**original.sql**

**EXPLAIN** **analyze**

**SELECT**

c.CityID,

(**SELECT** CityName **FROM** cities **WHERE** cities.CityID = c.CityID) **AS** CityName,

c.CustomerID **AS** MostActiveCustomer,

**CONCAT**(c.FirstName, ' ', c.LastName) **AS** FullName,

(**SELECT** **SUM**(s2.Quantity)

**FROM** sales s2

**WHERE** s2.CustomerID = c.CustomerID) **AS** TotalUnits,

(**SELECT** **COUNT**(\*)

**FROM** sales s3

**WHERE** s3.CustomerID = c.CustomerID) **AS** TotalSales,

(**SELECT** **AVG**(s4.Discount)

**FROM** sales s4

**WHERE** s4.CustomerID = c.CustomerID) **AS** AverageDiscount

**FROM** customers c

**WHERE** c.CustomerID **IN** (

**SELECT** CustomerID **FROM** (

**SELECT**

s.CustomerID,

**SUM**(s.Quantity) **AS** TotalQ,

c2.CityID,

**ROW\_NUMBER**() **OVER** (

**PARTITION** **BY** c2.CityID

**ORDER** **BY** **SUM**(s.Quantity) **DESC**

) **AS** rn

**FROM** sales s

**JOIN** customers c2 **ON** s.CustomerID = c2.CustomerID

**GROUP** **BY** s.CustomerID, c2.CityID

) **AS** ranked

**WHERE** rn = 1

)

**ORDER** **BY** c.CityID **ASC**, FullName **DESC**;

Explain analyze (took 26 min)

-> Nested loop inner join (cost=6.38e+9 rows=63.8e+9) (actual time=29764..29838 rows=96 loops=1)

-> Sort: c.CityID, FullName DESC (cost=10327 rows=98469) (actual time=351..369 rows=98759 loops=1)

-> Filter: (c.CustomerID is not null) (cost=10327 rows=98469) (actual time=5.22..93.1 rows=98759 loops=1)

-> Table scan on c (cost=10327 rows=98469) (actual time=5.21..82.1 rows=98759 loops=1)

-> Single-row index lookup on <subquery6> using <auto\_distinct\_key> (CustomerID=c.CustomerID) (cost=793982..793982 rows=1) (actual time=0.298..0.298 rows=972e-6 loops=98759)

-> Materialize with deduplication (cost=793982..793982 rows=648146) (actual time=29412..29412 rows=96 loops=1)

-> Filter: (ranked.CustomerID is not null) (cost=1.13..729167 rows=648146) (actual time=29391..29412 rows=96 loops=1)

-> Filter: (ranked.rn = 1) (cost=1.13..729167 rows=648146) (actual time=29391..29412 rows=96 loops=1)

-> Table scan on ranked (cost=2.5..2.5 rows=0) (actual time=29391..29405 rows=98759 loops=1)

-> Materialize (cost=0..0 rows=0) (actual time=29391..29391 rows=98759 loops=1)

-> Window aggregate: row\_number() OVER (PARTITION BY c2.CityID ORDER BY TotalQ desc ) (actual time=29330..29367 rows=98759 loops=1)

-> Sort: c2.CityID, TotalQ DESC (actual time=29330..29339 rows=98759 loops=1)

-> Table scan on <temporary> (actual time=29194..29207 rows=98759 loops=1)

-> Aggregate using temporary table (actual time=29194..29194 rows=98759 loops=1)

-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=12.4..21113 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=687840 rows=6.48e+6) (actual time=12.4..6485 rows=6.69e+6 loops=1)

-> Table scan on s (cost=687840 rows=6.48e+6) (actual time=12.4..5770 rows=6.69e+6 loops=1)

-> Single-row index lookup on c2 using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00192..0.00196 rows=1 loops=6.69e+6)

-> Select #2 (subquery in projection; dependent)

-> Single-row index lookup on cities using PRIMARY (CityID=c.CityID) (cost=1.1 rows=1) (actual time=0.0239..0.024 rows=1 loops=96)

-> Select #3 (subquery in projection; dependent)

-> Aggregate: sum(s2.Quantity) (cost=169323 rows=1) (actual time=5826..5826 rows=1 loops=96)

-> Filter: (s2.CustomerID = c.CustomerID) (cost=104508 rows=648146) (actual time=64.5..5825 rows=88.9 loops=96)

