Experiment 1

ADBMS

Ayush Jain 60004200132 B2

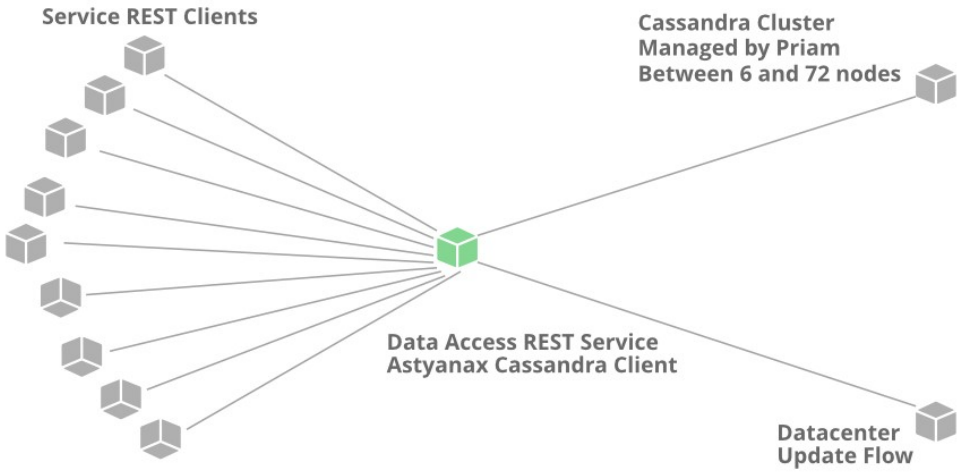
Aim : Case Study on Netflix database

1. **Application:** Netflix
   Netflix, Inc. is an American subscription streaming service and production company based in Los Gatos, California. Founded on August 29, 1997, by Reed Hastings and Marc Randolph in Scotts Valley, California, it offers a film and television series library through distribution deals as well as its own productions, known as Netflix Originals.
   As of September 16, 2022, Netflix had 222 million subscribers worldwide, including 73.3 million in the United States and Canada, 73.0 million in Europe, the Middle East and Africa, 39.6 million in Latin America and 34.8 million in the Asia-Pacific region.It is available worldwide aside from Mainland China, Syria, North Korea, and Russia. Netflix has played a prominent role in independent film distribution, and it is a member of the Motion Picture Association (MPA).

2. **Database used:** MySQL and Cassandra
3. **About MySQL**

   - MySQL databases are used for managing movie titles, billing, and transaction purposes.
   - To be specific, AWS EC2 Deployed MySQL is used to store the data.
   - MySQL is built using the InnoDB engine over large AWS EC2 instances. Data from User Service where we need strong ACID properties, this RDBMS is an obvious choice.
   - Replication on this database is done synchronously, which states that there is a master-master relationship between nodes, and any write operation on the primary node will be considered as done only if that data is synchronised by both local and remote nodes to ensure high availability.

4. **About Cassandra**



- Cassandra is a distributed column-based NoSQL database that is free to use and open source that enables the storage of a large amount of data over servers.
- As we know, Netflix has a large user base globally, so it requires such DB to store user history. It enables handling of large amounts of reading requests efficiently and optimises the latency for large read requests.
- As the user base grew, it became difficult to store so many rows of data, and it was also costly and slow. So Netflix designed a new Database to store the history of users based on the time frame and recent use.

5. **Cassandra with Netflix**

Cassandra is a NoSQL database that can handle large amounts of data and it can also handle heavy writing and reading. When Netflix started acquiring more users, the viewing history data for each member also started increasing. This increases the total number of viewing history data and it becomes challenging for Netflix to handle this massive amount of data. Netflix scaled the storage of viewing history data-keeping two main goals in their mind

Advantages

- Smaller Storage Footprint.
- Consistent Read/Write Performance as viewing per member grows (viewing history data write to read ratio is about 9:1 in Cassandra).

Total Denormalized Data Model

- Over 50 Cassandra Clusters
- Over 500 Nodes
- Over 30TB of daily backups
- The biggest cluster has 72 nodes.

