PES UNIVERSITY

ELECTIVE 1: DATABASE TECHNOLOGIES (UE18CS315)

ASSIGNMENT 1

NAME: SHAAZIN SHEIKH SHUKOOR

SRN: PES1201801754

SEMESTER: 5

SECTION: J

I. Review the data model of the project you had submitted in lieu of ISA-2 in 4th semester. Identify gaps in the steps followed and revise the model accordingly.

INTRODUCTION

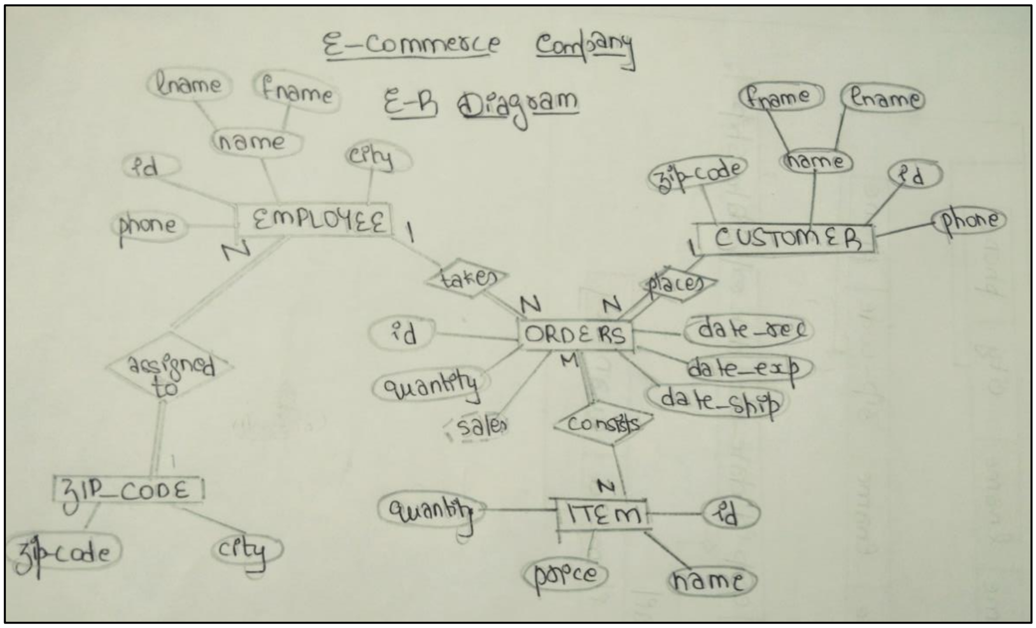
DATABASE: E-COMMERCE COMPANY

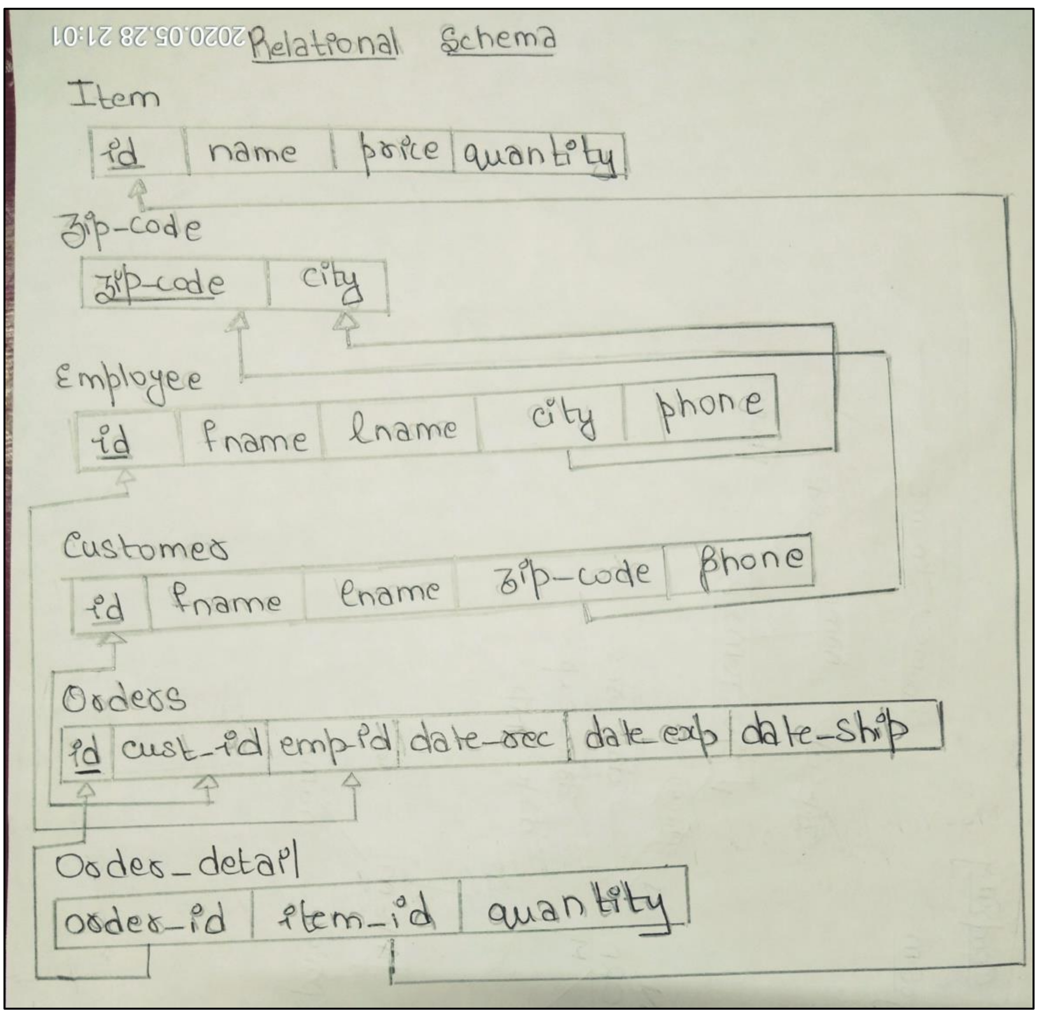
The E-commerce company needs to store information about Item(identified by id, name, price,quantity as attributes), Zip_code(identified by zip_code, city as attributes), Employee(identified by id, fname, lname, city, phone as attributes), Customer(identified by id, fname, lname, zip_code, phone as attributes), Oreders(identified by id,cust_id, emp_id, date_rec, date_exp, date_ship as attributes) and Oder_detail(identified by order_id, item_id, quantity as attributes).