-> Table scan on s2 (cost=104508 rows=6.48e+6) (actual time=7.56..5315 rows=6.69e+6 loops=96)

-> Select #4 (subquery in projection; dependent)

-> Aggregate: count(0) (cost=169323 rows=1) (actual time=5018..5018 rows=1 loops=96)

-> Filter: (s3.CustomerID = c.CustomerID) (cost=104508 rows=648146) (actual time=61.9..5018 rows=88.9 loops=96)

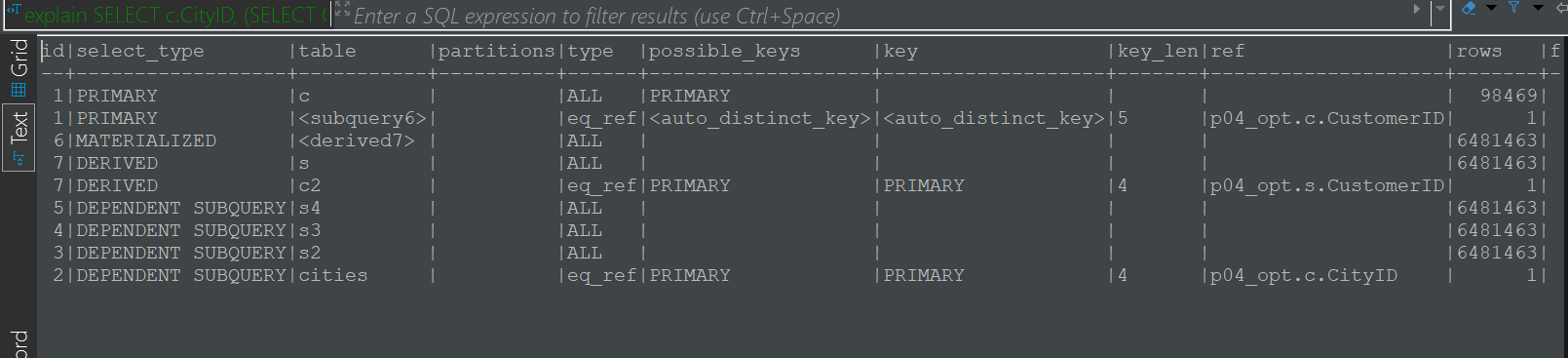
-> Table scan on s3 (cost=104508 rows=6.48e+6) (actual time=10.8..4473 rows=6.69e+6 loops=96)

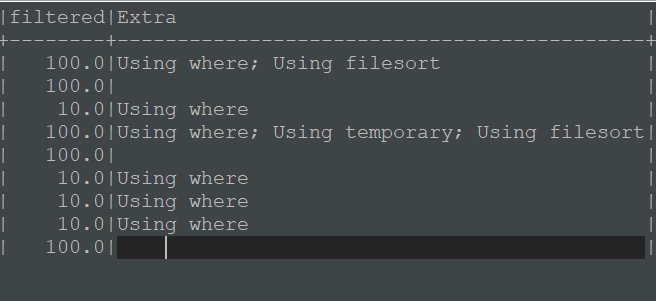
-> Select #5 (subquery in projection; dependent)

-> Aggregate: avg(s4.Discount) (cost=169323 rows=1) (actual time=5254..5254 rows=1 loops=96)

-> Filter: (s4.CustomerID = c.CustomerID) (cost=104508 rows=648146) (actual time=64.2..5253 rows=88.9 loops=96)

-> Table scan on s4 (cost=104508 rows=6.48e+6) (actual time=5.81..4748 rows=6.69e+6 loops=96)





Many dependent subquerys, that’s inefficient, as they are executed repeatedly

Acess methods: 3 – by index (eq\_ref), *6 – full table scan* (ALL). Indexes are not used properly.

Possible keys/keys. We can see that in ¾ cases possible key (primary index) is actually used, which is good, but in the main query(c) it is not used. We do not have any non-clustered indexes.

Rows – full table, over 6 million rows are scanned 5 times.

