Advanced Database Management System Laboratory

Student Name	Junaid Altaf Girkar
SAP ID	60004190057
Batch	TE Comps A4

Junaid A Girkar

60004190057

TE Comps A-4

EXPERIMENT – 1

1) **What database have you selected?**

ANS 1) The database I have selected is Amazon Kinesis which is a part of the Amazon Web Services (AWS). Amazon Kinesis makes it easy to collect, process, and analyse real-time, streaming data so you can get timely insights and react quickly to new information. Amazon Kinesis offers key capabilities to cost-effectively process streaming data at any scale, along with the flexibility to choose the tools that best suit the requirements of your application. With Amazon Kinesis, you can ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics, and other applications. Amazon Kinesis enables you to process and analyse data as it arrives and respond instantly instead of having to wait until all your data is collected before the processing can begin.

2) **Technical Specifications of the database?**

ANS 2)There are four main components of Kinesis that can be used to accomplish different tasks using their AWS services.

- Kinesis Firehose
- Kinesis Data Analytics
- Kinesis Data Streams
- Kinesis Video Streams

To help ingest real-time data or streaming data at large scales, AWS customers turn to Amazon Kinesis Data Streams. Kinesis Data Streams can continuously capture gigabytes of data per second from hundreds of thousands of sources. The data collected is available in milliseconds, enabling real-time analytics.

To provide this massively scalable throughput, Kinesis Data Streams relies on shards, which are units of throughput and represent a parallelism. One shard provides an ingest throughput of 1 MB/second or 1000 records/second. A shard also has an outbound throughput of 2 MB/sec. As you ingest more data, Kinesis Data Streams can add more shards. Customers often ingest thousands of shards in a single stream.

3) **What application is using this and for what purposes?**

ANS 3) The major application using this database is Netflix.

Netflix is the world's leading internet television network, with more than 200 million members in more than 190 countries enjoying 125 million hours of TV shows and movies each day. Netflix uses AWS for nearly all its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more—hundreds of functions that in total use more than 100,000 server instances on AWS.

Moreover, Netflix, a leading content producer, has used AWS to build a studio in the cloud. This virtual studio enables Netflix to engage top artistic talent, no matter the location, and Netflix artists and partners have the freedom to collaborate without technological or geographical barriers.

Another database that can be used as an alternative is Apache Kafka

Apache Kafka is an open-source "event streaming platform" — a platform that writes and reads event streams. Kafka handles data streams in real-time (like Kinesis.) It's used to read, store, and analyse streaming data and provides organizations with valuable data insights. Uber, for example, uses Kafka for business metrics related to ridesharing trips. The big difference between Kinesis and Kafka lies in the architecture. Kafka "decouples" applications that produce streaming data (called "producers") in the platform's data store *from* applications that consume streaming data (called "consumers") in the platform's data store. Kafka has more of a scattered nature than Kinesis, making it useful for node failures.

According to Apache, over 80 percent of Fortune 100 companies use (and trust) Kafka.

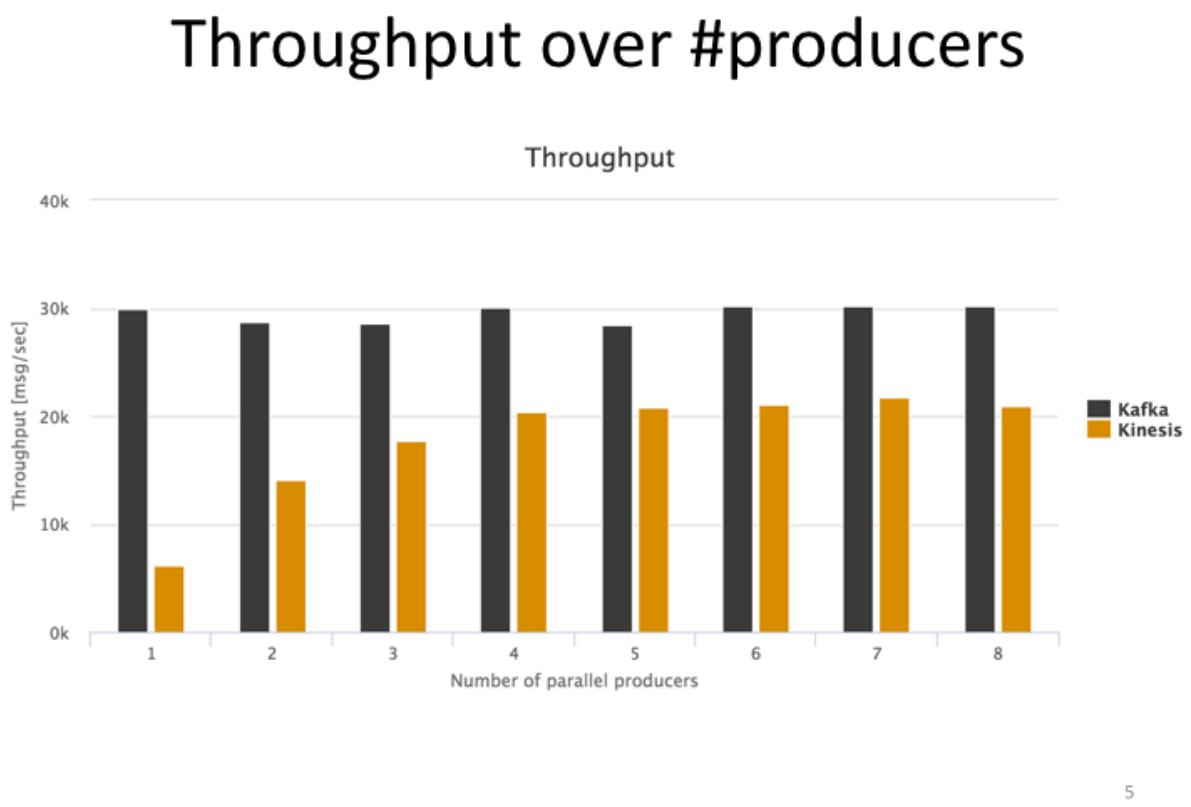
	KINESIS	KAFKA
Where is data stored?	Kinesis Shard	Kafka Partition
Support for SDK?	Java, Android, .NET, Go	Java
Data retention period	7 days	Longer (Users configure retention periods)
Skill level required	Basic	Advanced
Customization	Yes	Yes
Performance limitations	Write synchronously to 3 machines at a time	Fewer limitations
Store	Dynamo db	Zookeeper
Price	Based on resources used, with no upfront costs	Free (open-source) but consider hardware/s costs
Support	Developer center, tutorials, and more	Tutorials, meetups, videos, and more

Kinesis and Kafka have several unique features.

- Kinesis supports Java, Android, .NET, and Go; Kafka only supports Java.
- Kinesis lets users write synchronously to three machines or data centers; Kafka users have more configurations.
- Kinesis stores data in shards; Kafka stores data in partitions

Regarding performance, Kinesis reaches a throughput of thousands of messages every second. Kafka, however, reaches a throughput of around 30,000 messages every second, making it the clear winner.

There are limitations to both platforms. Kinesis has a 7-day data retention period. Kafka takes a lot of effort to set-up and run. (Kafka requires distributed engineering and cluster management experience.)



5

4) What is your recommendation

ANS 4) For Netflix, Amazon Kinesis is doing a good enough job. Apache Kafka would give a slightly better performance but in the case of Netflix, scalability and reliability is more important and for that Amazon is the better choice. Plus using Kafka would require them to hire more people, this would increase their costs and also Kafka requires higher maintenance time which would ensure Netflix losing more money that what they would have had to spend on using Kinesis.

5) **What is the conclusion?**

ANS 5) Both Apache Kafka and AWS Kinesis Data Streams are good choices for real-time data streaming platforms. If you need to keep messages for more than 7 days with no limitation on message size per blob, Apache Kafka should be your choice. However, Apache Kafka requires extra effort to set up, manage, and support. If your organization lacks Apache Kafka experts and/or human support, then choosing a fully-managed AWS Kinesis service will let you focus on the development. AWS Kinesis is catching up in terms of overall performance regarding throughput and events processing.

EXPERIMENT 2

Aim: Simulate query optimization by applying an SQL query on your selected database.

DATABASE SCHEMA:

```
/*==============================================================*/
/* Table: Customer                                              */
/*==============================================================*/
create table Customer (
   Id                int                not null,
   FirstName         varchar(40)        not null,
   LastName          varchar(40)        not null,
   City              varchar(40)        null,
   Country           varchar(40)        null,
   Phone             varchar(20)        null,
   constraint PK_CUSTOMER primary key (Id)
)
;

/*==============================================================*/
/* Index: IndexCustomerName                                     */
/*==============================================================*/
create index IndexCustomerName on Customer (
LastName ASC,
FirstName ASC
)
;

/*==============================================================*/
/* Table: ShopOrder                                             */
/*==============================================================*/
create table ShopOrder (
   Id                int                not null,
   OrderDate         datetime           not null,
```

```
  OrderNumber         varchar(10)        null,
  CustomerId          int                not null,
  TotalAmount         decimal(12,2)      null default 0,
  constraint PK_ORDER primary key (Id)
)
;

/*==============================================================*/
/* Index: IndexOrderCustomerId                                  */
/*==============================================================*/
create index IndexOrderCustomerId on ShopOrder (
CustomerId ASC
)
;

/*==============================================================*/
/* Index: IndexOrderOrderDate                                   */
/*==============================================================*/
create index IndexOrderOrderDate on ShopOrder (
OrderDate ASC
)
;

/*==============================================================*/
/* Table: OrderItem                                             */
/*==============================================================*/
create table OrderItem (
  Id                int                not null,
  OrderId           int                not null,
  ProductId         int                not null,
  UnitPrice         decimal(12,2)      not null default 0,
  Quantity          int                not null default 1,
  constraint PK_ORDERITEM primary key (Id)
)
;

/*==============================================================*/
/* Index: IndexOrderItemOrderId                                 */
/*==============================================================*/
create index IndexOrderItemOrderId on OrderItem (
OrderId ASC
)
;

/*==============================================================*/
```

```
/* Index: IndexOrderItemProductId                          */
/*==============================================================*/
create index IndexOrderItemProductId on OrderItem (
ProductId ASC
)
;

/*==============================================================*/
/* Table: Product                                       */
/*==============================================================*/
create table Product (
   Id                int                not null,
   ProductName       varchar(50)        not null,
   SupplierId        int                not null,
   UnitPrice         decimal(12,2)      null default 0,
   Package           varchar(30)        null,
   IsDiscontinued    bit                not null default 0,
   constraint PK_PRODUCT primary key (Id)
)
;

/*==============================================================*/
/* Index: IndexProductSupplierId                           */
/*==============================================================*/
create index IndexProductSupplierId on Product (
SupplierId ASC
)
;

/*==============================================================*/
/* Index: IndexProductName                                 */
/*==============================================================*/
create index IndexProductName on Product (
ProductName ASC
)
;

/*==============================================================*/
/* Table: Supplier                                      */
/*==============================================================*/
create table Supplier (
   Id                int                not null,
   CompanyName       varchar(40)        not null,
   ContactName       varchar(50)        null,
   ContactTitle      varchar(40)        null,
```

```
  City             varchar(40)        null,
  Country          varchar(40)        null,
  Phone            varchar(30)        null,
  Fax              varchar(30)        null,
  constraint PK_SUPPLIER primary key (Id)
)
;

/*==============================================================*/
/* Index: IndexSupplierName                                     */
/*==============================================================*/
create index IndexSupplierName on Supplier (
CompanyName ASC
)
;

/*==============================================================*/
/* Index: IndexSupplierCountry                                  */
/*==============================================================*/
create index IndexSupplierCountry on Supplier (
Country ASC
)
;

alter table ShopOrder
  add constraint FK_ORDER_REFERENCE_CUSTOMER foreign key (CustomerId)
    references Customer (Id)
;

alter table OrderItem
  add constraint FK_ORDERITE_REFERENCE_ORDER foreign key (OrderId)
    references ShopOrder (Id)
;

alter table OrderItem
  add constraint FK_ORDERITE_REFERENCE_PRODUCT foreign key (ProductId)
    references Product (Id)
;

alter table Product
  add constraint FK_PRODUCT_REFERENCE_SUPPLIER foreign key (SupplierId)
    references Supplier (Id)
;
```

QUERIES:

```
SELECT * FROM agents;
```

Purpose: This is used to give details of all agents of the company.

```
A007|Ramasundar|Bangalore|0.15|077-25814763|
A003|Alex |London|0.13|075-12458969|
A008|Alford|New York|0.12|044-25874365|
A011|Ravi Kumar|Bangalore|0.15|077-45625874|
A010|Santakumar|Chennai|0.14|007-22388644|
A012|Lucida|San Jose|0.12|044-52981425|
A005|Anderson|Brisban|0.13|045-21447739|
A001|Subbarao|Bangalore|0.14|077-12346674|
A002|Mukesh|Mumbai|0.11|029-12358964|
A006|McDen|London|0.15|078-22255588|
A004|Ivan|Torento|0.15|008-22544166|
A009|Benjamin|Hampshair|0.11|008-22536178|
```

```
SELECT * FROM customer;
```

Purpose: This is used to give details of all the customers of the company/

```
C00013|Holmes|London|London|UK|2|6000|5000|7000|4000|BBBBBBB|A003
C00001|Micheal|New York|New York|USA|2|3000|5000|2000|6000|CCCCCCC|A008
C00020|Albert|New York|New York|USA|3|5000|7000|6000|6000|BBBBSBB|A008
C00025|Ravindran|Bangalore|Bangalore|India|2|5000|7000|4000|8000|AVAVAVA|A011
C00024|Cook|London|London|UK|2|4000|9000|7000|6000|FSDDSDF|A006
C00015|Stuart|London|London|UK|1|6000|8000|3000|11000|GFSGERS|A003
C00002|Bolt|New York|New York|USA|3|5000|7000|9000|3000|DDNRDRH|A008
C00018|Fleming|Brisban|Brisban|Australia|2|7000|7000|9000|5000|NHBGVFC|A005
C00021|Jacks|Brisban|Brisban|Australia|1|7000|7000|7000|7000|WERTGDF|A005
C00019|Yearannaidu|Chennai|Chennai|India|1|8000|7000|7000|8000|ZZZZBFV|A010
C00005|Sasikant|Mumbai|Mumbai|India|1|7000|11000|7000|11000|147-25896312|A002
C00007|Ramanathan|Chennai|Chennai|India|1|7000|11000|9000|9000|GHRDWSD|A010
C00022|Avinash|Mumbai|Mumbai|India|2|7000|11000|9000|9000|113-12345678|A002
C00004|Winston|Brisban|Brisban|Australia|1|5000|8000|7000|6000|AAAAAAA|A005
C00023|Karl|London|London|UK|0|4000|6000|7000|3000|AAAABAA|A006
C00006|Shilton|Torento|Torento|Canada|1|10000|7000|6000|11000|DDDDDDD|A004
C00010|Charles|Hampshair|Hampshair|UK|3|6000|4000|5000|5000|MMMMMMM|A009
C00017|Srinivas|Bangalore|Bangalore|India|2|8000|4000|3000|9000|AAAAAAB|A007
C00012|Steven|San Jose|San Jose|USA|1|5000|7000|9000|3000|KRFYGJK|A012
C00008|Karolina|Torento|Torento|Canada|1|7000|7000|9000|5000|HJKORED|A004
C00003|Martin|Torento|Torento|Canada|2|8000|7000|7000|8000|MJYURFD|A004
C00009|Ramesh|Mumbai|Mumbai|India|3|8000|7000|3000|12000|Phone No|A002
C00014|Rangarappa|Bangalore|Bangalore|India|2|8000|11000|7000|12000|AAAATGF|A001
C00016|Venkatpati|Bangalore|Bangalore|India|2|8000|11000|7000|12000|JRTVFDD|A007
C00011|Sundariya|Chennai|Chennai|India|3|7000|11000|7000|11000|PPHGRTS|A010
```

```
SELECT * FROM orders;
```