Orders belong to a category, employee *takes* order, customer *places* an order, order *consists of* items, employee *assigned to* zip_code.

An employee can take many orders or may not take any orders at all. An order is assigned to only one employee. A customer can place many or no orders and an order is placed by one customer. One order can consist of many items and one item can be present in many orders. All employees are assigned to a zip_code(city) and all zip_codes have one or more employees assigned to it.

ER DIAGRAM AND RELATIONAL SCHEMA





FUNCTIONAL DEPENDENCIES AND CHOICE OF KEYS

Closure properties were applied to all the attributes to find the candidate keys of all relations. In case of multiple candidate keys, one was chosen as the primary key.
For example in item relation:
$\{id\}^+$= {id, name, price, quantity}
As closure of id is the set of all attributes of the relation, id hence can be considered as a candidate key.

ITEM relation:

Id->name, price, quantity

Primary key:id

ZIP_CODE relation:

Zip_code->city

Primary key: zip_code

EMPLOYEE relation:

Id->fname, lname, city, phone

Primary key:id

Foreign key: city(zip_code)

CUSTOMER relation:

Id-> fname, lname, zip_code, phone

Phone->id, fname, lname, zip_code (assuming phone nos are not repeated)

Primary key:id

Foreign key: zip_code(zip_code)

ORDERS relation:

Id->cust_id, emp_id, date_rec, date_exp, date_ship

Primary key: id

Foreign key: cust_id(customer), emp_id(employee)

ORDER_DETAIL relation:

Order_id, item_id -> quantity

Primary key: order_id and item_id

Foreign key: order_id(orders), item-id(item)

NORMALIZATION AND TESTING FOR LOSSLESS JOIN

All the relations in the database schema obtained from the ER diagram are in 3NF and BCNF as there exists no transitive dependencies in any of the relations. However, 3NF may be violated when upon the addition of an attribute; there exists a dependency from a non-prime to non-prime attribute.

For example, if the attribute ‘city’ is added to Customer relation, there is a new dependency from zip_code - > city which leads to a transitive dependency thereby violating 3NF. Another example is when ‘city’ is added to Orders relation. There is a dependency from emp_id - > city, state which violates 3NF.

A 2nd NF will be violated when there exists a partial dependency from a non-prime attribute to parts of a candidate key. 2nd NF will be violated if we add ‘item_name’ to Order_detail relation, because item_id->item_name but the key is item_id, order_id, which is a partial dependency thereby violating 2nd NF.