Filesort/temporary – sorting done on a disc or temporary table, slow

**version1.sql**

**SELECT**

c.CityID,

(**SELECT** CityName **FROM** cities **WHERE** cities.CityID = c.CityID) **AS** CityName,

c.CustomerID **AS** MostActiveCustomer,

**CONCAT**(c.FirstName, ' ', c.LastName) **AS** FullName,

**SUM**(s.Quantity) **AS** TotalUnits,

**COUNT**(SalesID) **AS** TotalSales,

**AVG**(s.Discount) **AS** AverageDiscount

**FROM** customers *c*

**JOIN** sales *s* **ON** *c*.CustomerID = *s*.CustomerID

**WHERE** c.CustomerID **IN** (

**SELECT** CustomerID **FROM** (

**SELECT**

s.CustomerID,

**SUM**(s.Quantity) **AS** TotalQ,

c2.CityID,

**ROW\_NUMBER**() **OVER** (

**PARTITION** **BY** c2.CityID

**ORDER** **BY** **SUM**(s.Quantity) **DESC**

) **AS** rn

**FROM** sales s

**JOIN** customers c2 **ON** s.CustomerID = c2.CustomerID

**GROUP** **BY** s.CustomerID, c2.CityID

) **AS** ranked

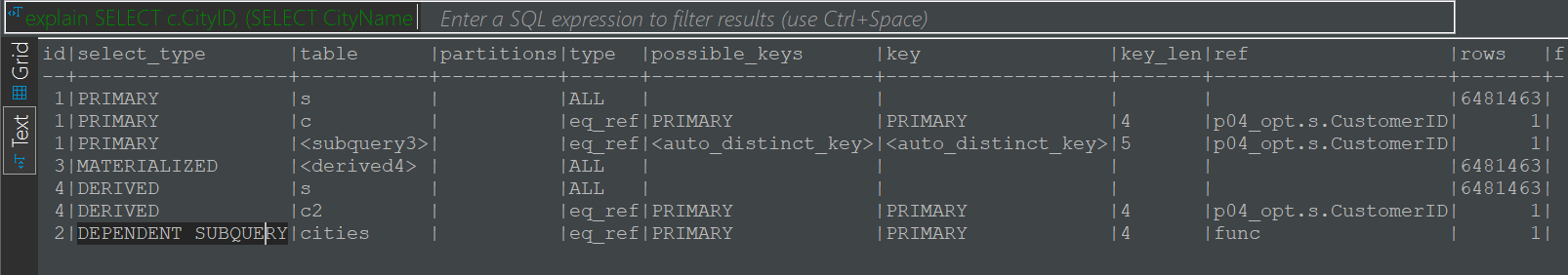
**WHERE** rn = 1

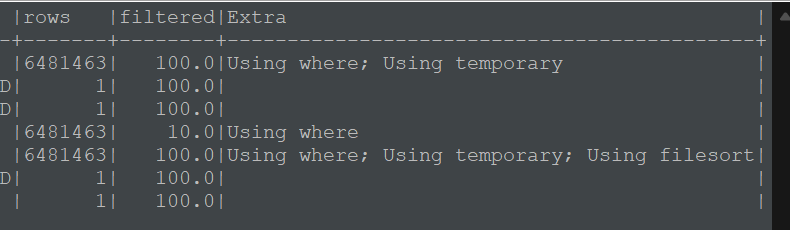
)

**GROUP** **BY** c.CityID, c.CustomerID, c.FirstName, c.LastName

What changed: removed unnecessary subqueries using customers joins sales, removed unneeded order by, added group by city

explain:

****

****

explain analyze:

-> Table scan on <temporary> (actual time=62371..62371 rows=96 loops=1)

-> Aggregate using temporary table (actual time=62371..62371 rows=96 loops=1)

-> Nested loop inner join (cost=420e+9 rows=4.2e+12) (actual time=32518..62257 rows=8532 loops=1)

-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=8.43..24495 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=687840 rows=6.48e+6) (actual time=8.4..7117 rows=6.69e+6 loops=1)

-> Table scan on s (cost=687840 rows=6.48e+6) (actual time=8.39..6376 rows=6.69e+6 loops=1)

-> Single-row index lookup on c using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00232..0.00236 rows=1 loops=6.69e+6)

-> Single-row index lookup on <subquery3> using <auto\_distinct\_key> (CustomerID=s.CustomerID) (cost=793983..793983 rows=1) (actual time=0.00549..0.00549 rows=0.00128 loops=6.69e+6)

-> Materialize with deduplication (cost=793982..793982 rows=648146) (actual time=32505..32505 rows=96 loops=1)

-> Filter: (ranked.CustomerID is not null) (cost=1.13..729167 rows=648146) (actual time=32492..32505 rows=96 loops=1)