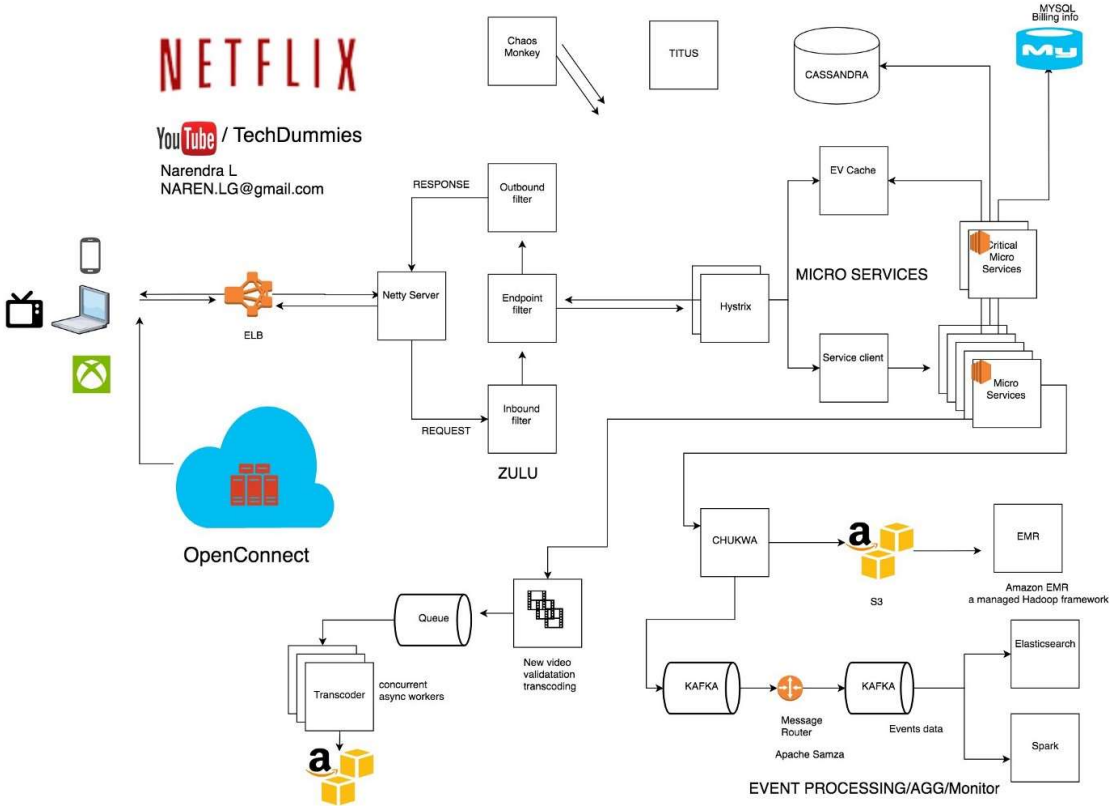
- 1 cluster over 250K writes/s

Initially, the viewing history was stored in Cassandra in a single row. When the users started increasing on Netflix the row sizes as well as the overall data size increased. This resulted in high storage, more operational cost, and slow performance of the application. The solution to this problem was to compress the old rows…

Netflix divided the data into two parts…

- **Live Viewing History (LiveVH):** This section included the small number of recent viewing historical data of users with frequent updates. The data is frequently used for the ETL jobs and stored in uncompressed form.
- **Compressed Viewing History (CompressedVH):** A large amount of older viewing records with rare updates is categorized in this section. The data is stored in a single column per row key, also in compressed form to reduce the storage footprint.

6. Structure



Netflix has 3 main components which we are going to discuss today

- OC or netflix CDN
- backend
- client

Lets first talk about some high level working of Netflix and then jump in to these 3 components.The client is the user interface on any device used to browse and play Netflix videos. TV, XBOX, laptop or mobile phone etc. Anything that doesn't involve serving video is handled in AWS.Everything that happens after you hit play is handled by Open Connect. Open Connect is Netflix's custom global content delivery network (CDN).

Open Connect stores Netflix video in different locations throughout the world. When you press play the video streams from Open Connect, into your device, and is displayed by the client.

CDN — A content delivery network (CDN) is a system of distributed servers (network) that deliver pages and other Web content to a user, based on the geographic locations of the user, the origin of the webpage and the content delivery server.

7. **Advantages/Disadvantage**

Advantages	Disadvantage
Cassandra is Free	NoSQL Will Bother Some Coders
Cassandra Has great Performance	Potential Latency Issues
Continuous Availability from Always-Online Architecture	
Cassandra Has Flexible Language Drivers	

8. **Security**
    - **HTTPS** — encrypting the traffic between client and server over HTTPS. this will ensure that no one in the middle is able to see the data especially passwords
    - **Authentication** — Each API request post-log-in, will do authentication by checking the validity of auth_token in the authorization HTTP header. This ensures that the requests are legitimate.

9. **Opensource**

Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

10. **Reasoning**

- Cassandra has no single point of failure as it automatically replicates to multiple nodes across data centers, making it highly available.
- Netflix launched its streaming service in 2007, using an Oracle database in a single data center. As the company's streaming service users, the devices they binge-watched with, and data expanded rapidly, the limitations on scalability and the potential for failures became a serious threat to Netflix's success.

- Cassandra, with its distributed architecture, was a natural choice, and by 2013, most of Netflix's data was housed there, and Netflix still uses Cassandra today.Cassandra survived its adolescent years by retaining its position as the database that scales more reliably than anything else, with a continual pursuit of operational simplicity at scale.