Purpose: This is used to give details of all the orders of the company

```
200100|1000|600|2008-08-01|C00013|A003|SOD
200110|3000|500|2008-04-15|C00019|A010|SOD
200107|4500|900|2008-08-30|C00007|A010|SOD
200112|2000|400|2008-05-30|C00016|A007|SOD
200113|4000|600|-2008-06-10|C00022|A002|SOD
200102|2000|300|-2008-05-25|C00012|A012|SOD
200114|3500|2000|-2008-08-15|C00002|A008|SOD
200122|2500|400|2008-09-16|C00003|A004|SOD
200118|500|100|2008-07-20|C00023|A006|SOD
200119|4000|700|2008-09-16|C00007|A010|SOD
200121|1500|600|2008-09-23|C00008|A004|SOD
200130|2500|400|2008-07-30|C00025|A011|SOD
200134|4200|1800|2008-09-25|C00004|A005|SOD
200108|4000|600|2008-02-15|C00008|A004|SOD
200103|1500|700|2008-05-15|C00021|A005|SOD
200105|2500|500|2008-07-18|C00025|A011|SOD
200109|3500|800|2008-07-30|C00011|A010|SOD
200101|3000|1000|2008-07-15|C00001|A008|SOD
200111|1000|300|2008-07-10|C00020|A008|SOD
200104|1500|500|2008-03-13|C00006|A004|SOD
200106|2500|700|2008-04-20|C00005|A002|SOD
200125|2000|600|2008-10-10|C00018|A005|SOD
200117|800|200|2008-10-20|C00014|A001|SOD
200123|500|100|2008-09-16|C00022|A002|SOD
200120|500|100|2008-07-20|C00009|A002|SOD
200116|500|100|2008-07-13|C00010|A009|SOD
200124|500|100|2008-06-20|C00017|A007|SOD
200126|500|100|2008-06-24|C00022|A002|SOD
200129|2500|500|2008-07-20|C00024|A006|SOD
200127|2500|400|2008-07-20|C00015|A003|SOD
200128|3500|1500|2008-07-20|C00009|A002|SOD
200135|2000|800|2008-09-16|C00007|A010|SOD
200131|900|150|2008-08-26|C00012|A012|SOD
200133|1200|400|2008-06-29|C00009|A002|SOD
```

```
SELECT * FROM orders WHERE CUST_CODE="C00022";
```

Purpose: This is used to give details of all the orders belonging to a customer whose CUST_CODE is C00022

```
200113|4000|600|2008-06-10|C00022|A002|SOD
200123|500|100|2008-09-16|C00022|A002|SOD
200126|500|100|2008-06-24|C00022|A002|SOD
```

```
SELECT * FROM orders WHERE AGENT_CODE = 'A002';
```

Purpose: This is used to give details of all orders of agent with AGENT_CODE = A002

```
200113|4000|600|-2008-06-10|C00022|A002|SOD
200106|2500|700|2008-04-20|C00005|A002|SOD
200123|500|100|2008-09-16|C00022|A002|SOD
200120|500|100|2008-07-20|C00009|A002|SOD
200126|500|100|2008-06-24|C00022|A002|SOD
200128|3500|1500|2008-07-20|C00009|A002|SOD
200133|1200|400|2008-06-29|C00009|A002|SOD
```

```
SELECT * FROM orders WHERE ORD_DATE='2008-07-13';
```

Purpose: This is used to give details of all orders that took place on 13th July 2008

```
200113|4000|600|2008-06-10|C00022|A002|SOD
200106|2500|700|2008-04-20|C00005|A002|SOD
200123|500|100|2008-09-16|C00022|A002|SOD
200120|500|100|2008-07-20|C00009|A002|SOD
200126|500|100|2008-06-24|C00022|A002|SOD
200128|3500|1500|2008-07-20|C00009|A002|SOD
200133|1200|400|2008-06-29|C00009|A002|SOD
```

```
SELECT * FROM agents WHERE WORKING_AREA='London';
```

Purpose: This is used to show details of all agents working in London

```
A003|Alex |London|0.13|075-12458969|
A006|McDen|London|0.15|078-22255588|
```

```
SELECT * FROM agents order by COMMISSION ASC;
```

Purpose: This is used to show details of all agents arranged in ascending order of their commission. Can be used during audits or performance discussion meetings.

```
A002|Mukesh|Mumbai|0.11|029-12358964|
A009|Benjamin|Hampshair|0.11|008-22536178|
A008|Alford|New York|0.12|044-25874365|
A012|Lucida|San Jose|0.12|044-52981425|
A003|Alex |London|0.13|075-12458969|
A005|Anderson|Brisban|0.13|045-21447739|
A010|Santakumar|Chennai|0.14|007-22388644|
A001|Subbarao|Bangalore|0.14|077-12346674|
A007|Ramasundar|Bangalore|0.15|077-25814763|
A011|Ravi Kumar|Bangalore|0.15|077-45625874|
A006|McDen|London|0.15|078-22255588|
A004|Ivan|Torento|0.15|008-22544166|
```

```
SELECT AVG(OUTSTANDING_AMT) FROM customer WHERE OUTSTANDING_AMT>5000;
```

Purpose: This is used to show the average outstanding amount of all customers filtered by a range greater than 5000

```
9000.0
```

```
SELECT CUST_CODE, CUST_NAME, CUST_COUNTRY, MAX(OUTSTANDING_AMT) FROM customer;
```

Purpose: This is used to show details of that customer who has the maximum outstanding amount.

```
C00009|Ramesh|India|12000
```

```
SELECT * FROM customer GROUP  BY AGENT_CODE ORDER  BY COUNT(*) DESC;
```

Purpose: This shows details of all customers grouped by AGENT_CODE ordered in descending order.

```
C00009|Ramesh|Mumbai|Mumbai|India|3|8000|7000|3000|12000|Phone No|A002
C00003|Martin|Torento|Torento|Canada|2|8000|7000|7000|8000|MJYURFD|A004
C00004|Winston|Brisban|Brisban|Australia|1|5000|8000|7000|6000|AAAAAAA|A005
C00002|Bolt|New York|New York|USA|3|5000|7000|9000|3000|DDNRDRH|A008
C00011|Sundariya|Chennai|Chennai|India|3|7000|11000|7000|11000|PPHGRTS|A010
C00015|Stuart|London|London|UK|1|6000|8000|3000|11000|GFSGERS|A003
C00023|Karl|London|London|UK|0|4000|6000|7000|3000|AAAABAA|A006
C00016|Venkatpati|Bangalore|Bangalore|India|2|8000|11000|7000|12000|JRTVFDD|A007
C00014|Rangarappa|Bangalore|Bangalore|India|2|8000|11000|7000|12000|AAAATGF|A001
C00010|Charles|Hampshair|Hampshair|UK|3|6000|4000|5000|5000|MMMMMMM|A009
```

```
C00025|Ravindran|Bangalore|Bangalore|India|2|5000|7000|4000|8000|AVAVAVA|A011
C00012|Steven|San Jose|San Jose|USA|1|5000|7000|9000|3000|KRFYGJK|A012
```

```
SELECT * FROM customer where CUST_CODE IN (SELECT CUST_CODE FROM orders
WHERE ORD_AMOUNT>4000);
```

Purpose: This is a nested query. It shows details of all customers whose order amount is greater than 4000

```
C00004|Winston|Brisban|Brisban|Australia|1|5000|8000|7000|6000|AAAAAAA|A005
C00007|Ramanathan|Chennai|Chennai|India|1|7000|11000|9000|9000|GHRDWSD|A010
```

```
SELECT customer.CUST_NAME, orders.ORD_AMOUNT, orders.ORD_DATE from orders
INNER JOIN customer on orders.CUST_CODE=customer.CUST_CODE;
```

Purpose: This query does an inner join of the Customer table and the order table.

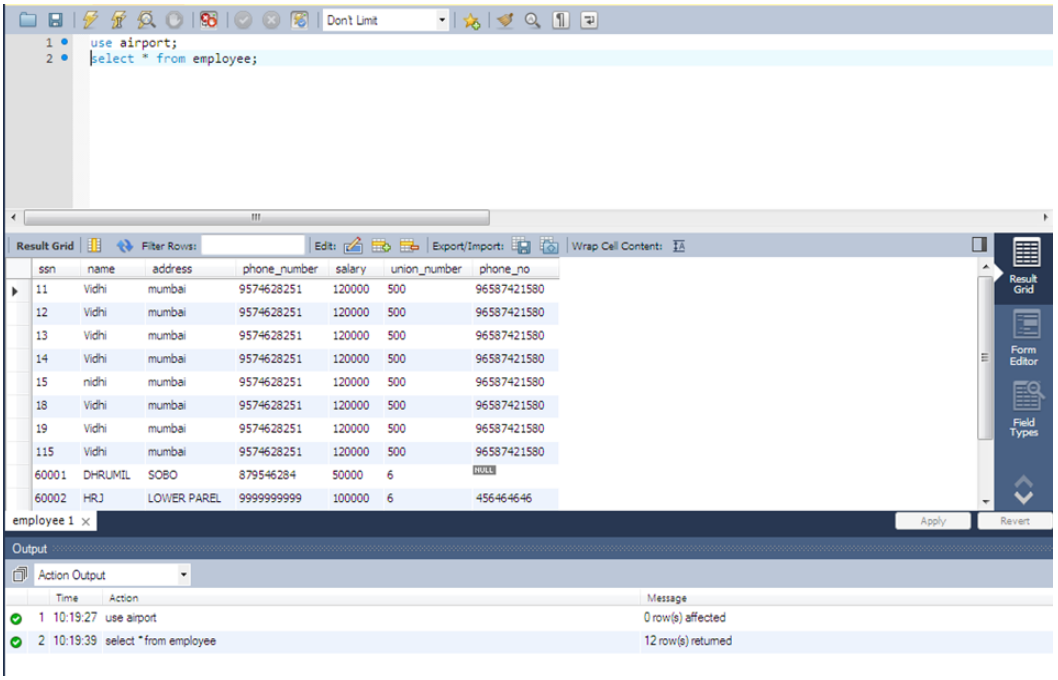
```
Holmes|1000|2008-08-01
Yearannaidu|3000|2008-04-15
Ramanathan|4500|2008-08-30
Venkatpati|2000|2008-05-30
Avinash|4000|-2008-06-10
Steven|2000|-2008-05-25
Bolt|3500|-2008-08-15
Martin|2500|2008-09-16
Karl|500|2008-07-20
Ramanathan|4000|2008-09-16
Karolina|1500|2008-09-23
Ravindran|2500|2008-07-30
Winston|4200|2008-09-25
Karolina|4000|2008-02-15
Jacks|1500|2008-05-15
Ravindran|2500|2008-07-18
Sundariya|3500|2008-07-30
Micheal|3000|2008-07-15
Albert|1000|2008-07-10
Shilton|1500|2008-03-13
Sasikant|2500|2008-04-20
Fleming|2000|2008-10-10
Rangarappa|800|2008-10-20
```

Avinash|500|2008-09-16
Ramesh|500|2008-07-20
Charles|500|2008-07-13
Srinivas|500|2008-06-20
Avinash|500|2008-06-24
Cook|2500|2008-07-20
Stuart|2500|2008-07-20
Ramesh|3500|2008-07-20
Ramanathan|2000|2008-09-16
Steven|900|2008-08-26
Ramesh|1200|2008-06-29

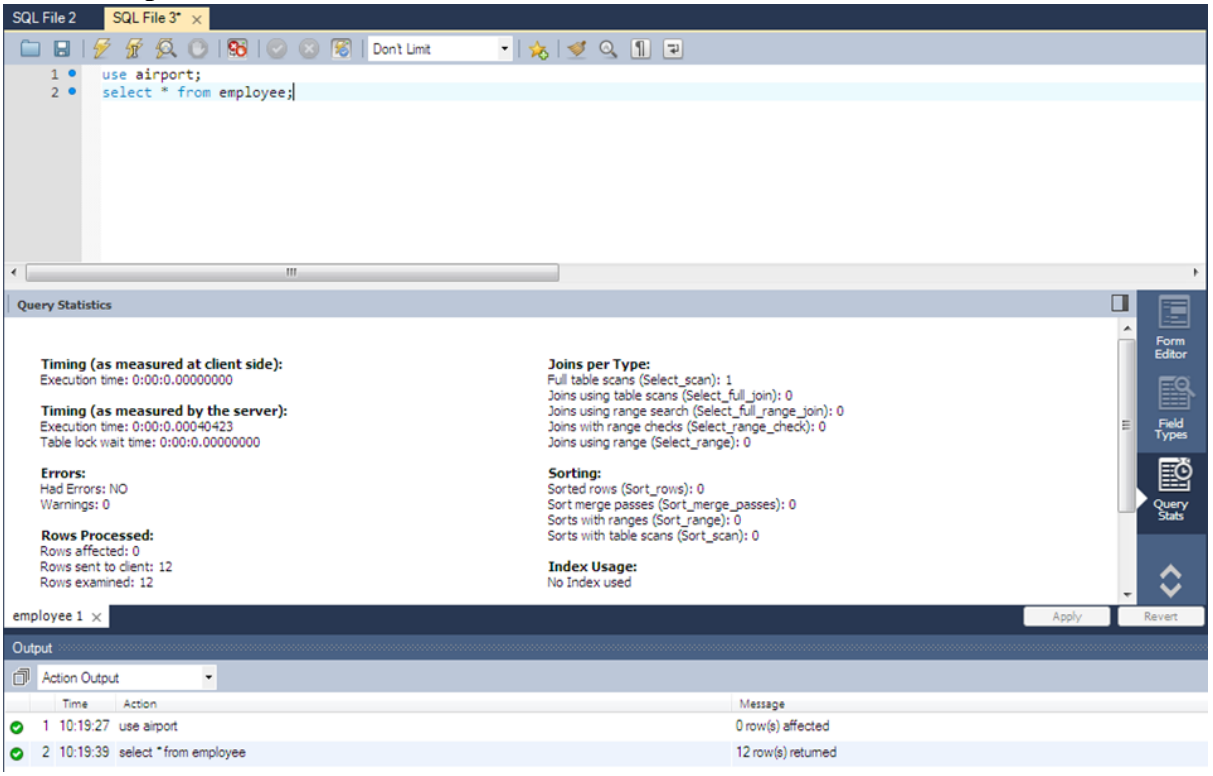
OPTIMIZATION

TABLE LEVEL OPTIMIZATION

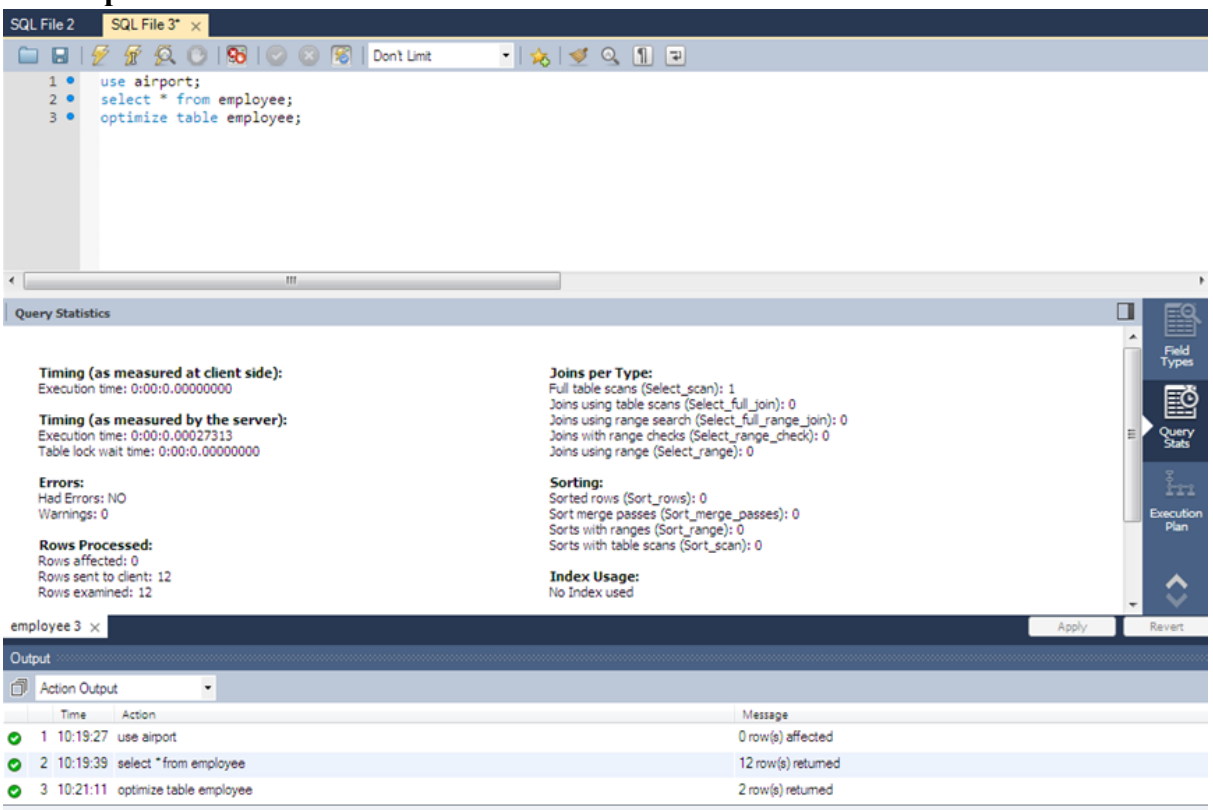
Normal Query : select * from employee;