A decomposition DECOMP = {R1, R2, . . . , Rm} of R has the lossless join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where ∗ is the NATUAL JOIN of all the relations in DECOMP:

$$*(\pi R1\ (r), \ldots, \pi Rm\ (r)) = r$$

Using Chase’s algorithm:

	id	name	price	quantity	zip_code	city	eid	efname	elname	ephone	cid	cfname	clname	cphone	oid	rec	exp	ship	quantity
item	b11	b12	b13	b14	b15	b16	b17	b18	b19	b110	b111	b112	b113	b114	b115	b116	b117	b118	b119
zip_code	b21	b22	b23	b24	b25	b26	b27	b28	b29	b210	b211	b212	b213	b214	b215	b216	b217	b218	b219
employee	b31	b32	b33	b34	b35	b36	b37	b38	b39	b310	b311	b312	b313	b314	b315	b316	b317	b318	b319
customer	b41	b42	b43	b44	b45	b46	b47	b48	b49	b410	b411	b412	b413	b414	b415	b416	b417	b418	b419
orders	b51	b52	b53	b54	b55	b56	b57	b58	b59	b510	b511	b512	b513	b514	b515	b516	b517	b518	b519
order_detail	b61	b62	b63	b64	b65	b66	b67	b68	b69	b610	b611	b612	b613	b614	b615	b616	b617	b618	b619

	id	name	price	quantity	zip_code	city	eid	efname	elname	ephone	cid	cfname	clname	cphone	oid	rec	exp	ship	quantity
item	a1	a2	a3	a4	b15	b16	b17	b18	b19	b110	b111	b112	b113	b114	b115	b116	b117	b118	b119
zip_code	b21	b22	b23	b24	a5	a6	b27	b28	b29	b210	b211	b212	b213	b214	b215	b216	b217	b218	b219
employee	b31	b32	b33	b34	b35	a6	a7	a8	a9	a10	b311	b312	b313	b314	b315	b316	b317	b318	b319
customer	b41	b42	b43	b44	a14	b46	b47	b48	b49	b410	a11	a12	a13	a14	b415	b416	b417	b418	b419
orders	b51	b52	b53	b54	b55	b56	a7	b58	b59	b510	a11	b512	b513	b514	a15	a16	a17	a18	b519
order_detail	a1	b62	b63	b64	b65	b66	b67	b68	b69	b610	b611	b612	b613	b614	a15	b616	b617	b618	a19

	id	name	price	quantity	zip_code	city	eid	efname	elname	ephone	cid	cfname	clname	cphone	oid	rec	exp	ship	quantity
item	a1	a2	a3	a4	b15	b16	b17	b18	b19	b110	b111	b112	b113	b114	b115	b116	b117	b118	b119
zip_code	b21	b22	b23	b24	a5	a6	b27	b28	b29	b210	b211	b212	b213	b214	b215	b216	b217	b218	b219
employee	b31	b32	b33	b34	b35	a6	a7	a8	a9	a10	b311	b312	b313	b314	b315	b316	b317	b318	b319
customer	b41	b42	b43	b44	a5	a6	b47	b48	b49	b410	a11	a12	a13	a14	b415	b416	b417	b418	b419
orders	b51	b52	b53	b54	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	b519
order_detail	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19

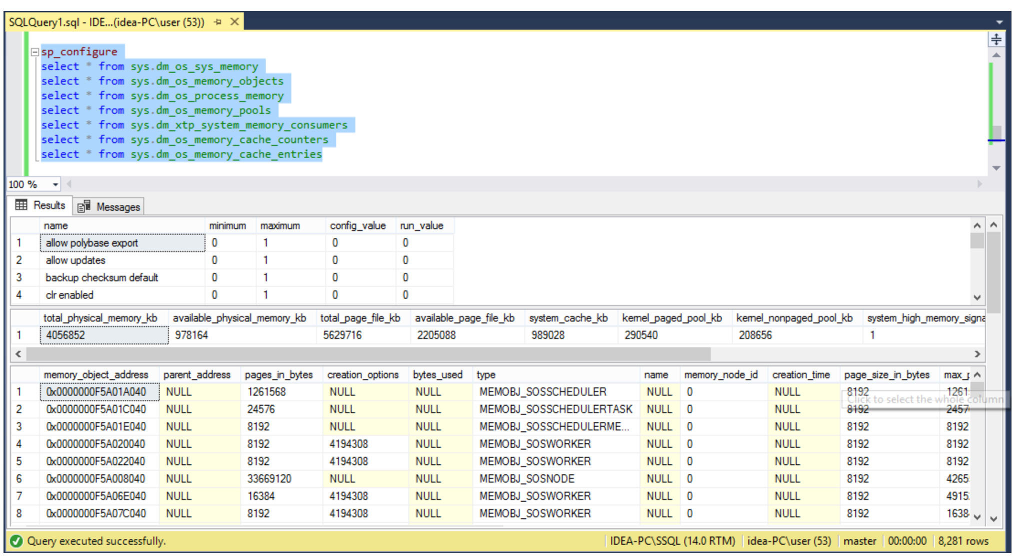
Since all the attributes of the order_detail has ‘a’ values (i.e., the last row), it is verified that the above relation has lossless join property.