-> Filter: (ranked.rn = 1) (cost=1.13..729167 rows=648146) (actual time=32492..32505 rows=96 loops=1)

-> Table scan on ranked (cost=2.5..2.5 rows=0) (actual time=32492..32501 rows=98759 loops=1)

-> Materialize (cost=0..0 rows=0) (actual time=32492..32492 rows=98759 loops=1)

-> Window aggregate: row\_number() OVER (PARTITION BY c2.CityID ORDER BY TotalQ desc ) (actual time=32427..32466 rows=98759 loops=1)

-> Sort: c2.CityID, TotalQ DESC (actual time=32427..32438 rows=98759 loops=1)

-> Table scan on <temporary> (actual time=32318..32328 rows=98759 loops=1)

-> Aggregate using temporary table (actual time=32318..32318 rows=98759 loops=1)

-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=2.85..23082 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=687840 rows=6.48e+6) (actual time=2.83..7037 rows=6.69e+6 loops=1)

-> Table scan on s (cost=687840 rows=6.48e+6) (actual time=2.83..6151 rows=6.69e+6 loops=1)

-> Single-row index lookup on c2 using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00211..0.00215 rows=1 loops=6.69e+6)

-> Select #2 (subquery in projection; dependent)

-> Single-row index lookup on cities using PRIMARY (CityID=c.CityID) (cost=1.1 rows=1) (actual time=0.00941..0.00946 rows=1 loops=96)

Query was executed in 57 seconds. Now full table scan was done only 3 times. The index use hasn’t changed. We have only one dependent subquery – good.   
The number of output rows is 96, which is the correct number of cities. The new inner join is consuming half of the query running time, it is estimated to scan 4.2e+12 rows. Because there is no index on sales.CustomerID.

**version2.sql**

**WITH** ranked **AS** (**SELECT**

s.CustomerID,

c.CityID,

**ROW\_NUMBER**() **OVER** (

**PARTITION** **BY** c.CityID

**ORDER** **BY** **SUM**(s.Quantity) **DESC**

) **AS** rn

**FROM** sales s

**JOIN** customers c **ON** s.CustomerID = c.CustomerID

**GROUP** **BY** c.CityID, s.CustomerID )

**SELECT**

c.CityID,

cities.CityName,

c.CustomerID **AS** *MostActiveCustomer*,

**CONCAT**(c.FirstName, ' ', c.LastName) **AS** FullName,

**SUM**(s.Quantity) **AS** TotalUnits,

**COUNT**(SalesID) **AS** TotalSales,

**AVG**(s.Discount) **AS** AverageDiscount

**FROM** customers *c*

**JOIN** sales *s* **ON** *c*.CustomerID = *s*.CustomerID

**join** cities **on** *c*.CityID = cities.CityID

**WHERE** c.CustomerID **IN** (

**SELECT** CustomerID **FROM** ranked **AS** r

**WHERE** rn = 1

)

**GROUP** **BY** c.CityID, c.CustomerID, c.FirstName, c.LastName

What changed: added CTE, added join with cities and simplified a subquery, used CTE

explain analyze:

-> Table scan on <temporary> (actual time=101167..101167 rows=96 loops=1)

-> Aggregate using temporary table (actual time=101167..101167 rows=96 loops=1)

-> Nested loop inner join (cost=420e+9 rows=4.2e+12) (actual time=40348..101048 rows=8532 loops=1)

-> Nested loop inner join (cost=14.9e+6 rows=6.48e+6) (actual time=7.94..51756 rows=6.69e+6 loops=1)

-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=7.94..34661 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=687840 rows=6.48e+6) (actual time=7.91..9327 rows=6.69e+6 loops=1)

-> Table scan on s (cost=687840 rows=6.48e+6) (actual time=7.91..8177 rows=6.69e+6 loops=1)

-> Filter: (c.CityID is not null) (cost=0.998 rows=1) (actual time=0.00329..0.00343 rows=1 loops=6.69e+6)

-> Single-row index lookup on c using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00294..0.00299 rows=1 loops=6.69e+6)

-> Single-row index lookup on cities using PRIMARY (CityID=c.CityID) (cost=1 rows=1) (actual time=0.00219..0.00223 rows=1 loops=6.69e+6)