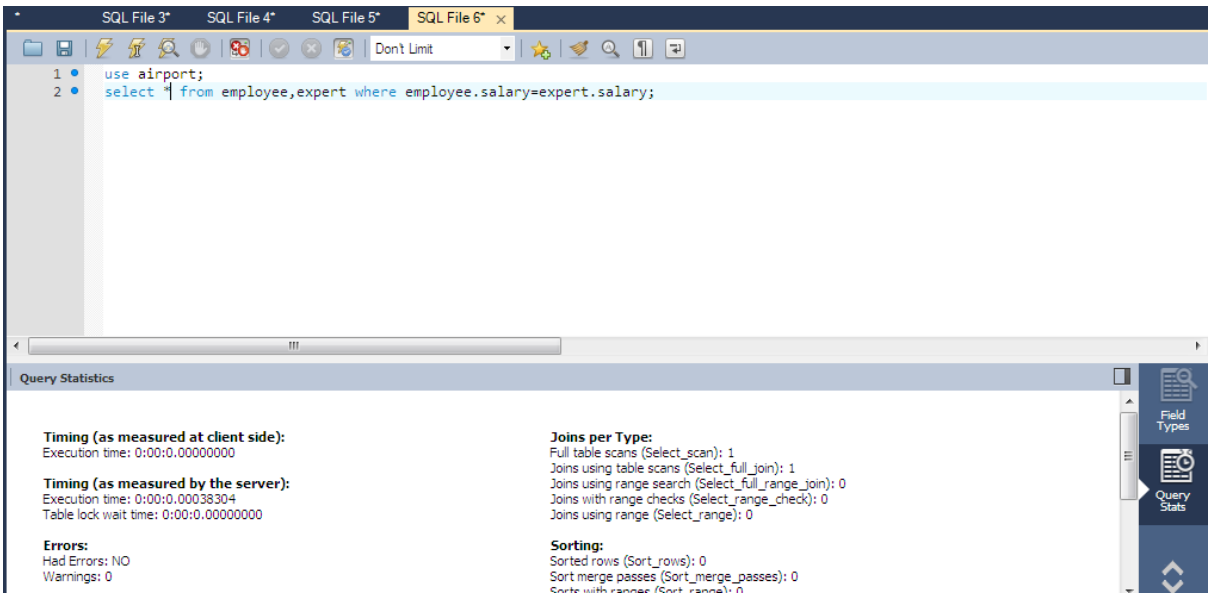
Without Optimization



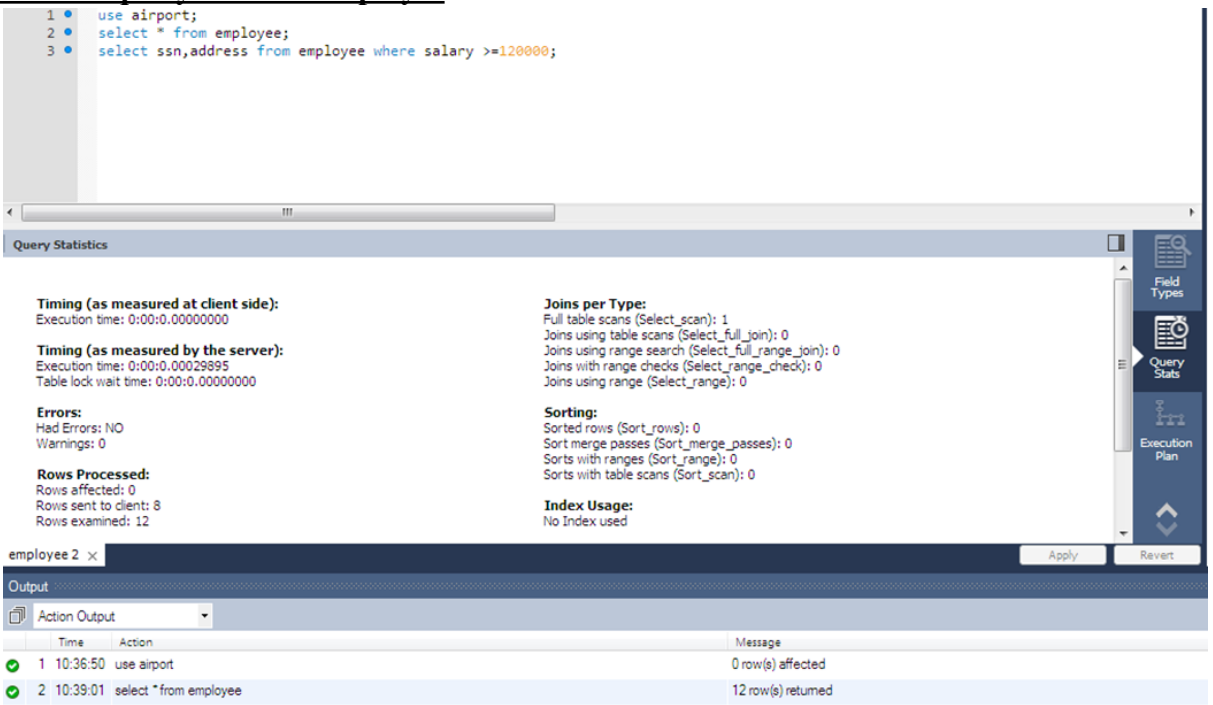
After Optimization



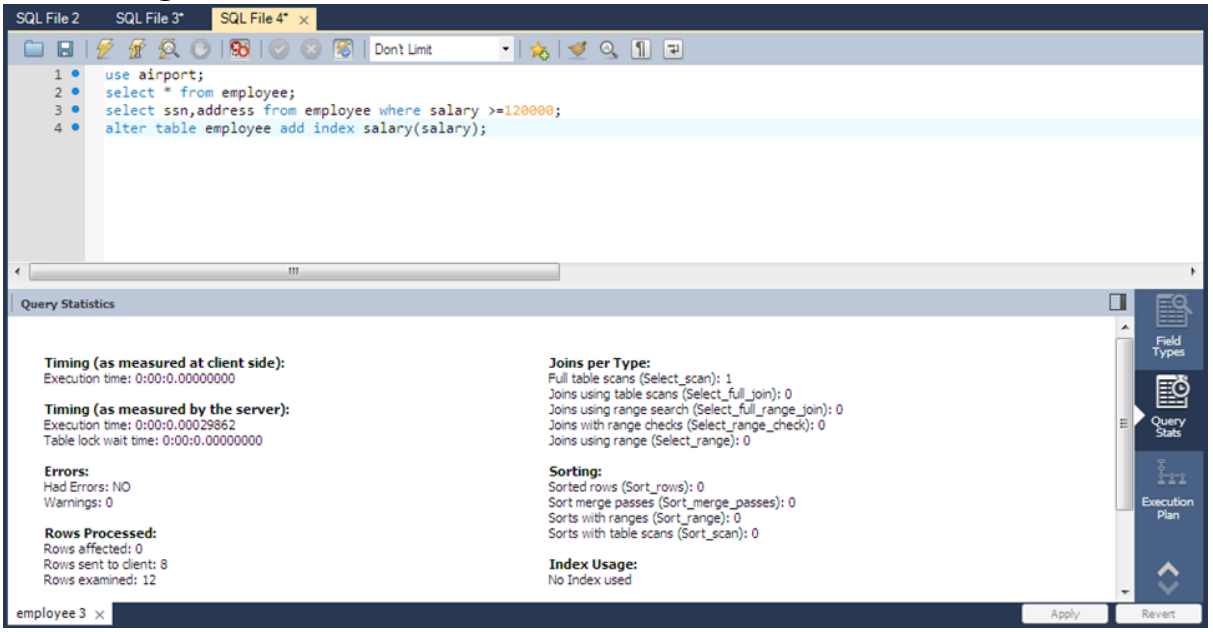
INDEX LEVEL OPTIMIZATION
Before adding index join query



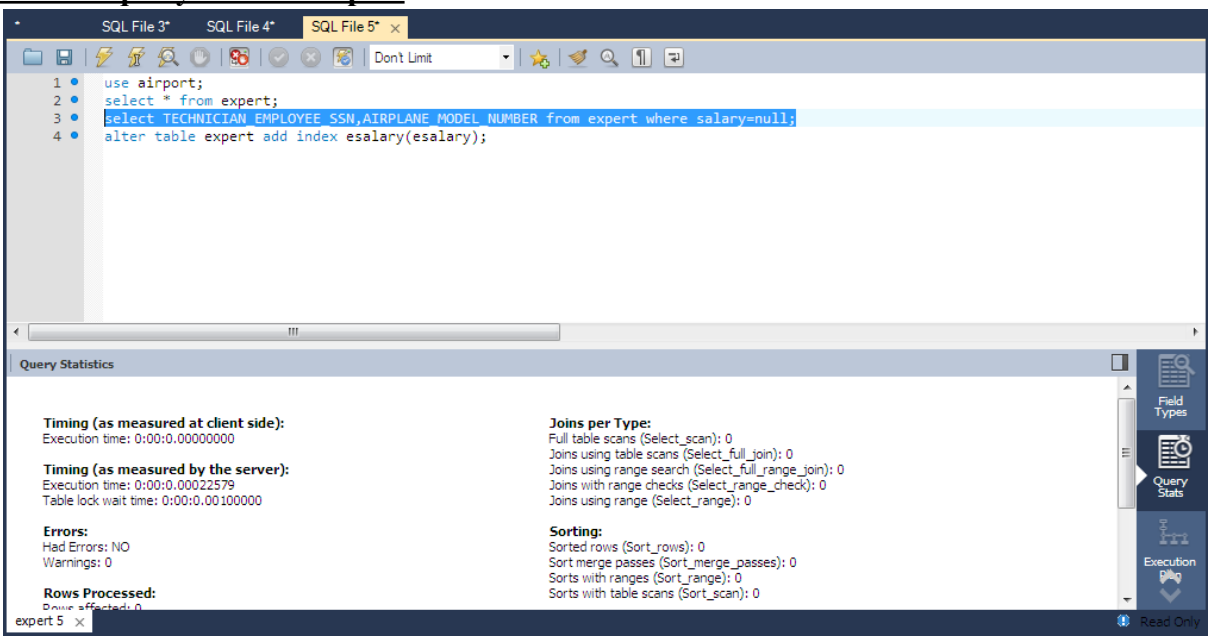
Normal query on table employee



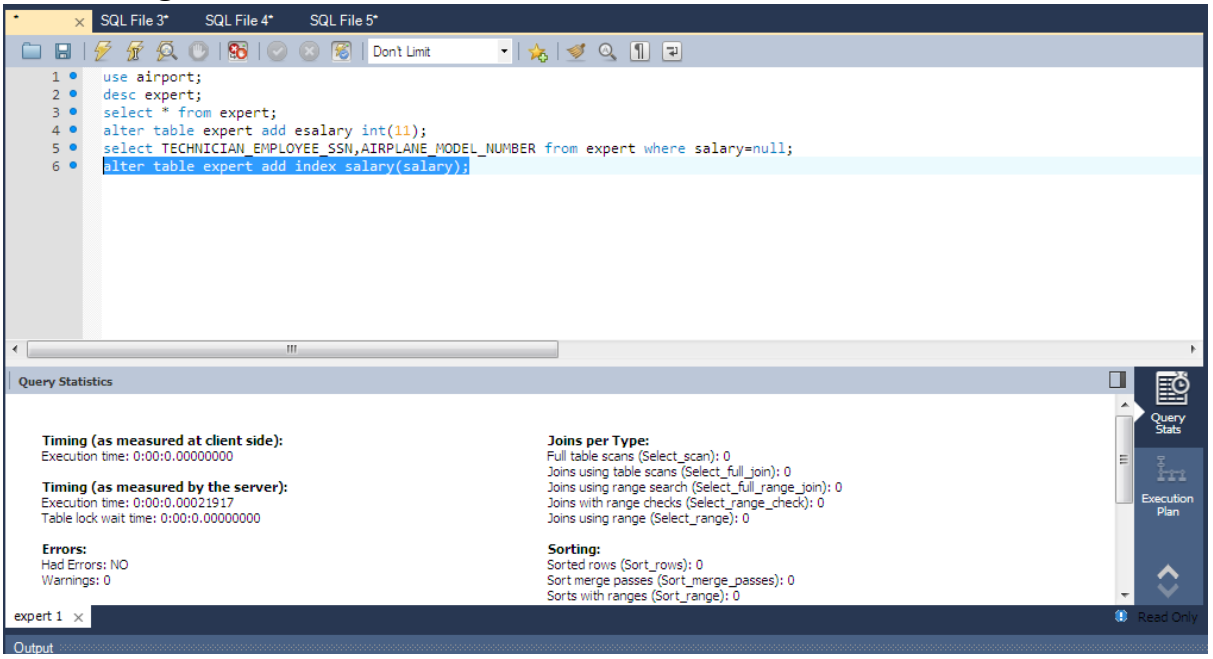
After adding index



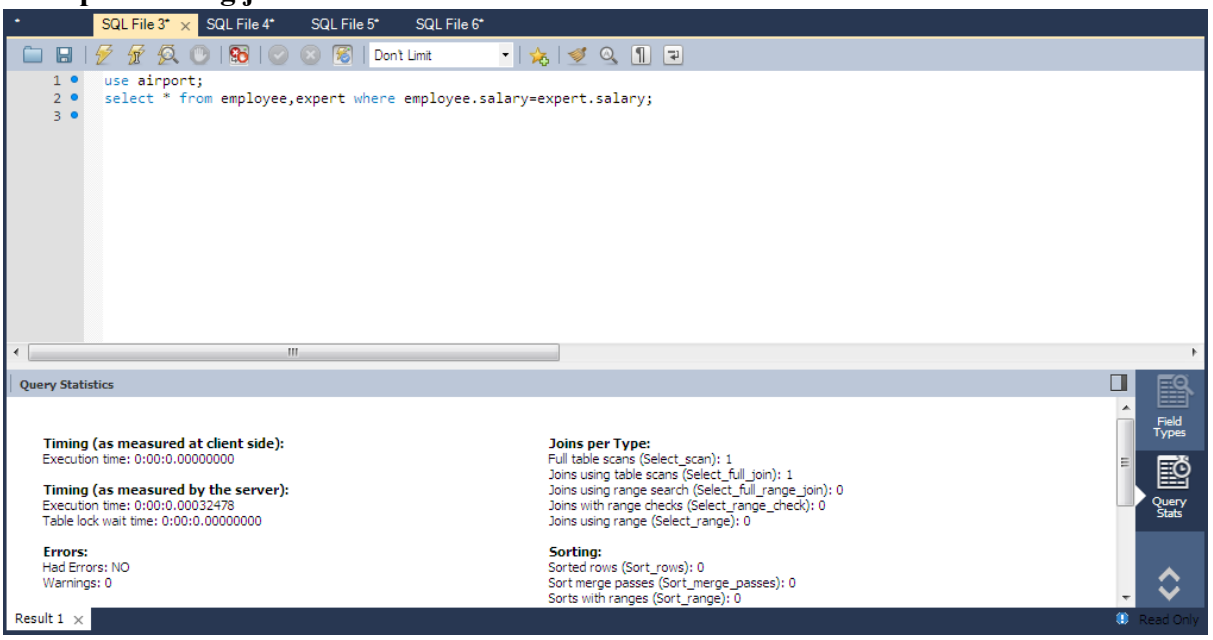
Normal query on table expert



After adding index



After performing join



INDEXING

A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

While creating an index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.