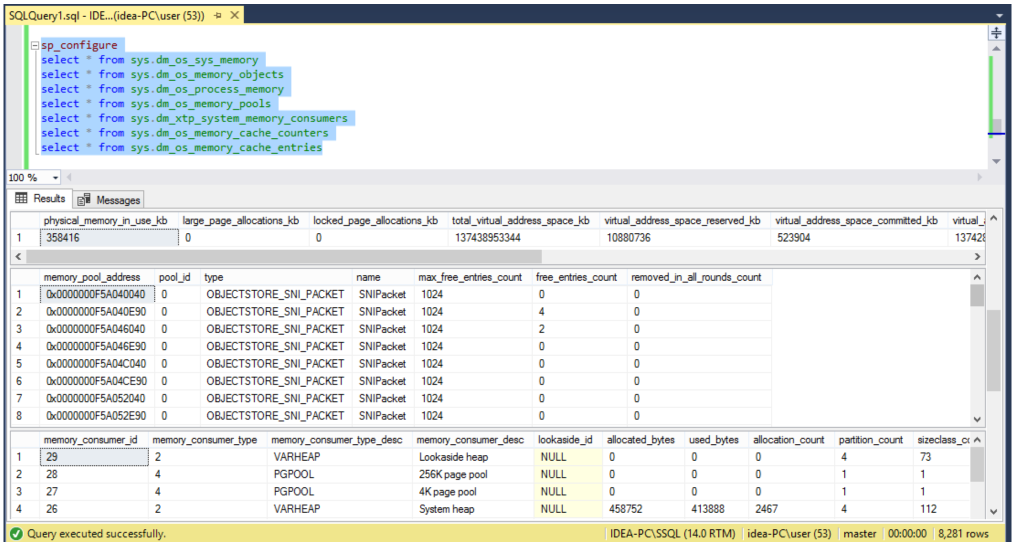
II. Review memory management in any of the contemporary DBMS

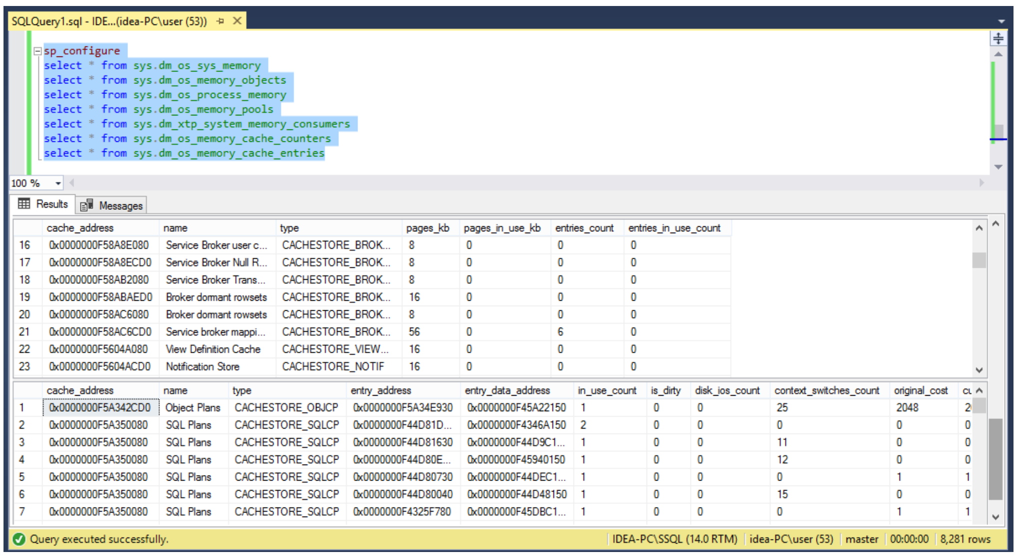
Using the below commands, the memory managed by Microsoft SQL Server can be understood. On executing, we see how different pools for different types of data are created and managed.

```
sp_configure
--based on the server memory, it shows how pools are allocated to store data

select * from sys.dm_os_sys_memory
--shows the total memory and memory allocated by each of the pools

select * from sys.dm_os_memory_objects
--memory object and address in each of the pools
--block size is 8K (page size in bytes)

select * from sys.dm_os_process_memory
--shows os memory, server memory, virtual memory, counters related to memory management
--by os

select * from sys.dm_os_memory_pools
--operating system memory pools, pools created to store os related objects

select * from sys.dm_xtp_system_memory_consumers
--showshow much and what is each consumer using, how much allocated and how much is being
--used

select * from sys.dm_os_memory_cache_counters
--how much page(=block) is allocated by each of the objects

select * from sys.dm_os_memory_cache_entries
--if something is present in cache, how much memory is being used
```

Execution:







These commands help us in check if memory is the bottleneck which is affecting the performance of the server. It is also helpful when we have to look at logs and system catalog tables.

III. Understand the advantages and disadvantages of each RAID level

RAID (redundant array of independent disks) is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy and performance improvement. Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance.