-> Single-row index lookup on <subquery2> using <auto\_distinct\_key> (CustomerID=s.CustomerID) (cost=793983..793983 rows=1) (actual time=0.0071..0.0071 rows=0.00128 loops=6.69e+6)

-> Materialize with deduplication (cost=793982..793982 rows=648146) (actual time=40332..40332 rows=96 loops=1)

-> Filter: (r.CustomerID is not null) (cost=1.13..729167 rows=648146) (actual time=40307..40332 rows=96 loops=1)

-> Filter: (r.rn = 1) (cost=1.13..729167 rows=648146) (actual time=40307..40332 rows=96 loops=1)

-> Table scan on r (cost=2.5..2.5 rows=0) (actual time=40307..40323 rows=98759 loops=1)

-> Materialize CTE ranked (cost=0..0 rows=0) (actual time=40307..40307 rows=98759 loops=1)

-> Window aggregate: row\_number() OVER (PARTITION BY c.CityID ORDER BY `sum(s.Quantity)` desc ) (actual time=40208..40272 rows=98759 loops=1)

-> Sort: c.CityID, `sum(s.Quantity)` DESC (actual time=40208..40224 rows=98759 loops=1)

-> Table scan on <temporary> (actual time=40034..40052 rows=98759 loops=1)

-> Aggregate using temporary table (actual time=40034..40034 rows=98759 loops=1)

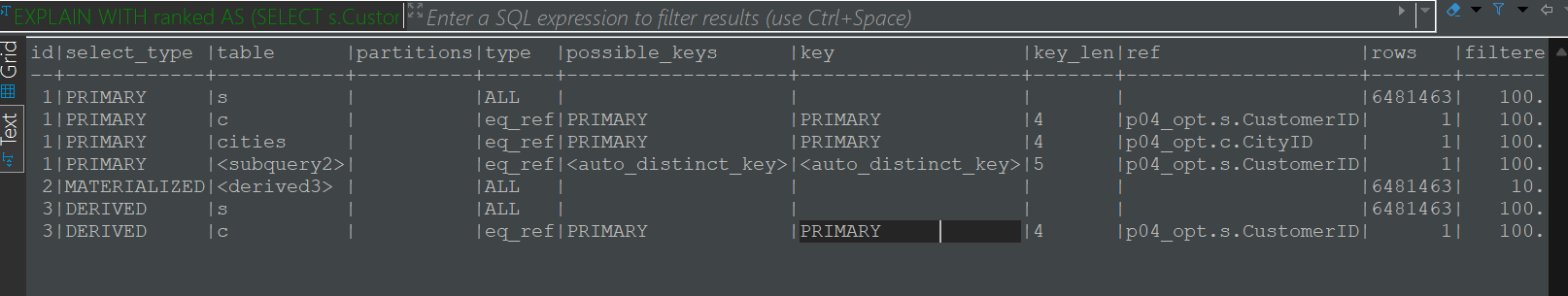
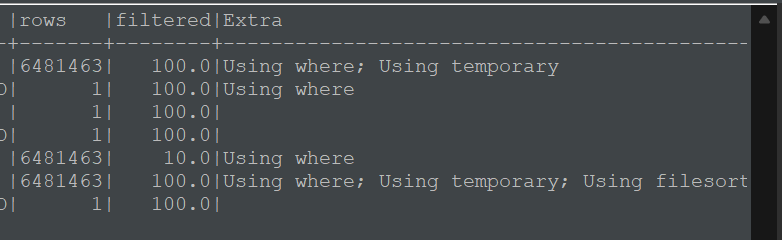
-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=1.77..27147 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=687840 rows=6.48e+6) (actual time=1.76..8599 rows=6.69e+6 loops=1)

-> Table scan on s (cost=687840 rows=6.48e+6) (actual time=1.76..7396 rows=6.69e+6 loops=1)

-> Single-row index lookup on c using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00238..0.00242 rows=1 loops=6.69e+6)

Explain

**** ****

The query took longer time than previous, 104s, the explain is almost the same. But this step will help us improve in the next one.