Practically, indexes are also a type of tables, which keep the primary key or index field and a pointer to each record into the actual table.

The users cannot see the indexes, they are just used to speed up queries and will be used by the Database Search Engine to locate records very fast.

The INSERT and UPDATE statements take more time on tables having indexes, whereas the SELECT statements become fast on those tables. The reason is that while doing insert or update, a database needs to insert or update the index values as well.

By default, MySQL allows index type **BTREE** if we have not specified the type of index. The following table shows the different types of an index based on the storage engine of the table.

SYNTAX:

CREATE INDEX [index_name] **ON** [table_name] (**column** names)

CODE:

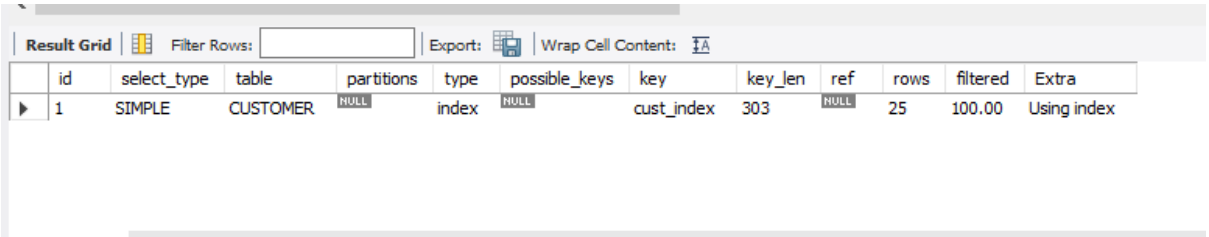
```
CREATE INDEX cust_index ON CUSTOMER (CUST_NAME, CUST_CITY);
```

```
SHOW INDEXES FROM CUSTOMER;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expr
customer	0	PRIMARY	1	CUST_CODE	A	25	NULL	NULL		BTREE			YES	NULL
customer	1	cust_index	1	CUST_NAME	A	25	NULL	NULL		BTREE			YES	NULL
customer	1	cust_index	2	CUST_CITY	A	25	NULL	NULL	YES	BTREE			YES	NULL

```
EXPLAIN SELECT CUST_NAME FROM CUSTOMER;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	CUSTOMER	NULL	index	NULL	cust_index	303	NULL	25	100.00	Using index

---

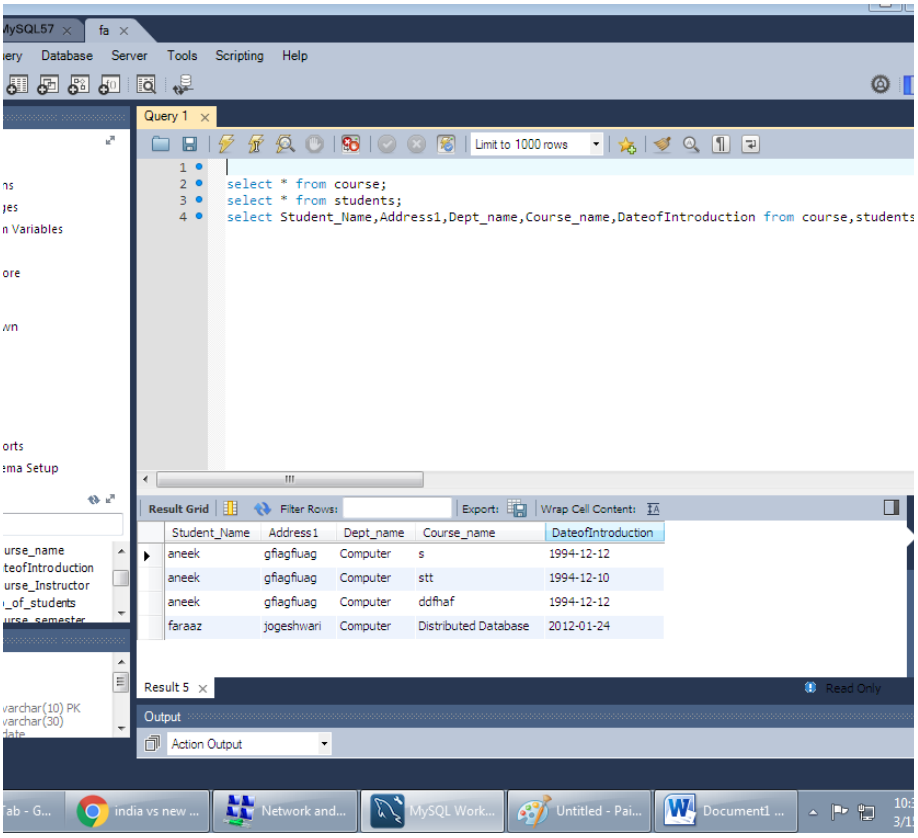
Conclusion: We learnt about SQL queries and how to optimise it. We then learnt about the different optimization queries and indexing.

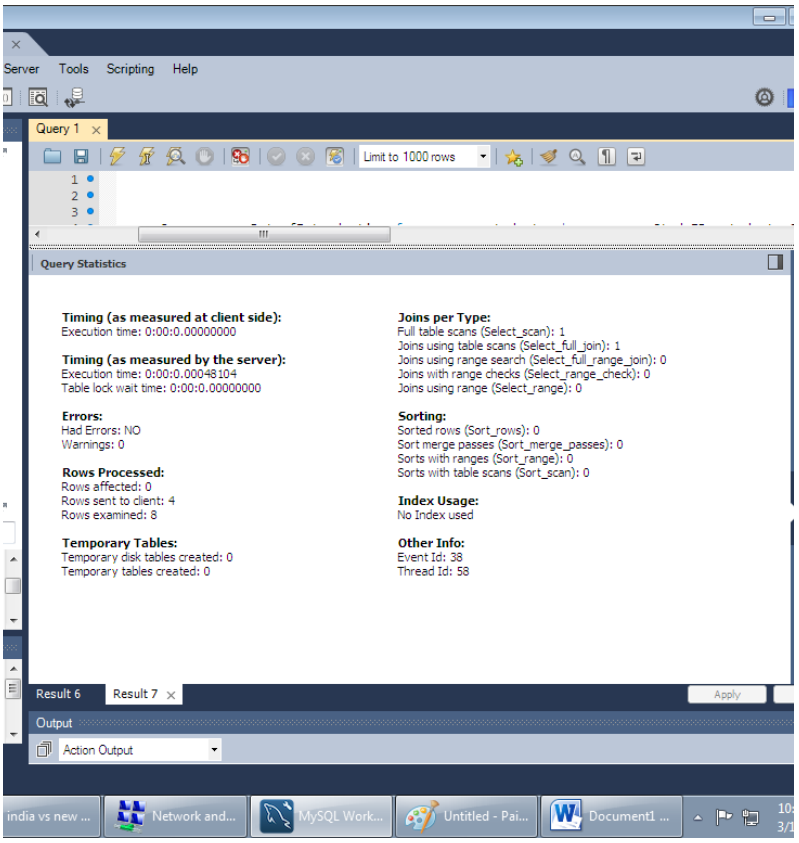
EXPERIMENT 3

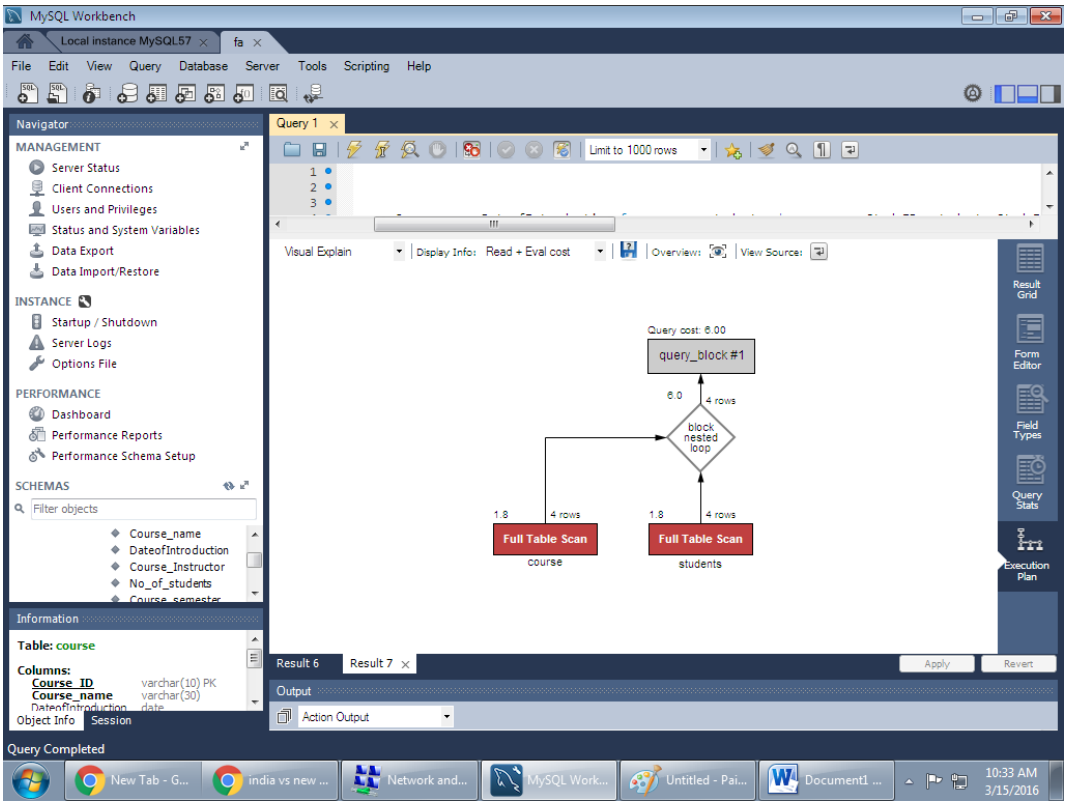
Aim: Implementation of query monitor.

Labwork:

Join query on course and students tables
select
Student_Name,Address1,Dept_name,Course_name,DateofIntroduction from
course,students where course.Stud_ID= students.Stud_ID;

1. 


Server Tools Scripting Help
Query 1
Limit to 1000 rows
Query Statistics
Timing (as measured at client side):
Execution time: 0:00:0.00000000
Timing (as measured by the server):
Execution time: 0:00:0.00048104
Table lock wait time: 0:00:0.00000000
Errors:
Had Errors: NO
Warnings: 0
Rows Processed:
Rows affected: 0
Rows sent to client: 4
Rows examined: 8
Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0
Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 1
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0
Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0
Index Usage:
No Index used
Other Info:
Event Id: 38
Thread Id: 58
Result 6
Result 7
Apply
Output
Action Output


MySQL Workbench
Local instance MySQL57
fa
File Edit View Query Database Server Tools Scripting Help
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
Course_name
DateofIntroduction
Course_Instructor
No_of_students
Information
Table: course
Columns:
Course_ID varchar(10) PK
Course_name varchar(30)
Object Info
Session
Query 1
Limit to 1000 rows
Visual Explain
Display Info: Read + Eval cost
Overview:
View Source:
Query cost: 6.00
query_block #1
6.0
4 rows
block nested loop
1.8
4 rows
Full Table Scan
course
1.8
4 rows
Full Table Scan
students
Result Grid
Form Editor
Field Types
Query Stats
Execution Plan
Result 6
Result 7
Apply
Revert
Output
Action Output
Query Completed
10:33 AM
3/15/2016

What is an execution plan?

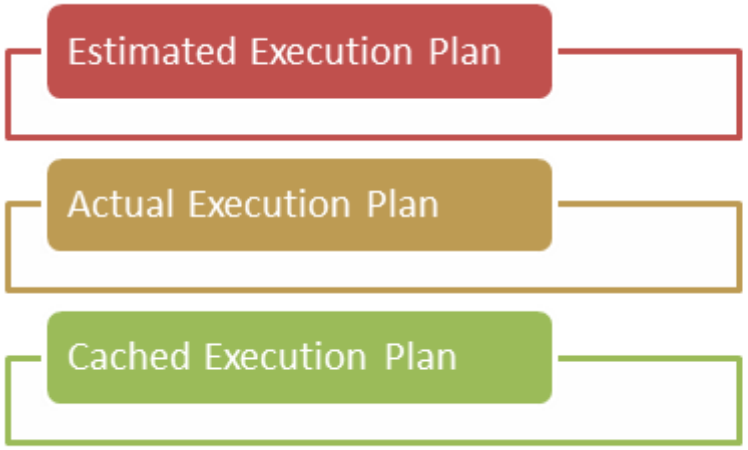
It is a graphical representation of the operation performed by the SQL server database engine. Execution plan is generated by query optimizer. It tells us the flow of the query.

Execution plan lets us know how a query will execute on the database engine to return some results.

Types of execution plan

Most people think that there are only two types of execution plans, estimated execution plan and actual execution plan, but that's not true, there are three types of execution plans as mentioned below,

SQL Query Execution Plan



Estimated execution plan

This execution plan shows us SQL optimizer view. Basically it provides us an estimation of query execution process to get the query results. It is a compiled plan, which means query does not get executed to get execution plan.

To get a query's estimated execution plan we have to do the following steps,

- Switch to SSMS (Sql Server Management Studio)
- highlight the query
- click on Query
- click Display Estimated Execution Plan

Conclusion: In this experiment we have learned what is an execution, how many types of execution plans sql has and the display options of the plans.

EXPERIMENT 4

PARTITIONING

THEORY:
Partitioning in MySQL is used to split or partition the rows of a table into separate tables in different locations, but still, it is treated as a single table. It distributes the portions of the table's data across a file system based on the rules we have set as our requirement. The rule that we have set to accomplish the division of table data is called a partitioning function (modulus, a linear or internal hashing function, etc.). The selected function is based on the partitioning type we have specified and takes a user-supplied expression as its parameter. The user- expression can be a column value or a function acting on column values, depending on the type of partitioning used.

MySQL has mainly two forms of partitioning:

1. Horizontal Partitioning

This partitioning split the rows of a table into multiple tables based on our logic. In horizontal partitioning, the number of columns is the same in each table, but no need to keep the same number of rows. It physically divides the table but logically treated as a whole. Currently, MySQL supports this partitioning only.

2. Vertical Partitioning

This partitioning splits the table into multiple tables with fewer columns from the original table. It uses an additional table to store the remaining columns. Currently, MySQL does not provide supports for this partitioning.