Raid levels	Advantages	Disadvantages
Raid level 0: Stripped disk array without fail tolerance	Complete utilization of storage capacity	A single drive failure will result in complete data loss
Raid level 1: mirroring and duplexing	Simple and easy to implement technology	Usable data storage capacity is only half of the total drive capacity
Raid level 2: hamming code ecc	High data transfer rates	Inefficient as very high ratio of ecc disks to data disks
Raid level 3: parallel transfer with parity	High read and write transfer rate	Difficult and resource intensive to use as a software rate
Raid level 4: independent data disks with shared parity disks	High aggregate read transfer rate	Worst write transaction rate
Raid level 5: independent data disks with distributed parity blocks	Highest read data transaction rate	Difficult to rebuild in an event of a disk failure
Raid level 6: independent data disks with two independent distributed parity schemes	Perfect solution for mission critical applications	More complex controller design
Raid level 10: very high reliability with high performance	Has higher performance	Very limited scalability at a very high inherent cost
Raid level 50: high i/o rates and data transfer performance	High data transfer rates	Very expensive to implement

IV. Performance evaluation when tables are across multiple devices

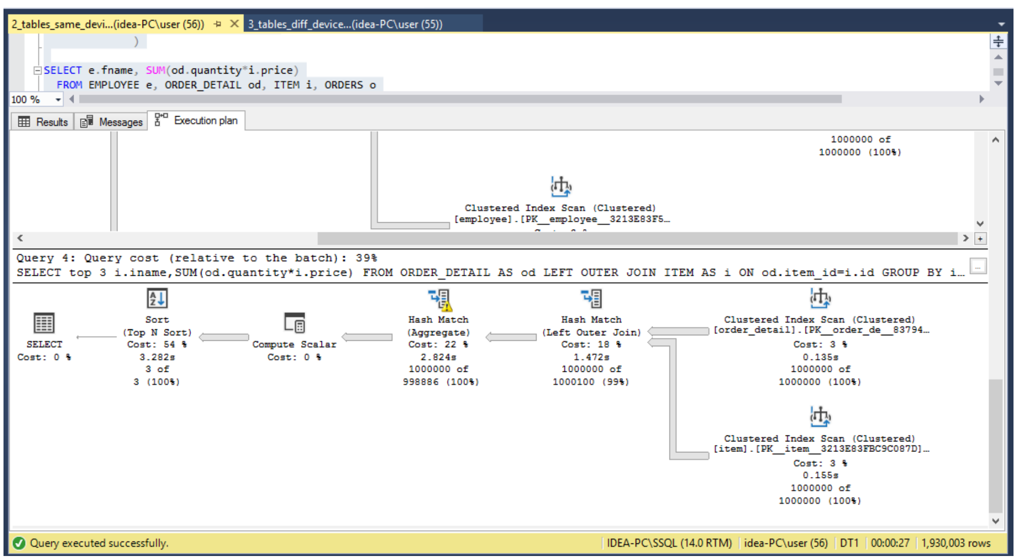
The tables of the database are spread across two devices (D and G drives). This is done so that the input-output gets divided across the disks and the queries are executed faster.

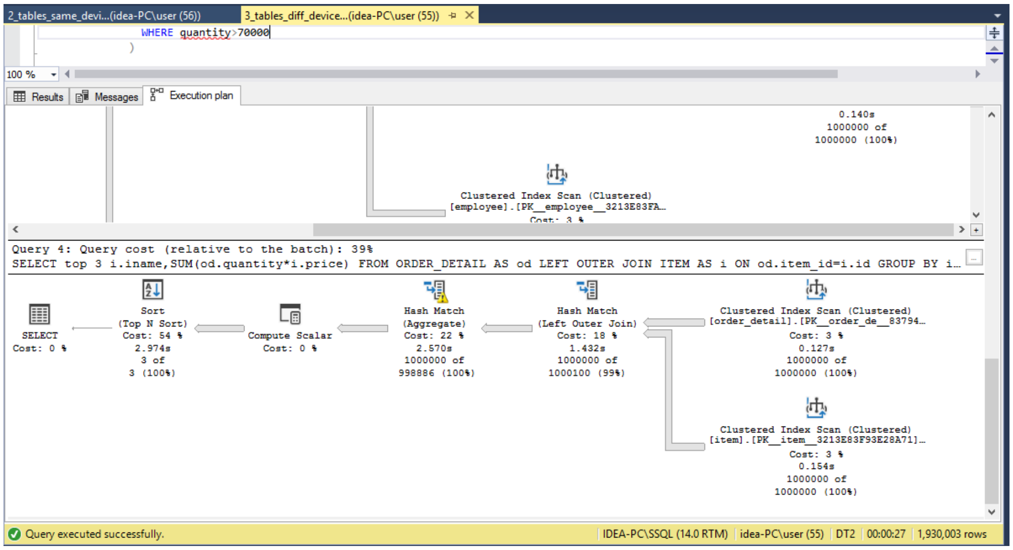
Comparing:

	Same device	Different device
Inserting data into the tables (1M records)	Execution Time: 00:03:18	Execution Time: 00:01:53
Query execution	Execution Time: 00:00:27	Execution Time: 00:00:27
Sort, hash match(aggregate, join)	Time taken was slightly more compared to when tables are on different device.	Time taken was slightly less compared to when tables are on same device.