**Version2.sql**

**WITH** CustomerOrders **AS** (

**SELECT**

*c*.CityID **AS** *City*,

*ci*.CityName **AS** *CityName*,

*c*.CustomerID **AS** *CustomerID*,

**concat**(*c*.FirstName, ' ', *c*.LastName) **AS** *FullName*,

**sum**(*s*.Quantity) **AS** *TotalUnits*,

**count**(SalesID) **AS** *TotalSales*,

**avg**(Discount) **AS** *AverageDiscount*,

**ROW\_NUMBER**() **OVER** (

**PARTITION** **BY** *c*.CityID

**ORDER** **BY** **SUM**(*s*.Quantity) **DESC**

) **AS** *rn*

**FROM** sales *s*

**JOIN** customers *c* **ON** *s*.CustomerID = *c*.CustomerID

**JOIN** cities *ci* **on** *c*.CityID=*ci*.CityID

**GROUP** **BY** *c*.CityID, *ci*.CityName, *c*.CustomerID, *c*.FirstName, *c*.LastName

)

**SELECT**

City, CityName,

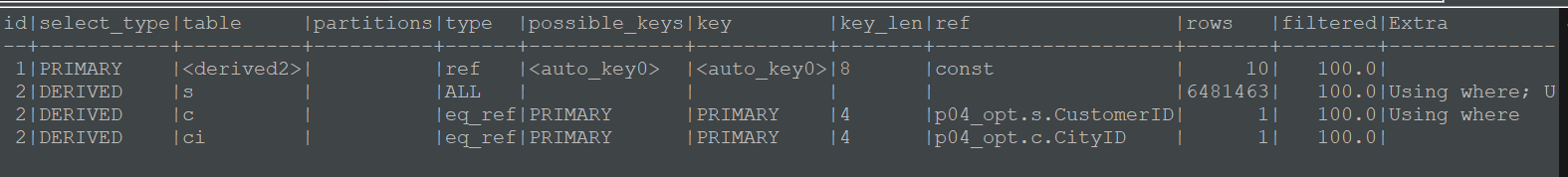
CustomerID **AS** *MostActiveCustomer*, FullName,

TotalUnits, TotalSales, AverageDiscount

**FROM** CustomerOrders

**WHERE** rn = 1;

What changed: basically whole query is now in CTE because I wanted joins not to repeqt. Joins between sales, customers and cities are now done only once.



-> Index lookup on CustomerOrders using <auto\_key0> (rn=1) (cost=0.35..3.5 rows=10) (actual time=217632..217632 rows=96 loops=1)

-> Materialize CTE customerorders (cost=0..0 rows=0) (actual time=217632..217632 rows=98759 loops=1)

-> Window aggregate: row\_number() OVER (PARTITION BY c.CityID ORDER BY TotalUnits desc ) (actual time=217195..217397 rows=98759 loops=1)

-> Sort: c.CityID, TotalUnits DESC (actual time=217195..217242 rows=98759 loops=1)

-> Table scan on <temporary> (actual time=216942..217035 rows=98759 loops=1)

-> Aggregate using temporary table (actual time=216942..216942 rows=98758 loops=1)

-> Nested loop inner join (cost=14.9e+6 rows=6.48e+6) (actual time=11.3..68488 rows=6.69e+6 loops=1)

-> Nested loop inner join (cost=7.81e+6 rows=6.48e+6) (actual time=11.3..46173 rows=6.69e+6 loops=1)

-> Filter: (s.CustomerID is not null) (cost=689179 rows=6.48e+6) (actual time=11.3..10618 rows=6.69e+6 loops=1)

-> Table scan on s (cost=689179 rows=6.48e+6) (actual time=11.3..8757 rows=6.69e+6 loops=1)

-> Filter: (c.CityID is not null) (cost=0.998 rows=1) (actual time=0.00478..0.00492 rows=1 loops=6.69e+6)

-> Single-row index lookup on c using PRIMARY (CustomerID=s.CustomerID) (cost=0.998 rows=1) (actual time=0.00438..0.00443 rows=1 loops=6.69e+6)

-> Single-row index lookup on ci using PRIMARY (CityID=c.CityID) (cost=1 rows=1) (actual time=0.00286..0.00294 rows=1 loops=6.69e+6)

Full scan is done only once, the primary keys are used as indexes. MySQL automatically created auto\_key0 to help primary select access rows faster, but this is probably the reason why query took 217s this time.

**Indexes**

To relieve DBeaver of the need to automatically generate keys and shorten the query execution time, I added the following indexes

**CREATE** **INDEX** customers\_city\_idx **ON** customers(CityID, CustomerID);

**CREATE** **INDEX** sales\_idx **ON** sales(CustomerID, Quantity);

Couldn’t run the query with indexes because MySQL gives an error that I was not able to solve.