Benefits of Partitioning

The following are the benefits of partitioning in MySQL:

- It optimizes the query performance. When we query on the table, it scans only the portion of a table that will satisfy the particular statement.
- It is possible to store extensive data in one table that can be held on a single disk or file system partition.
- It provides more control to manage the data in your database.

How can we partition the table in MySQL?

We can create a partition in MySQL using the CREATE TABLE or ALTER TABLE statement. Below is the syntax of creating partition using CREATE TABLE command:

```
CREATE TABLE [IF NOT EXISTS] table_name
(column_definitions)
[table_options]
[partition_options]
```

The below is the syntax of creating partition using ALTER TABLE command:

```
ALTER TABLE [IF EXISTS] tab_name
(colm_definitions)
[tab_options]
[partition_options]
```

Types of MySQL Partitioning

MySQL has mainly six types of partitioning, which are given below:

- RANGE Partitioning
- LIST Partitioning
- COLUMNS Partitioning
- HASH Partitioning
- KEY Partitioning
- Subpartitioning

MySQL RANGE Partitioning

This partitioning allows us to partition the rows of a table based on column values that fall within a specified range. The given range is always in a contiguous form but should not overlap each other, and also uses the VALUES LESS THAN operator to define the ranges.

CODE:

```
CREATE TABLE Sales ( cust_id INT NOT NULL, name VARCHAR(40),
store_id VARCHAR(20) NOT NULL, bill_no INT NOT NULL,
bill_date DATE PRIMARY KEY NOT NULL, amount DECIMAL(8,2) NOT NULL)
PARTITION BY RANGE (year(bill_date))(
PARTITION p0 VALUES LESS THAN (2016),
```

```
PARTITION p1 VALUES LESS THAN (2017),
PARTITION p2 VALUES LESS THAN (2018),
PARTITION p3 VALUES LESS THAN (2020));

INSERT INTO Sales VALUES
(1, 'Mike', 'S001', 101, '2015-01-02', 125.56),
(2, 'Robert', 'S003', 103, '2015-01-25', 476.50),
(3, 'Peter', 'S012', 122, '2016-02-15', 335.00),
(4, 'Joseph', 'S345', 121, '2016-03-26', 787.00),
(5, 'Harry', 'S234', 132, '2017-04-19', 678.00),
(6, 'Stephen', 'S743', 111, '2017-05-31', 864.00),
(7, 'Jacson', 'S234', 115, '2018-06-11', 762.00),
(8, 'Smith', 'S012', 125, '2019-07-24', 300.00),
(9, 'Adam', 'S456', 119, '2019-08-02', 492.20);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Sales';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
sales	p0	2	8192	16384
sales	p1	2	8192	16384
sales	p2	2	8192	16384
sales	p3	3	5461	16384

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01600000

Timing (as measured by the server):
Execution time: 0:00:0.00683610
Table lock wait time: 0:00:0.00396400

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 160
Thread Id: 49

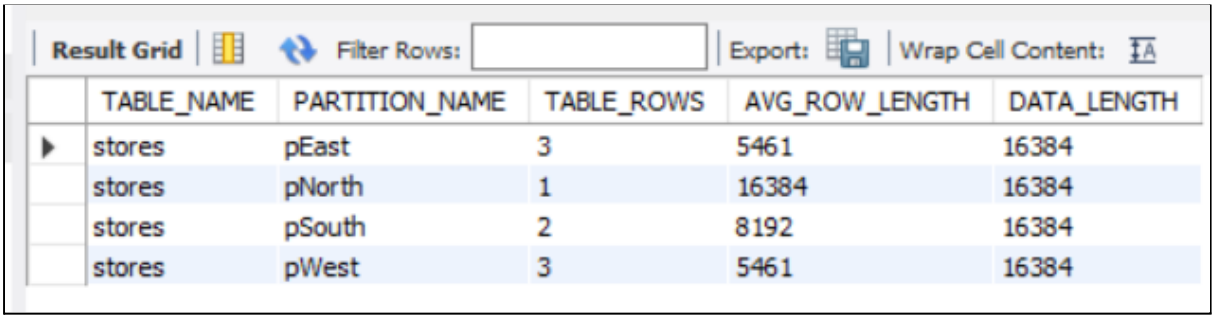
MySQL LIST Partitioning

It is the same as Range Partitioning. Here, the partition is defined and selected based on columns matching one of a set of discrete value lists rather than a set of a contiguous range of values. It is performed by the PARTITION BY LIST(exp) clause. The exp is an expression or column value that returns an integer value. The VALUES IN(value_lists) statement will be used to define each partition.

CODE:

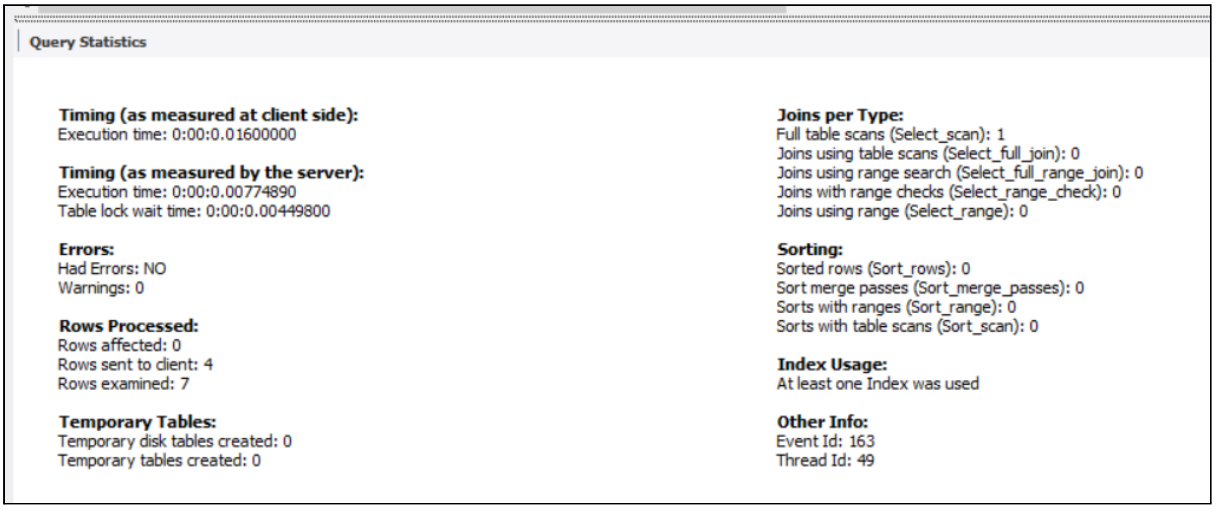
```
CREATE TABLE Stores (
    cust_name VARCHAR(40),
    bill_no VARCHAR(20) NOT NULL,
    store_id INT PRIMARY KEY NOT NULL,
    bill_date DATE NOT NULL,
    amount DECIMAL(8,2) NOT NULL
)
PARTITION BY LIST(store_id) (
PARTITION pEast VALUES IN (101, 103, 105),
PARTITION pWest VALUES IN (102, 104, 106),
PARTITION pNorth VALUES IN (107, 109, 111),
PARTITION pSouth VALUES IN (108, 110, 112));

INSERT INTO Stores VALUES
("Mike", "1", 101, "2015-01-25", 100.00),
("Joseph", "2", 102, "2015-01-25", 100.00),
("Robert", "3", 103, "2015-01-25", 100.00),
("Peter", "4", 104, "2015-01-25", 100.00),
("Joseph", "5", 105, "2015-01-25",100.00),
("Harry", "6", 106, "2015-01-25", 100.00),
("Jacson", "7", 107, "2015-01-25", 100.00),
("Smith", "8", 108, "2015-01-25", 100.00),
("Adam", "9", 110, "2015-01-25", 100.00);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Stores';
```

OUTPUT:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
stores	pEast	3	5461	16384
stores	pNorth	1	16384	16384
stores	pSouth	2	8192	16384
stores	pWest	3	5461	16384



Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01600000

Timing (as measured by the server):
Execution time: 0:00:0.00774890
Table lock wait time: 0:00:0.00449800

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 163
Thread Id: 49

MySQL HASH Partitioning

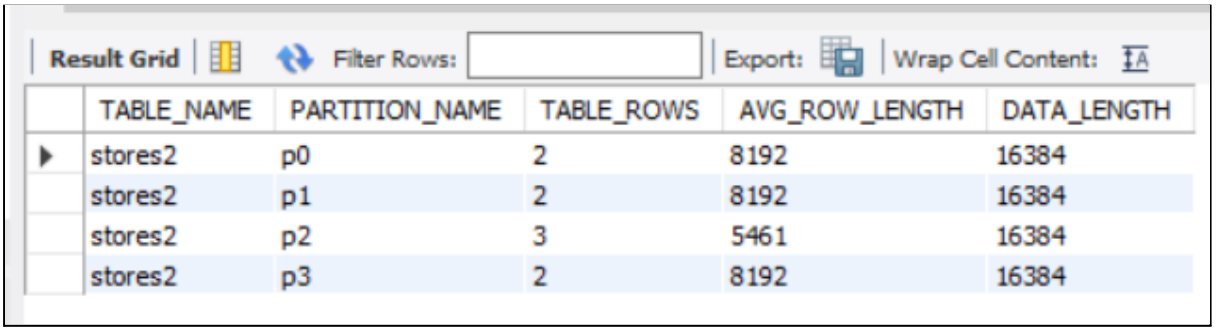
This partitioning is used to distribute data based on a predefined number of partitions. In other words, it splits the table as of the value returned by the user-defined expression. It is mainly used to distribute data evenly into the partition. It is performed with the PARTITION BY HASH(expr) clause. Here, we can specify a column value based on the column_name to be hashed and the number of partitions into which the table is divided.

CODE:

```
CREATE TABLE Stores2 (
    cust_name VARCHAR(40),
    bill_no VARCHAR(20) NOT NULL,
    store_id INT PRIMARY KEY NOT NULL,
    bill_date DATE NOT NULL,
    amount DECIMAL(8,2) NOT NULL
)
PARTITION BY HASH(store_id)
PARTITIONS 4;

INSERT INTO Stores2 VALUES
```

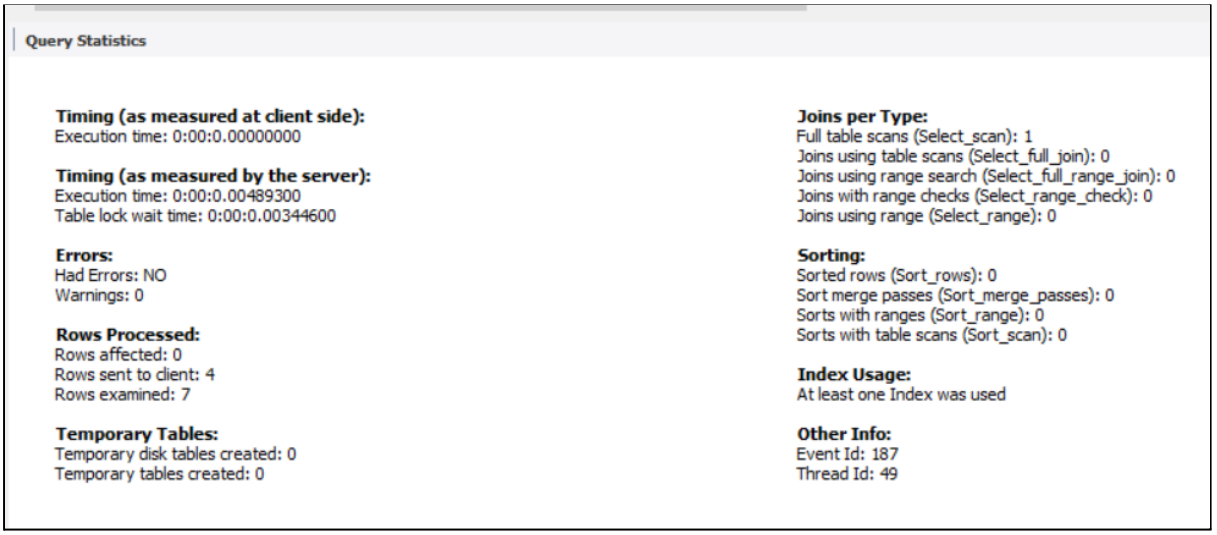
```
("Mike", "1", 101, "2015-01-25", 100.00),
("Joseph", "2", 102, "2015-01-25", 100.00),
("Robert", "3", 103, "2015-01-25", 100.00),
("Peter", "4", 104, "2015-01-25", 100.00),
("Joseph", "5", 105, "2015-01-25",100.00),
("Harry", "6", 106, "2015-01-25", 100.00),
("Jacson", "7", 107, "2015-01-25", 100.00),
("Smith", "8", 108, "2015-01-25", 100.00),
("Adam", "9", 110, "2015-01-25", 100.00);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Stores2';
```

OUTPUT:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
stores2	p0	2	8192	16384
stores2	p1	2	8192	16384
stores2	p2	3	5461	16384
stores2	p3	2	8192	16384



Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00489300
Table lock wait time: 0:00:0.00344600

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 187
Thread Id: 49

MySQL COLUMN Partitioning

This partitioning allows us to use the multiple columns in partitioning keys. The purpose of these columns is to place the rows in partitions and determine which partition will be validated for matching rows. It is mainly divided into two types:

RANGE Columns Partitioning

LIST Columns Partitioning

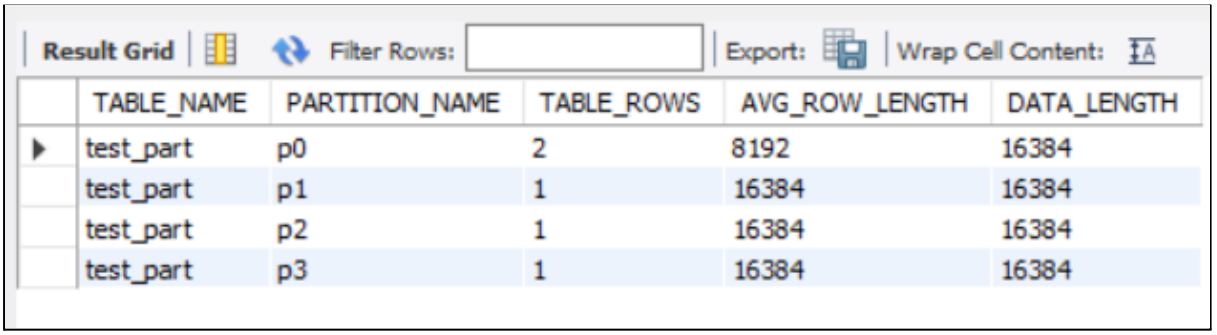
They provide supports for the use of non-integer columns to define the ranges or value lists. They support the following data types:

All Integer Types: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT.
String Types: CHAR, VARCHAR, BINARY, and VARBINARY.
DATE and DATETIME data types.
Range Column Partitioning: It is similar to the range partitioning with one difference. It defines partitions using ranges based on various columns as partition keys. The defined ranges are of column types other than an integer type.

CODE:

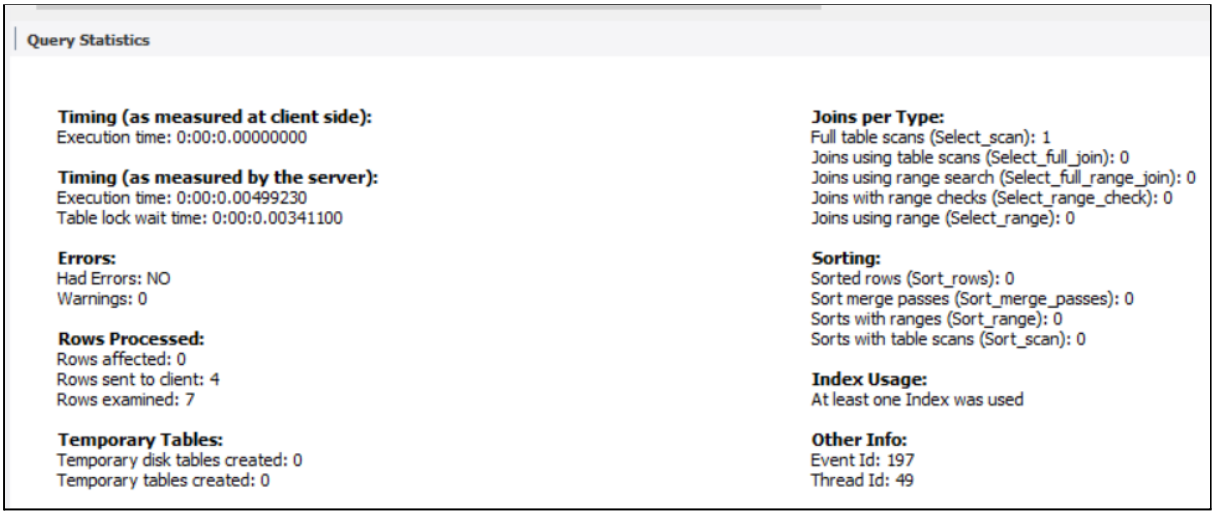
```
CREATE TABLE test_part (A INT, B CHAR(5), C INT, D INT)
PARTITION BY RANGE COLUMNS(A, B, C)
 (PARTITION p0 VALUES LESS THAN (50, 'test1', 100),
 PARTITION p1 VALUES LESS THAN (100, 'test2', 200),
 PARTITION p2 VALUES LESS THAN (150, 'test3', 300),
 PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE));