Inference:

The io time and the execution time is slightly reduced when tables are spread across multiple devices.



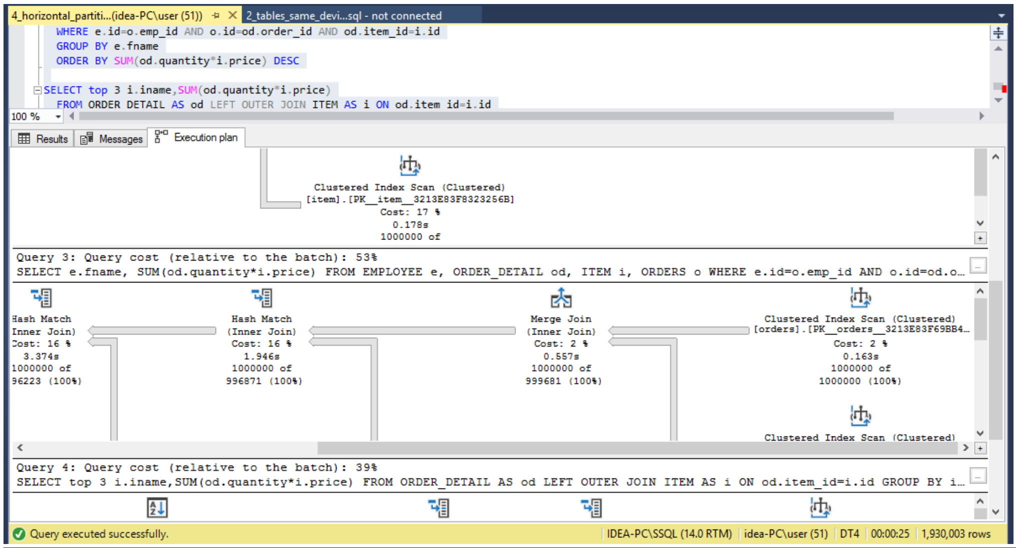


V. Performance evaluation when a table is partitioned across multiple devices

Horizontal partitioning: Horizontal partitioning divides a table into multiple tables that contain the same number of columns, but fewer rows. As order_detail table is assessed multiple times, order_detail is partitioned using partition function and schema.

<u>Comparing:</u>

	Not partitioned	Partitioned
Inserting data into the tables (1M records)	Execution time: 00:03:18	Execution time: 00:02:38
Query execution	Execution Time: 00:00:27	Execution Time: 00:00:25
Queries on order_detail alone	Execution Time: 6550ms	Execution Time: 3814ms
Sort, hash match(aggregate, join)	Time taken was slightly more compared to when table is partitioned.	Time taken was slightly less compared to when table is not partitioned.



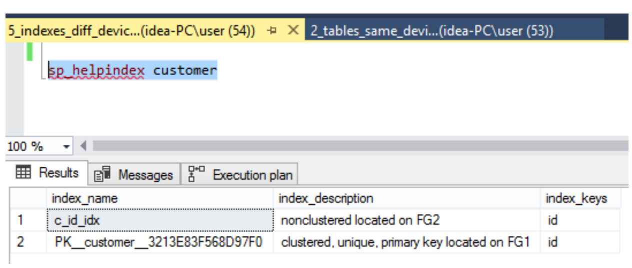
Inference:

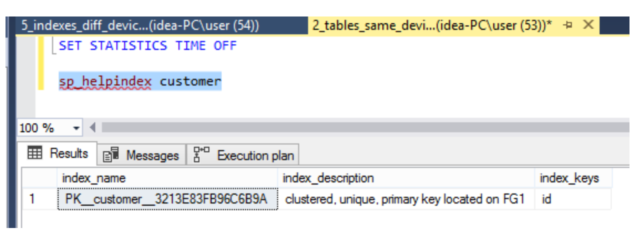
Although there is no much difference in the estimated io cost, but a significant reduction in time is noticed when a table is partitioned.

VI. Performance evaluation on creation of indexes and storing them in separate devices

Clustered indexes were created on all the tables and were stored on a different device.

