMAND.  
PROJECT [21] (  
  PROJECT [20] (  
    RESTRICT [9] (  
      PROJECT [18] (  
        PUB.ORDERLINE_LARGE.[1](  
          INDEX SCAN OF (  
            LINENUM_LARGE.  
            (PUB.ORDERLINE_LARGE.Linumum) = (3))  
          )  
        )  
      )  
    )  
  )  
  (PEXPR1) = (10)  
  Evaluation callback list(  
    col id# 2  
  )  
)
```

The status bar at the bottom of the window shows the following information:

- Query 1 of 1, Rows read: 1, Elapsed time (seconds) - Total: 0.611, SQL query: 0.014, Reading results: 0.597
- Query 1 of 1, Rows read: 22, Elapsed time (seconds) - Total: 0.012, SQL query: 0.005, Reading results: 0.007
- Logs: Errors 0, Warnings 0, Infos 9
- 70 of 244 MB
- 4:18:38 PM IST
- Desktop
- 4:18 PM 6/6/2016

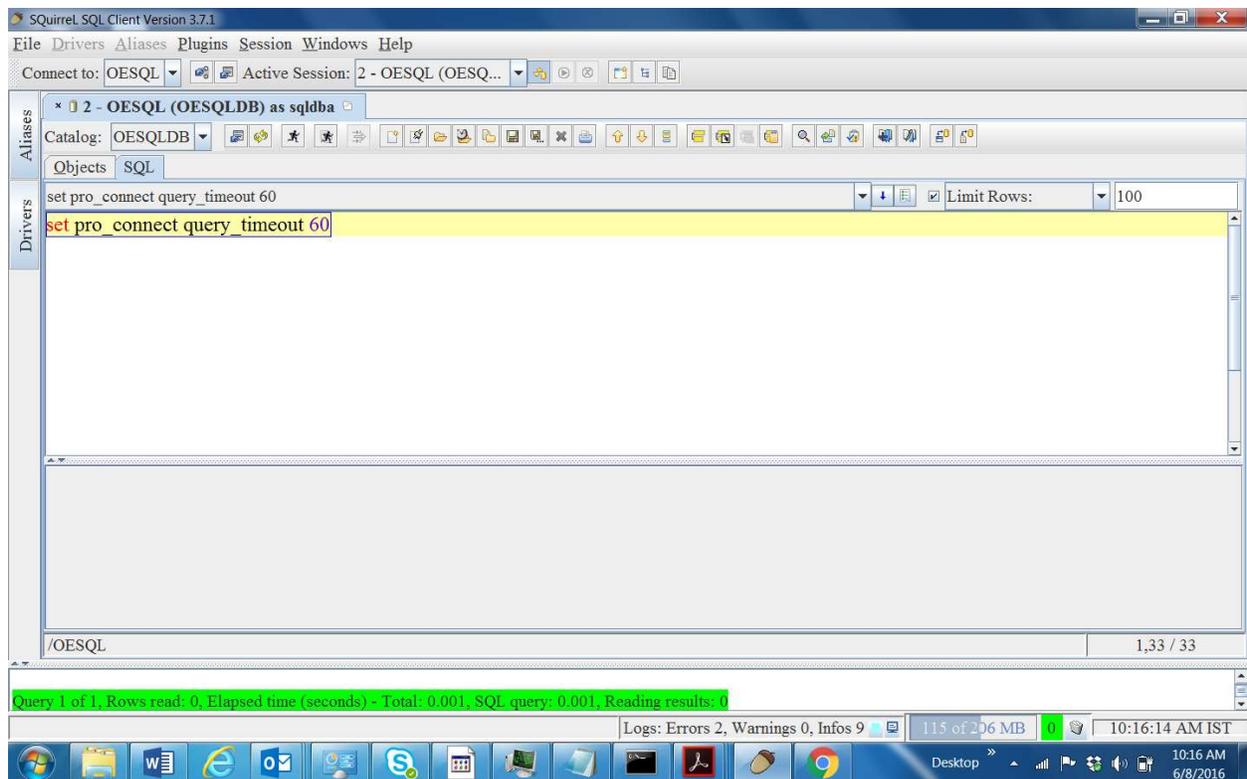
Chapter 11 – Puzzles

In this chapter you will be given two queries having performance problems which performs bad. You are supposed to solve these two performance problems by understanding the reason for bad performance and tuning your database, such that these queries performs better. Note that, obvious way of executing all type of statistics on all the involved tables is not allowed here (This will help us to understand the reason behind the slowness of the query).

Important Note:

Below queries might take more than couple of minutes to complete the execution, so make sure to execute the **QUERY_TIMEOUT** command, so that query aborts after 60 seconds.

```
SET PRO_CONNECT QUERY_TIMEOUT 60
```



Query 1: Below query performs very bad, how do you tune your database to make sure that, it performs better?

```
select top 30 po.price, po.qty, po.itemnum, po.linenum from pub.poline_large po, pub.orderline_large ol  
where po.linenum=ol.linenum and po.itemnum = ol.itemnum order by po.price, po.qty;
```

Query 2: Below query performs very bad, how do you tune your database to make sure that, it performs better?

```
select count (*) from pub.order_10k O, pub.orderline_10k ol, pub.customer_10k c where ol.country=c.country  
and o.custnum=c.custnum and o.ordernum < 2;
```