 INSERT INTO test_part VALUES
 (10, 'a', 50, 3),
 (30, 'test1', 150, 3),
 (55, 'test1', 175, 3),
 (123, 'test2', 233, 3),
 (160, 'test4', 333, 3);

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'test_part';
```

OUTPUT:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
test_part	p0	2	8192	16384
test_part	p1	1	16384	16384
test_part	p2	1	16384	16384
test_part	p3	1	16384	16384



Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00499230
Table lock wait time: 0:00:0.00341100

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 4
Rows examined: 7

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 197
Thread Id: 49

LIST Columns Partitioning

It takes a list of single or multiple columns as partition keys. It enables us to use various columns of types other than integer types as partitioning columns. In this partitioning, we can use String data types, DATE, and DATETIME columns.

CODE:

```
CREATE TABLE AgentDetail (
agent_id VARCHAR(10),
agent_name VARCHAR(40),
city VARCHAR(10))
PARTITION BY LIST COLUMNS(agent_id) (
PARTITION pNewyork VALUES IN('A1', 'A2', 'A3'),
PARTITION pTexas VALUES IN('B1', 'B2', 'B3'),
PARTITION pCalifornia VALUES IN ('C1', 'C2', 'C3'));

INSERT INTO AgentDetail VALUES
('A1', 'DummyName', 'CityName'),
('A2', 'DummyName', 'CityName'),
('A3', 'DummyName', 'CityName'),
```

```
('B1', 'DummyName', 'CityName'),
('B2', 'DummyName', 'CityName'),
('B3', 'DummyName', 'CityName'),
('C1', 'DummyName', 'CityName'),
('C2', 'DummyName', 'CityName'),
('C3', 'DummyName', 'CityName'),
('A1', 'DummyName', 'CityName'),
('C2', 'DummyName', 'CityName'),
('B1', 'DummyName', 'CityName');

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'AgentDetail';
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
agentdetail	pCalifornia	4	4096	16384
agentdetail	pNewyork	4	4096	16384
agentdetail	pTexas	4	4096	16384

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01600000

Timing (as measured by the server):
Execution time: 0:00:0.01013060
Table lock wait time: 0:00:0.00426000

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 3
Rows examined: 6

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 207
Thread Id: 49

MySQL KEY Partitioning

It is similar to the HASH partitioning where the hash partitioning uses the user-specified expression, and MySQL server supplied the hashing function for key. If we use other storage engines, the MySQL server employs its own internal hashing function that is performed by

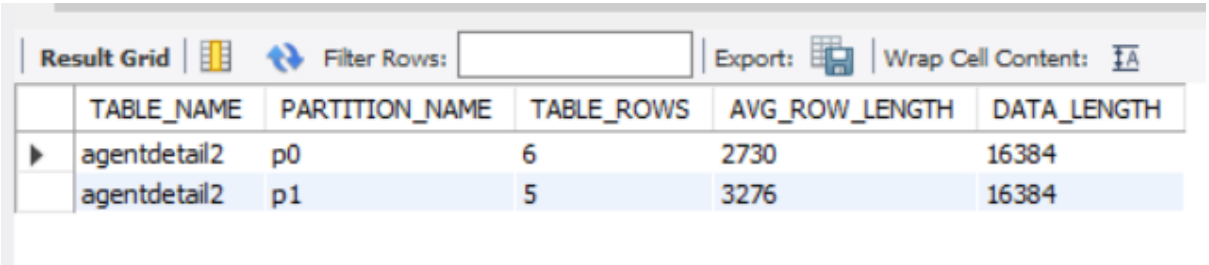
using the PARTITION BY KEY clause. Here, we will use KEY rather than HASH that can accept only a list of zero or more column names.

If the table contains a PRIMARY KEY and we have not specified any column for partition, then the primary key is used as partitioning key.

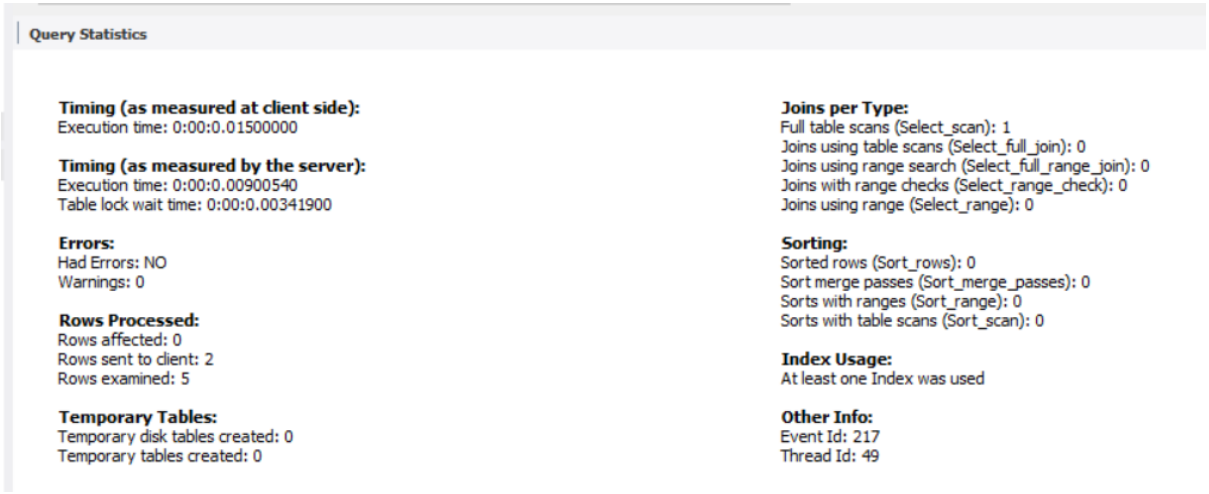
CODE:

```
CREATE TABLE AgentDetail2 (
    agent_id INT NOT NULL PRIMARY KEY,
    agent_name VARCHAR(40)
)
PARTITION BY KEY()
PARTITIONS 2;

INSERT INTO AgentDetail2 VALUES
(1, "Name"),
(2, "Name"),
(3, "Name"),
(4, "Name"),
(5, "Name"),
(6, "Name"),
(7, "Name"),
(8, "Name"),
(9, "Name"),
(10, "Name"),
(11, "Name");

SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH,
DATA_LENGTH
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'AgentDetail2';
```

OUTPUT:



TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
agentdetail2	p0	6	2730	16384
agentdetail2	p1	5	3276	16384



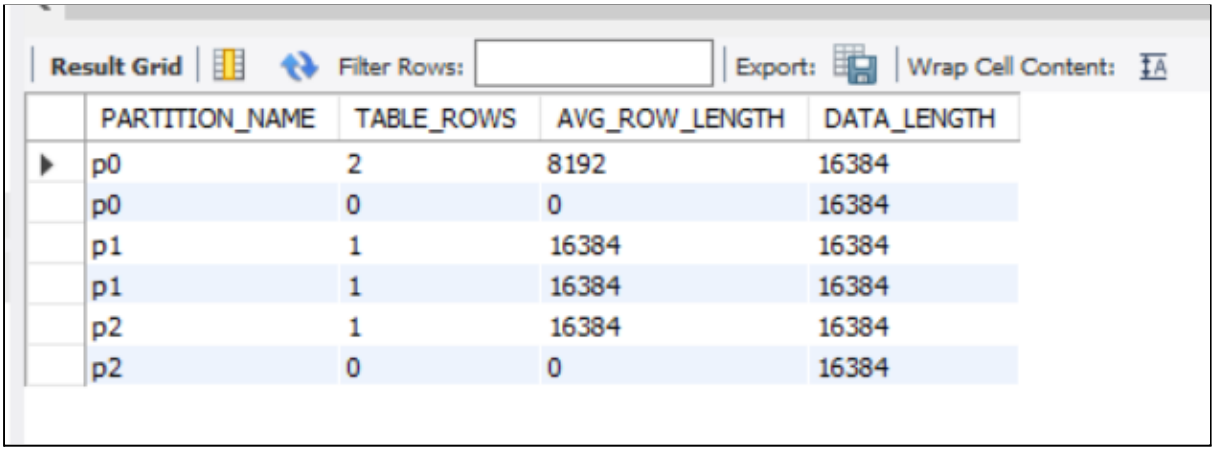
SUBPARTITIONING

It is a composite partitioning that further splits each partition in a partition table.

CODE:

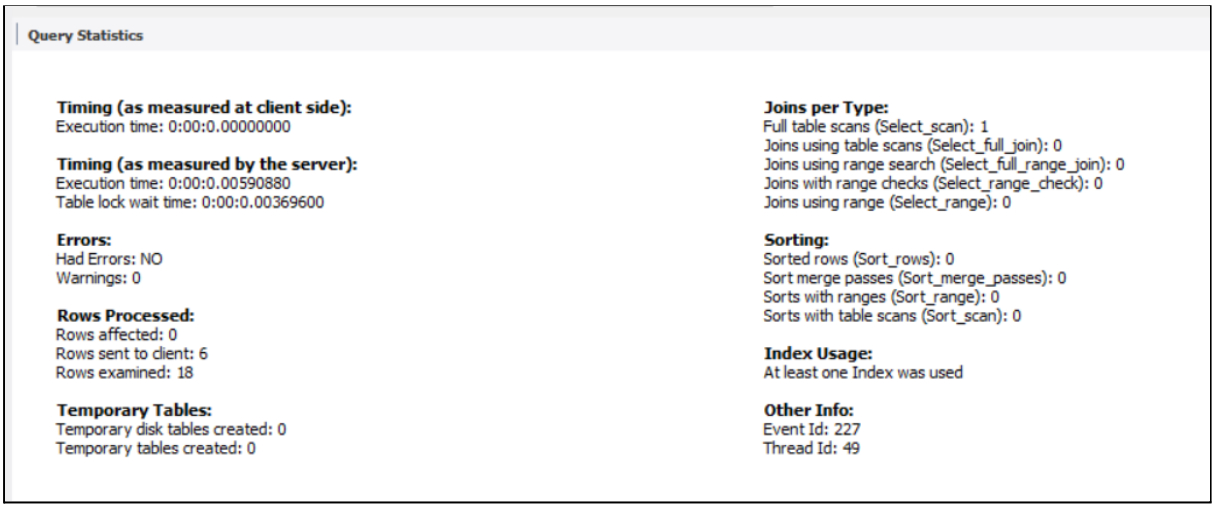
```
CREATE TABLE Person (
    id INT NOT NULL,
    name VARCHAR(40),
    purchased DATE,
    PRIMARY KEY (`id`, `purchased`)
)
 PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (2015),
        PARTITION p1 VALUES LESS THAN (2020),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );

INSERT INTO Person VALUES
(1, "Name", "2013-01-13"),
(2, "Name2", "2014-04-22"),
(3, "Name3", "2015-02-25"),
(4, "Name4", "2018-05-05"),
(5, "Name5", "2020-02-04");

SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'partitioning' AND TABLE_NAME = 'Person';
```

OUTPUT:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
p0	2	8192	16384
p0	0	0	16384
p1	1	16384	16384
p1	1	16384	16384
p2	1	16384	16384
p2	0	0	16384



Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00590880
Table lock wait time: 0:00:0.00369600

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 6
Rows examined: 18

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 227
Thread Id: 49

Conclusion: We learnt about partitioning and the different types of partitions available in MySQL. We then learnt about its uses and applications in real life where the main of goal of partitioning is to aid in maintenance of large tables and to reduce the overall response time to read and load data for particular SQL operations.

Experiment No: 05

Aim: Implementation of Replication transparency in DDB.

Labwork:

Theory: Replication is actually quite straightforward. At its core, it merely involves an administrator picking a server to act as the master, and then registering one or more slave servers to receive updates from the master. Each slave server is responsible for contacting the master server. This master server records all data manipulation statements in a binary log, which is then fed in a stream to any slave(s) that contact the master. The slave computers then play back these statements locally, thus updating their own data copies accordingly. In addition, a slave can, in turn, act as a master to other servers. This lets you construct sophisticated chains of replication servers.
Obviously, there are many steps to follow to correctly configure and use replication, but the preceding discussion describes it accurately at a high level.

BENEFITS OF REPLICATION:
Replication is recommended if any of the following are met:
1. high availability--the data stored on your MySQL server needs to be accessible 24 x 7
2. Frequent backups--to protect against data loss, you often back up your databases.
3. Mixed processing profiles--your MySQL database server must field requests from online transaction process (OLTP) and decision support system (DSS) users.
4. Abundant, low-performance computers--your organization might not have the fastest computers, but they have lots of them.
5. Widely dispersed users--your MySQL users are spread among multiple locations.
6. Modular application code--your MySQL-based applications can be easily altered to read data from the slave servers while writing data to the master.

Implementation:

Replication

Slave configuration - Change or reconfigure master server

Make sure, you have unique server-id in your configuration file (my.cnf). If not, please add the following line into [mysqld] section:

```
server-id=1456800786
```

User name:	user1
Password:	•••••
Host:	localhost
Port:	3306

You have added a new user.