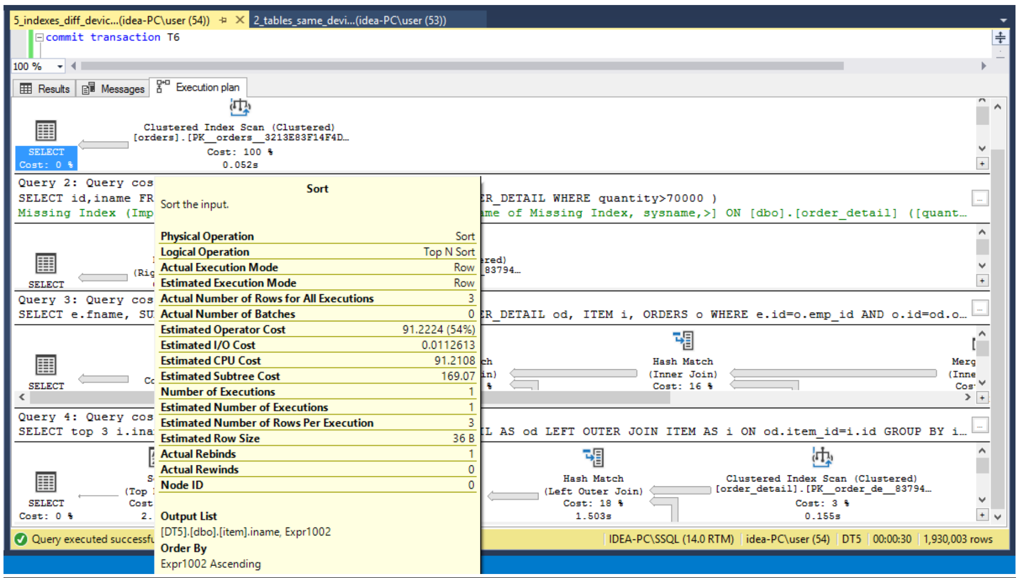
Example:

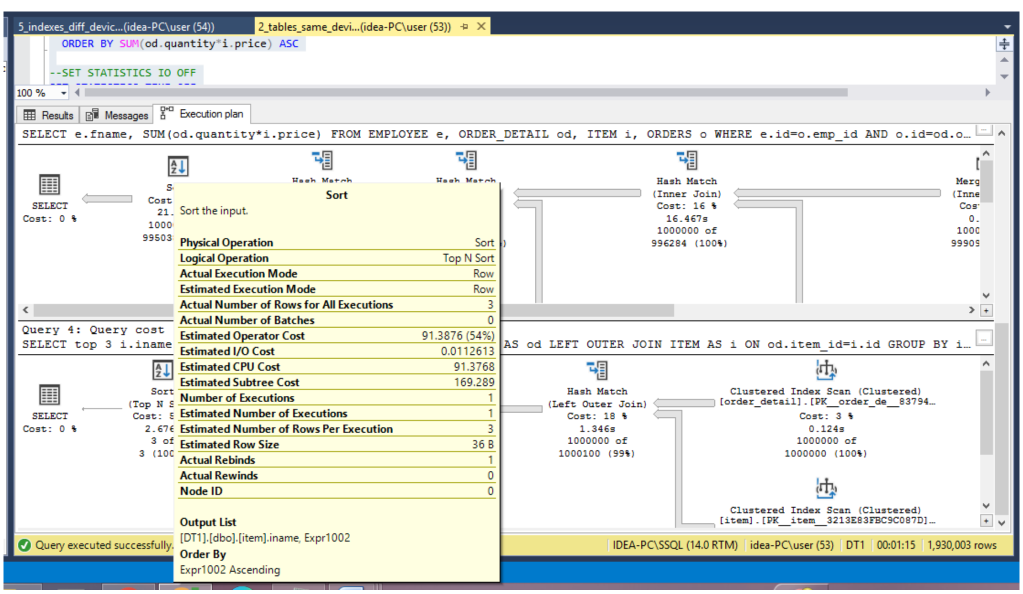




Comparing:

	clustered index(pk)	Non-clustered index (stored in different device)
Inserting data into the tables (1M records)	Execution time: 00:03:18	Execution time: 00:00:09
Query execution	Execution Time: 00:01:15	Execution Time: 00:00:30
Estimated i/o, cpu cost	Cost is higher compared to non-clustered index across multiple devices	Comparatively lower cost and faster execution



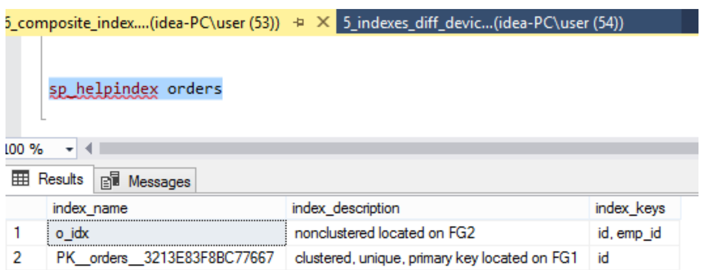


Inference:

Creating non-clustered indexes across multiple devices not only helped in faster execution, but also affected the i/o and cpu cost by reducing it. Spreading over multiple disks reduces i/o (which is a potential bottleneck). It performs better when separated as there is no seek time and just rotational latency.

VII. Performance evaluation on creation of composite indexes and storing them in separate devices

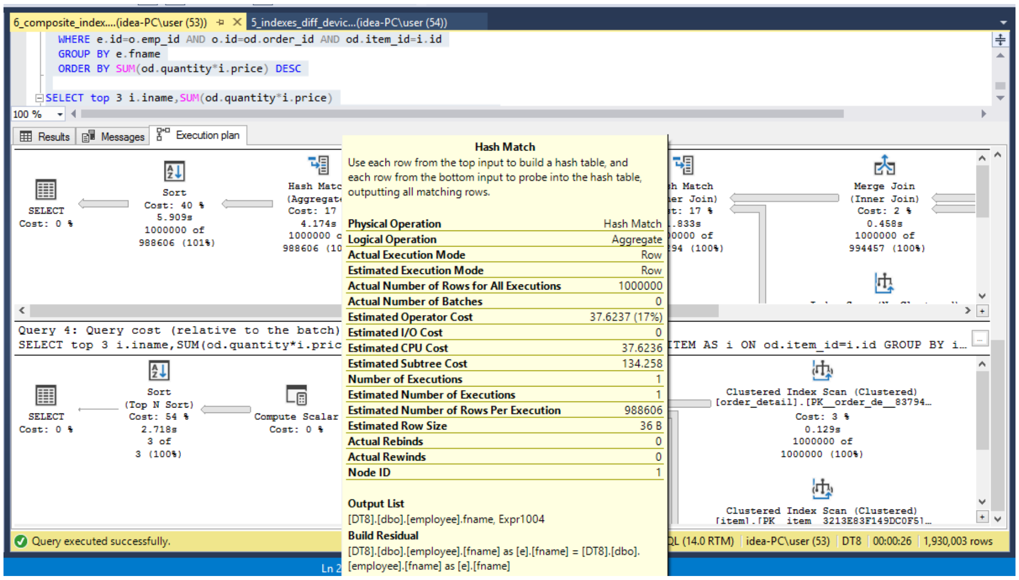
A composite index is an index on two or more columns of the table. Based on the queries, we find those attributes other than the primary key, which are mostly used and create a composite index on that.

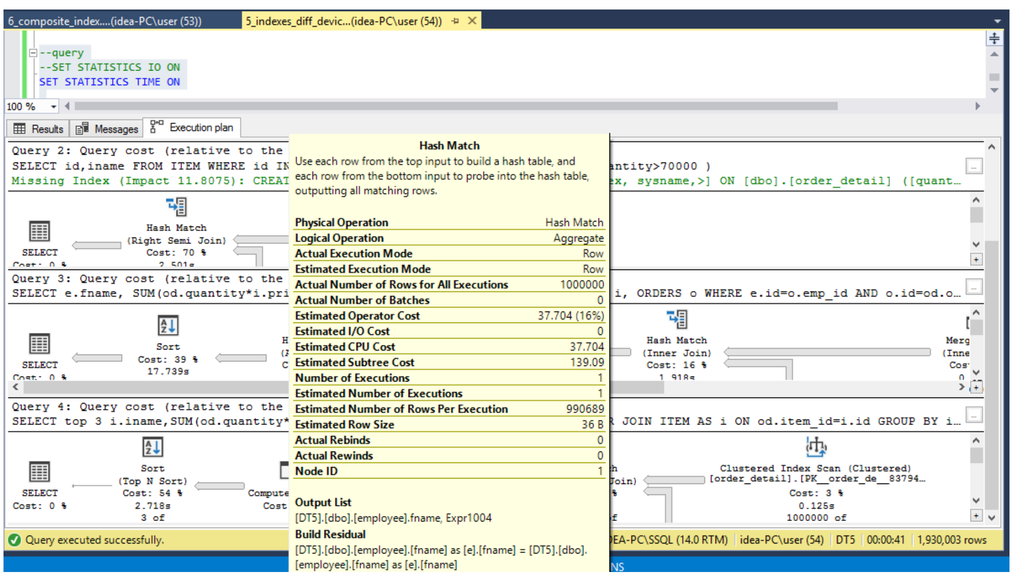


Comparing:

	clustered index(pk)	Composite index (multiple columns)
Inserting data into the tables (1M records)	Execution time: 00:03:18	Execution time: 00:02:14
Query execution	Execution Time: 00:01:15	Execution Time: 00:00:26
Estimated i/o, cpu cost	Cost is higher compared to composite indexes across multiple devices	Comparatively lower cost and faster execution

Execution:





Inference:

Composite indexes has also helped in faster execution and reducing the cost.

Creating indexes thus help improve the performance of queries.