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY '****';GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user1'@'localhost' IDENTIFIED BY '****';
```

Edit Privileges: User *'user1'@'localhost'*

Global privileges Check All

Note: MySQL privilege names are expressed in English.

Data

- SELECT
- INSERT
- UPDATE
- DELETE
- FILE

Structure

- CREATE
- ALTER
- INDEX
- DROP
- CREATE TEMPORARY TABLES
- SHOW VIEW
- CREATE ROUTINE
- ALTER ROUTINE
- EXECUTE
- CREATE VIEW
- EVENT
- TRIGGER

Administration

- GRANT
- SUPER
- PROCESS
- RELOAD
- SHUTDOWN
- SHOW DATABASES
- LOCK TABLES
- REFERENCES
- ☑ REPLICATION CLIENT
- ☑ REPLICATION SLAVE
- CREATE USER

Resource limits

Note: Setting these options to 0 (zero) removes the limit.

MAX QUERIES PER HOUR	0
MAX UPDATES PER HOUR	0
MAX CONNECTIONS PER HOUR	0
MAX USER_CONNECTIONS	0

Replication

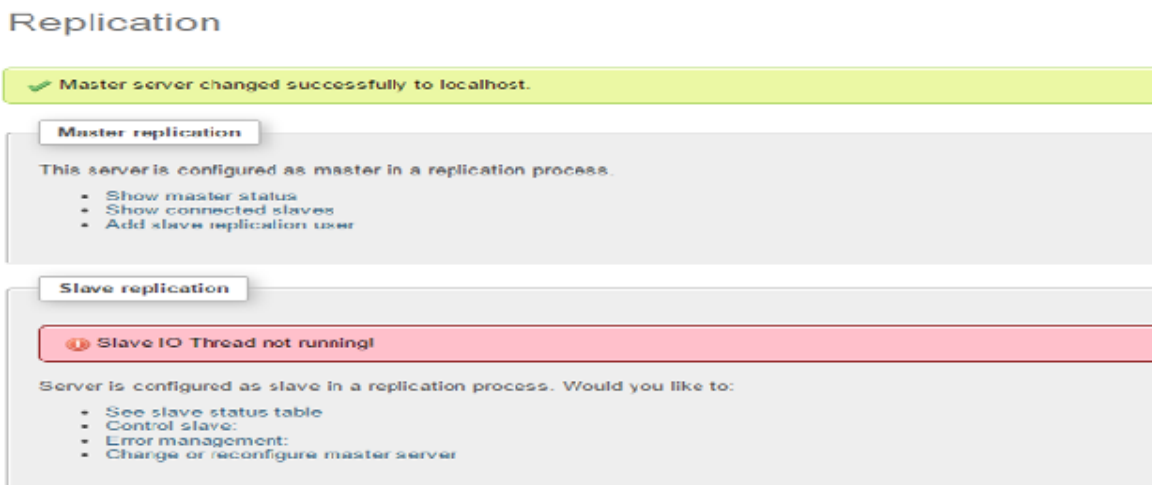
Master replication

This server is configured as master in a replication process.

- Show master status
- Show connected slaves
- Add slave replication user

Slave replication

This server is not configured as slave in a replication process. Would you like to configure it?



Conclusion: Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.In case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure which is why it has many real life applications.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Junaid Girkar

60004190057

TE Comps A4

Experiment No 6

Aim: Implementation of 2 phase commit protocol in a distributed system.

Theory:

The two-phase commit protocol breaks a database commit into two phases to ensure correctness and fault tolerance in a distributed database system.

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site. When the controlling site has received "DONE" message from all slaves, it sends a "Prepare" message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a "Ready" message.
- A slave that does not want to commit sends a "Not Ready" message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received "Ready" message from all the slaves –
    - The controlling site sends a "Global Commit" message to the slaves.
    - The slaves apply the transaction and send a "Commit ACK" message to the controlling site.

- When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.

- After the controlling site has received the first “Not Ready” message from any slave –
  - The controlling site sends a “Global Abort” message to the slaves.
  - The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  - When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Code:

1. **Server**

```
package com.company;
import java.io.*;
import java.net.*;

public class TPCServer {
   public static void main(String a[])throws Exception
   {
       BufferedReader br;
       InetAddress lclhost;
       lclhost=InetAddress.getLocalHost();
       Server ser=new Server(lclhost);

       System.out.println("TPCServer in sending mode.....");
```

```
        // Sending data to client 1
        ser.setSendPort(9000);   //recport=8000
        ser.setRecPort(8001);   //sendport=9001
        System.out.println("Send request data to client1..");
        br=new BufferedReader(new InputStreamReader(System.in));
        String s=br.readLine();
        System.out.println("Data is "+s);
        ser.sendData();
        System.out.println("Waiting for response from client1....");
        ser.recData();

        // Sending data to client 2
        ser.setSendPort(9002);  //recport=8002
        ser.setRecPort(8003);  //senport=9003
        System.out.println("Send request data to client2..");
        br=new BufferedReader(new InputStreamReader(System.in));
        String s1=br.readLine();
        System.out.println("Data is "+s1);
        ser.sendData();
        System.out.println("Waiting for response from client2....");
        ser.recData();

        //Sending the final result to client 1
        ser.setSendPort(9000);
        ser.sendData();
        //Sending the final result to client 2
        ser.setSendPort(9002);
        ser.sendData();
    }

}

class Server
{
    InetAddress lclhost;
    int sendPort,recPort;
    int ssend =0;
    int scounter=0;
    Server(InetAddress lclhost)
```

```
    {
        this.lclhost=lclhost;
    }
    public void setSendPort(int sendPort)
    {
        this.sendPort=sendPort;
    }

    public void setRecPort(int recPort)
    {
        this.recPort=recPort;
    }
    public void sendData()throws Exception
    {

        DatagramSocket ds;
        DatagramPacket dp;
        String data="";
        if(scounter<2 && ssend<2)
        {
            data="VOTE_REQUEST";
        }
        if(scounter<2 && ssend>1)
        {
            data="GLOBAL_ABORT";
            data= data + " TRANSACTION ABORTED";

        }
        if(scounter==2 && ssend>1)
        {
            data="GLOBAL_COMMIT";
            data= data + " TRANSACTION COMMITED";

        }



        ds=new DatagramSocket(sendPort);
        dp=new
DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
```

```
        ds.send(dp);
        ds.close();
        ssend++;
    }
    public void recData()throws Exception
    {
        byte[] buf =new byte[256];
        DatagramPacket dp=null;
        DatagramSocket ds=null;
        String msgStr="";
        try{
            ds=new DatagramSocket(recPort);
            dp=new DatagramPacket(buf,buf.length);
            ds.receive(dp);
            ds.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        msgStr=new String(dp.getData(),0,dp.getLength());
        System.out.println("String = "+msgStr);
        if(msgStr.equalsIgnoreCase("VOTE_COMMIT"))
        {
            scounter++;
        }
        System.out.println("Counter value = "+scounter + "n Send value =
"+ssend);
    }
};
```

2. Client 1

```
package com.company;

import java.net.*;
import java.io.*;
```

```
public class Client1 {
   public static void main(String[] a)throws Exception
   {
       InetAddress lclhost;
       lclhost=InetAddress.getLocalHost();
       Client clnt=new Client(lclhost);

       clnt.setSendPort(9001);   //recport=8000
       clnt.setRecPort(8000);   //sendport=9001
       clnt.recData();
       clnt.sendData();
       clnt.recData();
   }
}

class Client
{
   InetAddress lclhost;
   int sendPort,recPort;
   Client(InetAddress lclhost)
   {
       this.lclhost=lclhost;
   }
   public void setSendPort(int sendPort)
   {
       this.sendPort=sendPort;
   }

   public void setRecPort(int recPort)
   {
       this.recPort=recPort;
   }
   public void sendData()throws Exception
   {
       BufferedReader br;
       DatagramSocket ds;
       DatagramPacket dp;
       String data="";
       System.out.println("Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT'
");
```

```
        br=new BufferedReader(new InputStreamReader(System.in));
        data = br.readLine();
        System.out.println("Data is "+data);

        ds=new DatagramSocket(sendPort);
        dp=new
DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
        ds.send(dp);
        ds.close();

    }
    public void recData()throws Exception
    {
        byte[] buf =new byte[256];
        DatagramPacket dp;
        DatagramSocket ds;

        ds=new DatagramSocket(recPort);
        dp=new DatagramPacket(buf,buf.length);
        ds.receive(dp);
        ds.close();
        String msgStr=new String(dp.getData(),0,dp.getLength());
        System.out.println("Client1 data " +msgStr);

    }
};
```

3. Client 2

```
package com.company;
import java.io.*;
import java.net.*;

public class Client2 {
    public static void main(String[] a)throws Exception
    {
        InetAddress lclhost;
        lclhost=InetAddress.getLocalHost();
```

```
        Clientt2 client=new Clientt2(lclhost);
        // Sending data to client 2
        client.setSendPort(9003);  //recport=8002
        client.setRecPort(8002);  //senport=9003
        client.recData();
        client.sendData();
        client.recData();

    }
}

class Clientt2
{
    InetAddress lclhost;
    int sendPort,recPort;
    Clientt2(InetAddress lclhost)
    {
        this.lclhost=lclhost;
    }
    public void setSendPort(int sendPort)
    {
        this.sendPort=sendPort;
    }

    public void setRecPort(int recPort)
    {
        this.recPort=recPort;
    }
    public void sendData()throws Exception
    {
        BufferedReader br;
        DatagramSocket ds;
        DatagramPacket dp;
        String data="";
        System.out.println("Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT'
");
        br=new BufferedReader(new InputStreamReader(System.in));

        data = br.readLine();
        System.out.println("Data is "+data);
```

```
        ds=new DatagramSocket(sendPort);
        dp=new
DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
        ds.send(dp);
        ds.close();

    }
    public void recData()throws Exception
    {
        byte[] buf =new byte[256];
        DatagramPacket dp;
        DatagramSocket ds;
        ds=new DatagramSocket(recPort);
        dp=new DatagramPacket(buf,buf.length);
        ds.receive(dp);
        ds.close();
        String msgStr=new String(dp.getData(),0,dp.getLength());
        System.out.println(msgStr);
    }
};
```

Output:

```
C:\Users\Junaid\Documents\sem5\ADBMS>java TCPServer
TCPServer in sending modeâ??..
Send request data to client1..
VOTE COMMIT
Data is VOTE COMMIT
Waiting for response from client1â??.
String = VOTE_ABORT
Counter value = 0n Send value = 1
Send request data to client2..
VOTE COMMIT
Data is VOTE COMMIT
Waiting for response from client2â??.
String = VOTE_COMMIT
Counter value = 1n Send value = 2
```

```
C:\Users\Junaid\Documents\sem5\ADBMS>java Client1
Client1 data VOTE_REQUEST
Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT'
VOTE_ABORT
Data is VOTE_ABORT
Client1 data GLOBAL_ABORT TRANSACTION ABORTED
```

```
C:\Users\Junaid\Documents\sem5\ADBMS>java Client2
VOTE_REQUEST
Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT'
VOTE_COMMIT
Data is VOTE_COMMIT
GLOBAL_ABORT TRANSACTION ABORTED
```

Conclusion:

Here we implement the 2 phase commit protocol in a distributed system.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Junaid Girkar
60004190057
TE Comps A4

Experiment No. 7

Aim: Query execution on XML Database.

Theory:

An **XML database** is a data persistence software system that allows data to be specified and sometimes stored, in XML format. This data can be queried, transformed, exported, and returned to a calling system. XML databases are a flavor of document-oriented databases which are in turn a category of NoSQL databases.

Types of XML databases

There are two types of XML databases.

1. XML-enabled database
2. Native XML database (NXD)

XML-enable Database:

An XML-enabled database functions in the same way as a relational database. It's similar to an extension that allows you to convert XML documents. Data is kept in this database in the form of rows and columns in a table.

Native XML Database:

Large volumes of data are contained in native XML databases. Native XML databases use container format instead of table format. XPath expressions can be used to query data.

Because it is incredibly competent in storing, maintaining, and querying XML content, native XML databases are favored over XML-enabled databases.

Implementation:

1. **Creating an XML Table:**

```
postgres=# create table users as select xml $$<users>
postgres$#     <user id="NoTechnician1">
postgres$#         <username>Star</username>
postgres$#         <email>star@gmail.com</email>
postgres$#         <duration>1y</duration>
postgres$#         <karma>60</karma>
postgres$#         <followers>127</followers>
postgres$#         <posts>20</posts>
postgres$#         <comments>9</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician2">
postgres$#         <username>Daisy</username>
postgres$#         <email>daisy@gmail.com</email>
postgres$#         <duration>2m</duration>
postgres$#         <karma>2</karma>
postgres$#         <followers>7</followers>
postgres$#         <posts>5</posts>
postgres$#     </user>
postgres$#     <user id="NoTechnician3">
postgres$#         <username>Lora</username>
postgres$#         <email>lora@gmail.com</email>
postgres$#         <duration>5y2m</duration>
postgres$#         <karma>700</karma>
postgres$#         <followers>555</followers>
postgres$#         <posts>70</posts>
postgres$#         <comments>89</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician4">
postgres$#         <username>Sam</username>
postgres$#         <email>sam@gmail.com</email>
postgres$#         <duration>6m</duration>
postgres$#         <karma>100</karma>
postgres$#         <followers>27</followers>
postgres$#         <posts>60</posts>
postgres$#     </user>
```

```
postgres$#     <user id="NoTechnician5">
postgres$#         <username>Roma</username>
postgres$#         <email>roma@gmail.com</email>
postgres$#         <duration>8y</duration>
postgres$#         <karma>500</karma>
postgres$#         <followers>327</followers>
postgres$#         <posts>80</posts>
postgres$#         <comments>90</comments>
postgres$#     </user>
postgres$#     <user id="NoTechnician6">
postgres$#         <username>Mike</username>
postgres$#         <email>mike@gmail.com</email>
postgres$#         <duration>1m</duration>
postgres$#         <karma>5</karma>
postgres$#         <followers>21</followers>
postgres$#         <posts>4</posts>
postgres$#     </user>
postgres$# </users>$$;
SELECT 1
```

2. Selecting from Users Table:

```
postgres=# select * from users;
                  xml
-----------------------------------------
 <users>                                 +
     <user id="NoTechnician1">           +
         <username>Star</username>       +
         <email>star@gmail.com</email>   +
         <duration>1y</duration>         +
         <karma>60</karma>               +
         <followers>127</followers>      +
         <posts>20</posts>               +
         <comments>9</comments>          +
     </user>                             +
     <user id="NoTechnician2">           +
         <username>Daisy</username>      +
         <email>daisy@gmail.com</email>+
         <duration>2m</duration>         +
         <karma>2</karma>                +
         <followers>7</followers>        +
         <posts>5</posts>                +
     </user>                             +
     <user id="NoTechnician3">           +
         <username>Lora</username>       +
         <email>lora@gmail.com</email>   +
         <duration>5y2m</duration>       +
         <karma>700</karma>              +
         <followers>555</followers>      +
         <posts>70</posts>               +
         <comments>89</comments>         +
     </user>                             +
     <user id="NoTechnician4">           +
         <username>Sam</username>        +
         <email>sam@gmail.com</email>    +
         <duration>6m</duration>         +
         <karma>100</karma>              +
         <followers>27</followers>       +
         <posts>60</posts>               +
     </user>                             +
```

```
     <user id="NoTechnician5">          +
         <username>Roma</username>      +
         <email>roma@gmail.com</email>  +
         <duration>8y</duration>        +
         <karma>500</karma>             +
         <followers>327</followers>     +
         <posts>80</posts>              +
         <comments>90</comments>        +
     </user>                            +
     <user id="NoTechnician6">          +
         <username>Mike</username>      +
         <email>mike@gmail.com</email>  +
         <duration>1m</duration>        +
         <karma>5</karma>               +
         <followers>21</followers>      +
         <posts>4</posts>               +
     </user>                            +
 </users>
(1 row)
```

3. Using xpath to extract data from XML:

```
postgres=# With tbl2(p_xml) as (SELECT
postgres(# '<users>
postgres'#     <user id="NoTechnician1">
postgres'#         <username>Star</username>
postgres'#         <email>star@gmail.com</email>
postgres'#         <duration>1y</duration>
postgres'#         <karma>60</karma>
postgres'#         <followers>127</followers>
postgres'#         <posts>20</posts>
postgres'#         <comments>9</comments>
postgres'#     </user>
postgres'#     <user id="NoTechnician6">
postgres'#         <username>Mike</username>
postgres'#         <email>mike@gmail.com</email>
postgres'#         <duration>1m</duration>
postgres'#         <karma>5</karma>
postgres'#         <followers>21</followers>
postgres'#         <posts>4</posts>
postgres'#     </user>
postgres'# </users>'::xml)
postgres-# select xpath('/users/user/username/text()',p_xml) from tbl2;
    xpath
------------
 {Star,Mike}
(1 row)
```

Conclusion:

We implemented queries in PostgreSQL database in XML format and conclude that XML Database implementation is very simple and user-friendly.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Junaid Girkar
60004190057
TE COMPS A4

Experiment No. 8

Aim: Data handling using JSON.

Theory:

Documents

A document is a record in a document database. A document typically stores information about one object and any of its related metadata. Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML.

Collections

A collection is a group of documents. Collections typically store documents that have similar contents. Not all documents in a collection are required to have the same fields, because document databases have a flexible schema.

Document Database: A document database is a NoSQL data store that is designed to store and query data as JSON-like documents. The data in document databases is stored as documents with their metadata. The document stored is in a key/value pair where the key is the unique identifier of the document. Unlike relational databases, document databases are faster to load, access, and parse.

Document databases are also referred to as document database management systems, document-oriented databases, or document store databases.

Here are the key characteristics of document databases:

1. Document DBMSs are NoSQL databases.
2. Document DBMSs use key/value to store and access documents data.
3. Document DBMSs have a flexible schema that can be different for each document. For example, one document can be an Author profile, while another document can be a blog.

4. Common examples of document DBMS include JSON, XML docs, Catalogs, serialized PDFs and Excel docs, Profile data, and serialized objects.

Traditional relational DBMSs are not designed to provide efficient access to large documents or unstructured data. In case of catalogs, or profiles, or document storages, we don't need structured design. For example, storing a document in a CMS does not require a structured format.

Document databases are designed to store large documents in a key/value store that are easy to search and access. The entire document is read into a memory object that is easy to read and present. User profiles, content management systems, and catalogs are some common use cases of document DBMS. One of the perfect examples of use of a document database is storing C#
Corner articles in a document DB, rather than a DBMS.

MongoDB

MongoDB is one of the most popular document databases. MongoDB is a free, distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in your application code, making data easy to work with.

Implementation:

1. Importing a JSON file into Mongo

Data:

```
{
    "quiz": {
        "sport": {
            "q1": {
                "question": "Which one is correct team name in NBA?",
                "options": [
                    "New York Bulls",
                    "Los Angeles Kings",
                    "Golden State Warriros",
                    "Huston Rocket"
                ],
```

```
                "answer": "Huston Rocket"
            }
        },
        "maths": {
            "q1": {
                "question": "5 + 7 = ?",
                "options": [
                    "10",
                    "11",
                    "12",
                    "13"
                ],
                "answer": "12"
            },
            "q2": {
                "question": "12 - 8 = ?",
                "options": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "answer": "4"
            }
        }
    }
}
```


Code:

```
C: \Junaid\SEM5\ADBMS\Experiment 8>mongoimport --uri
mongodb://localhost:27017/ADBMS --collection EXP8 --type json --file
file.json
```

Output:

```
2022-01-28T12:52:29.15.0530   connected to: mongodb://localhost:27017/ADBMS
2022-01-28T12:52:29.174.0530  1 document(s) imported successfully. 0
document(s) failed to import.
```

2. Printing the imported data in MongoDB shell:

Code:

```
>
> use ADBMS
switched to db ADBMS
> db.EXP8.find()
```

Output:

```
{ "_id" : ObjectId("61f399b5082f327f97e133d8"), "quiz" : { "sport" : { "q1"
: { "question" : "Which one is correct team name in NBA?", "options" : [
"New York Bulls", "Los Angeles Kings", "Golden State Warriros", "Huston
Rocket" ], "answer" : "Huston Rocket" } }, "maths" : { "q1" : { "question"
: "5 + 7 = ?", "options" : [ "10", "11", "12", "13" ], "answer" : "12" },
"q2" : { "question" : "12 - 8 = ?", "options" : [ "1", "2", "3", "4" ],
"answer" : "4" } } } }
>
```

Conclusion: Hence, in this way we can import any JSON file into MongoDB and perform operations on it. This is how we handle JSON data by implementing it in document Databases.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Junaid Girkar
60004190057
TE COMPS A4

Experiment No 9

Aim: A case study on Database security issues and measures taken to handle those issues.

Theory:

The breach could be caused by a variety of software vulnerabilities, misconfigurations, or habits of misuse or carelessness. Here are some of the most well-known causes and forms of cyber threats to database security.

Insider Threats

An insider threat is a security risk posed by one of the three sources listed below, each of which has privileged access to the database:

- A nefarious insider with nefarious intentions
- An irresponsible employee who exposes the database to attack as a result of his or her conduct.
- Any unauthorized person who obtains credentials by social engineering or other means, or gains access to the database's credentials.

One of the most common causes of database security breaches is an insider threat, which generally happens when a large number of employees have been given privileged user access.

Human Error

Nearly 50% of all data breaches are still caused by weak passwords, password sharing, data erasure or damage by mistake, and other undesired user behaviors.

The exploitation of Database Software Vulnerabilities

Attackers constantly attempt to isolate and target vulnerabilities in software, and database management software is a highly valuable

target. New vulnerabilities are discovered daily, and all open source database management platforms and commercial database software vendors issue security patches regularly. However, if you don't use these patches quickly, your database might be exposed to attack.

Even if you do apply patches on time, there is always the risk of zero-day attacks, when attackers discover a vulnerability, but it has not yet been discovered and patched by the database vendor.

SQL/NoSQL Injection Attacks

The use of arbitrary non-SQL and SQL attack strings in database queries is a database-specific danger. These are usually queried that are developed as extensions of web application forms or that are received via HTTP requests. If developers do not follow secure coding practises and the organization does not do regular vulnerability testing, every database system is vulnerable to these attacks.

Buffer Overflow Attacks

A buffer overflow occurs when a process attempts to write more data to a fixed-length block of memory than it is capable of holding. Attackers could utilize the surplus data stored at nearby memory addresses as a jumping-off point for their attacks.

Denial of Service (DoS/DDoS) Attacks

In a denial of service (DoS) assault, the cybercriminal uses a huge number of bogus requests to overwhelm the target service—in this case, the database server. As a result, the server is unable to handle genuine user requests and frequently crashes or becomes unstable. Fake traffic is generated by a large number of computers in a botnet controlled by the attacker in a distributed denial of service (DDoS) assault. Without a sufficiently scalable defensive architecture, this results in extremely high traffic volumes, which are impossible to halt. Cloud-based DDoS prevention services can dynamically scale up to handle massive DDoS attacks.

Malware

Malware is computer programme designed to exploit security flaws or harm a database. Malware could infiltrate the database's network through any endpoint device. Because of their high value and sensitivity, malware security is critical on any endpoint, but more so on database servers.

Distributed Systems

Because there are many users, diverse data, multiple sites, and distributed control, a distributed system requires more security than a centralised system. We'll look at the many aspects of distributed database security in this chapter.

There are two categories of intruders in distributed communication systems:

- They are passive eavesdroppers who monitor messages and obtain private information.
- Active attackers not only monitor messages, but actively tamper with data by adding new data or changing current data.

Security measures include communications security, data security, and data auditing.

Communications Security

In a distributed database, a lot of data communication takes place owing to the diversified location of data, users and transactions. So, it demands secure communication between users and databases and between the different database environments.

Security in communication encompasses the following −

- Data should not be corrupt during transfer.
- The communication channel should be protected against both passive eavesdroppers and active attackers.
- In order to achieve the above stated requirements, well-defined security algorithms and protocols should be adopted.

Two popular, consistent technologies for achieving end-to-end secure communications are −

- Secure Socket Layer Protocol or Transport Layer Security Protocol.
- Virtual Private Networks (VPN).

Data Security

In distributed systems, it is imperative to adopt measures to secure data apart from communications. The data security measures are −

- Authentication and authorization − These are the access control measures adopted to ensure that only authentic users can use the database. To provide authentication digital certificates are used. Besides, login is restricted through username/password combination.
- Data encryption − The two approaches for data encryption in distributed systems are −
    - Internal to distributed database approach: The user applications encrypt the data and then store the encrypted data in the database. For using the stored data, the applications fetch the encrypted data from the database and then decrypt it.
    - External to distributed database: The distributed database system has its own encryption capabilities. The user applications store data and retrieve them without realizing that the data is stored in an encrypted form in the database.
- Validated input − In this security measure, the user application checks for each input before it can be used for updating the database. An unvalidated input can cause a wide range of exploits like buffer overrun, command injection, cross-site scripting and corruption in data.

Data Auditing

In order to determine the security measures that should be implemented, a database security system must detect and monitor security infractions. It's often difficult to discover security breaches right after they happen. Examining audit logs is one way to

find security issues. Information like this can be found in audit logs.

- Date, time, and location of unsuccessful access attempts.
- Attempts to gain access that were successful.
- Changes to the database system that are critical.
- Access to massive amounts of data, particularly from various databases.

All of the foregoing data provides insight into the database's activity. Periodic log analysis aids in identifying any unusual activity, as well as its location and time of occurrence.

Document Databases(NoSQL)

Authentication and encryption are practically non-existent in NoSQL databases, or are very poor when implemented. The following are some of the security concerns with NoSQL databases:

- By default, administrative user or authentication is disabled.
- It features a password storing system that is extremely vulnerable.
- The client sends plaintext messages to the server (MongoDB)
- External encryption techniques such as LDAP, Kerberos, and others are not supported.
- Data files are not encrypted due to a lack of encryption capabilities.
- Weak authentication on both the client and server sides
- SQL injection vulnerability
- Denial-of-service (DoS) attacks are a type of cyber-attack
- At rest, data is unencrypted.
- The available encryption solution isn't ready for use in production.
- For client communication, encryption is not accessible.

With all these security problems, it is best to understand that NoSQL databases are still new technologies and more security enhancements will be added to newer versions. Enterprise package Cassandra tools provided by companies like Datastax do have more security enhancements and hence are more secure and provide companies with all the security needed.

Solution

1. Pseudonyms-based Communication Network:

In the context of this system, users can have access to multiple services by inserting their credentials only once, that is when they are initially connected to the system. Such a system is called anonymous because users can be known only through their pseudonyms, and the transactions demonstrated by the same user cannot be linked, as their identity is disclosed.

For this reason, it is considered the best means in terms of user protection.

Additionally, the Credential Authority CA prevents the sharing of credentials or pseudonyms and guarantees that users who enter the system have a public and secret key that makes them unique to the system. Another entity in the system is the Verifier V, whose role is to certify the validity of the user credentials and to communicate with either the Issuing Authority or the Credential Authority and inform that the user is not the owner of the credential that is presenting.

Users, in terms of a digital credential, transmit the public key and the CAs digital signature derived from a Proof of Knowledge, through which they prove that they know the secret key and the attributes in the digital credential that satisfies the particular attribute property they are revealing.

2. Monitoring, Filtering, Blocking:

Malicious tasks and queries cannot be detected and disabled in NoSQL databases. Advanced scripts can readily evade the Kerberos central authentication mechanism, and in general, monitoring is limited to data processing primarily in the API.

No information about cluster communication, user connection details, or data changing information (even editing or deleting) is recorded in a cloud environment. Because there are no log files, identifying

incidents of data breach or malicious data loss in the cluster is a difficult task.

Big data technologies provide real-time security systems, resulting in high-speed data analysis. As a result, anomaly detection is applied in real time, and security analytics can be recorded and updated on a regular basis.

Mobile Database

1. **Security**

   The data which is left at the fixed location is more secure as compared to mobile data. That is mobile data is less secure. Data is also becoming more volatile and techniques must be able to compensate for its loss. The most important thing needed in this environment is the authorizing access to critical data and proper techniques.

2. **Data distribution and Replication**

   Uneven distribution of data among the mobile units and the base stations take place here. Higher data availability and low cost of remote access is there in data distribution and replication. The problem of Cache management is compounded by the consistency constraints. The most updated data and frequently accessed data is provided by the Caches to the mobile units. It processes their own transactions. There is more efficient access to data and higher security is available.

3. **Replication issues**

   There is an increase in costs for updates and signaling due to an increase in the number of replicas. Mobile hosts can move anywhere and anytime.

4. **Division of labor**

   There is a certain change in the division of labor in query processing because of certain characteristics of the mobile

environment. There are some cases in which the client must function independently of the server.

5. **Recovery and fault tolerance –**

   Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. Faults can be classified in two types: transient and permanent. Without any apparent intervention, a transient fault will eventually disappear but a permanent fault will remain unless it is removed by some external agency.

   The mobile database environment must deal with site, transaction, media, and communication failures. Due to limited battery power there is a site failure at MU. If a voluntary shutdown occurs in MU, then it should not be treated as a failure. Whenever Mu crosses the cells, most frequently there will be transaction failures during handoff. Due to failure of MU, there is a big cause of network partitioning and affection of the routing algorithms. The characterization of mobile computing is done by:

   - Limiting resource availability
   - Frequent disconnection
   - High mobility
   - Low bandwidth

Solution

For mobile operators, the first step in defeating attacks on their networks is to recognize their newfound role as an ISP. This means implementing a layered defense for their network that:

1. Changes security policies and practices to better reflect the new threats.

2. Concentrates, whenever possible, wireless data services into a smaller number of data centers. Many mobile operators in Europe have already taken these types of steps to protect their core networks.

3. Protects end users by implementing technology on their devices and in the network – e.g., anti-virus, firewalls, content scanning – that provides file-level security.

4. Take an architecture approach to implementing security solutions in their network; point solutions are not sufficient.

5. Make client-side anti-virus and firewall software readily available to their subscribers who use data devices (e.g., feature phones with data capabilities, Smartphone, notebook computers)

Object Oriented Database

1. Aggregation:

The aggregation problem occurs when a user can form aggregates of related items, all of which are classified at some level, that deduce classified data [26]. The higher level information (which may be thought to be subject to a higher level of security clearance) may be inferred from a large number of lower level data items. A collection

of information items may be required to be classified at a higher security level than any of the individual items that comprise it.

The aggregation problem occurs when a subject's right to individual pieces of information results in knowledge to which it does not have a right. The aggregation problem prevents the subject from gaining access to information of higher sensitivity by aggregating lower sensitivity data. This is usually addressed by restricting combinations of accesses in certain ways.

2. Inference problem:

The word “inference” means “forming a conclusion from premises”. Users of any database can draw inferences from the information they have obtained from the database and prior additional information (called supplementary knowledge) they have. The inference can lead to information disclosure if the user is able to access information they are not authorized to read. This is the inference problem in database security. Inference problem occurs when a user can deduce (or infer) information from a collection of individual accesses against a database summarized different approaches to handle the inference problem:

(1) place restrictions on the set of allowable queries that can be issued by a user;

(2) add noise to the data; and

(3) augment a database with a logic-based inference engine to modify queries before the database processes them.

3. Polyinstantiation:

This problem arises when users with different security levels attempt to use the same information. The variety of clearances and sensitivities in a secure database system result in conflicts between the objects that can be accessed and modified by the users. Through the use of polyinstantiation, information is located in more than one location, usually with different security levels.

Obviously, the more sensitive information is omitted from the instances with lower security levels. Although polyinstantiation solves the multiparty update conflict problem, it raises a potentially greater problem in the form of ensuring the integrity of the data within the database. Without some method of simultaneously updating all occurrences of the data in the database, the integrity of the information quickly disappears. In essence, the system becomes a collection of several distinct database systems, each with its own data.

Solution

Multilevel Security:

Multilevel secure computers protect objects classified at more than one level and allow sharing between users of different clearance levels. Objects are labeled with their sensitivity levels. Subject arc associated with clearance levels.

A multilevel secure computer arbitrates all access of objects by subjects. The arbitration is carried out by the rcfcrcncc monitor according to a security policy. MLS/DRMSs must deal with large numbers of objects, intcrrclatcd in complex ways which have semantic meaning. This causes several problems.

The first is efficiency. Large numbers of objects can cause a large burden on the access monitor. Secondly, all of these objects must be classified in a complete and consistent way. The third problem is representing and manipulating objects containing data of multiple sensitivity levels. The interrelations of the data and their semantics lead to an inference problem. Inference occurs when information which can be rctricvcd from the database allows other data to be deduced. Inference provides a flow of data which is not arbitrated by the reference monitor.

Conclusion: We studied the different types of security issues in various types of database systems and the solution to these issues.