Conclusion: We successfully did the case study on NETFLIX database

Experiment 2

ADBMS

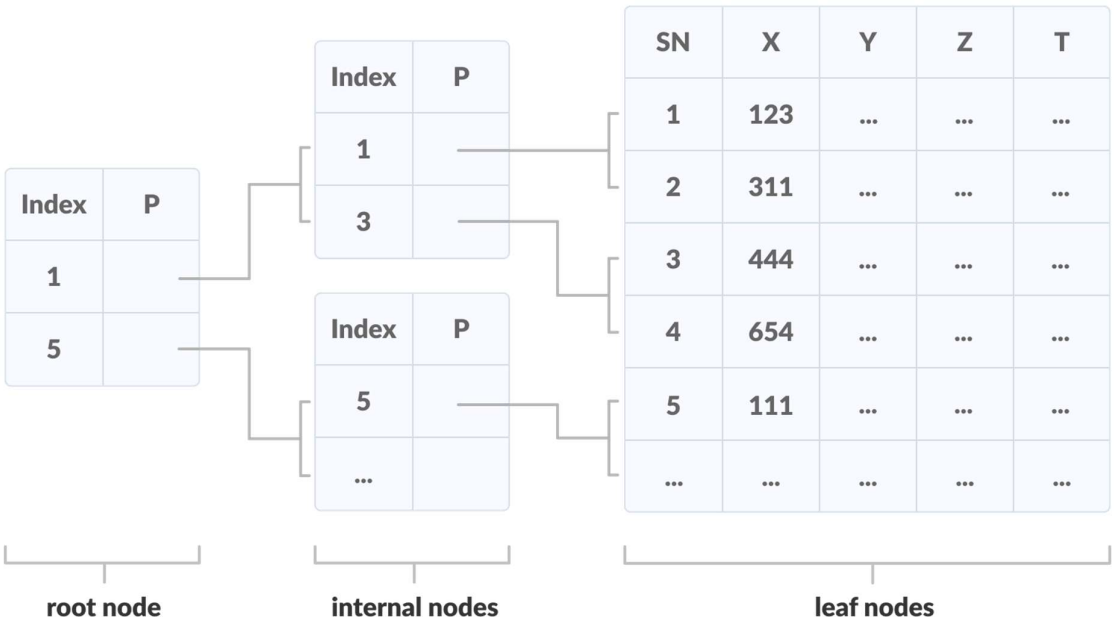
Ayush Jain 60004200132 B2

Aim : Write a program to implement B+ Tree.

Theory:

A B+ tree is an advanced form of a self-balancing tree in which all the values are present in the leaf level.

An important concept to be understood before learning B+ tree is multilevel indexing. In multilevel indexing, the index of indices is created as in figure below. It makes accessing the data easier and faster.

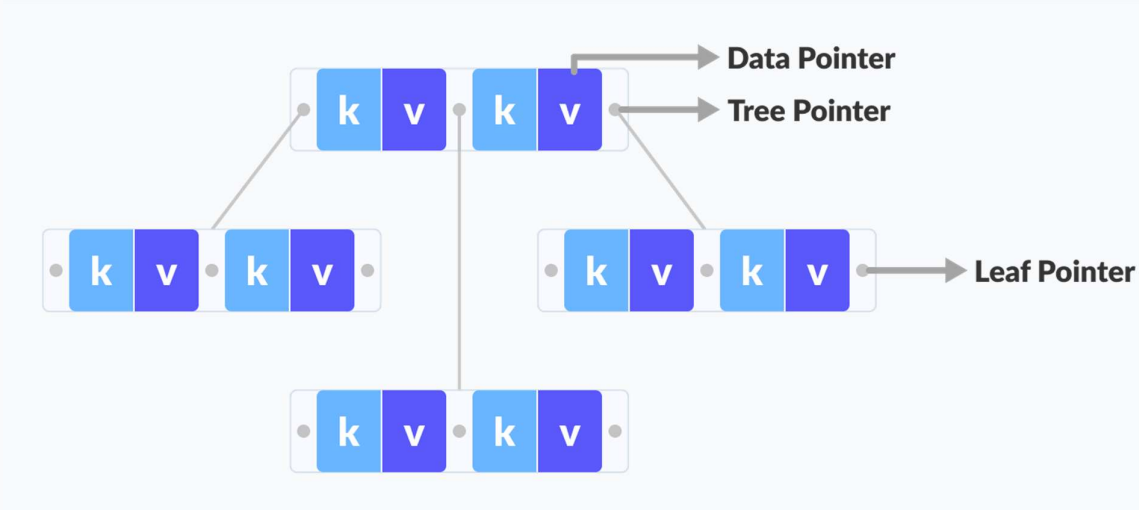


Multilevel Indexing using B+ tree

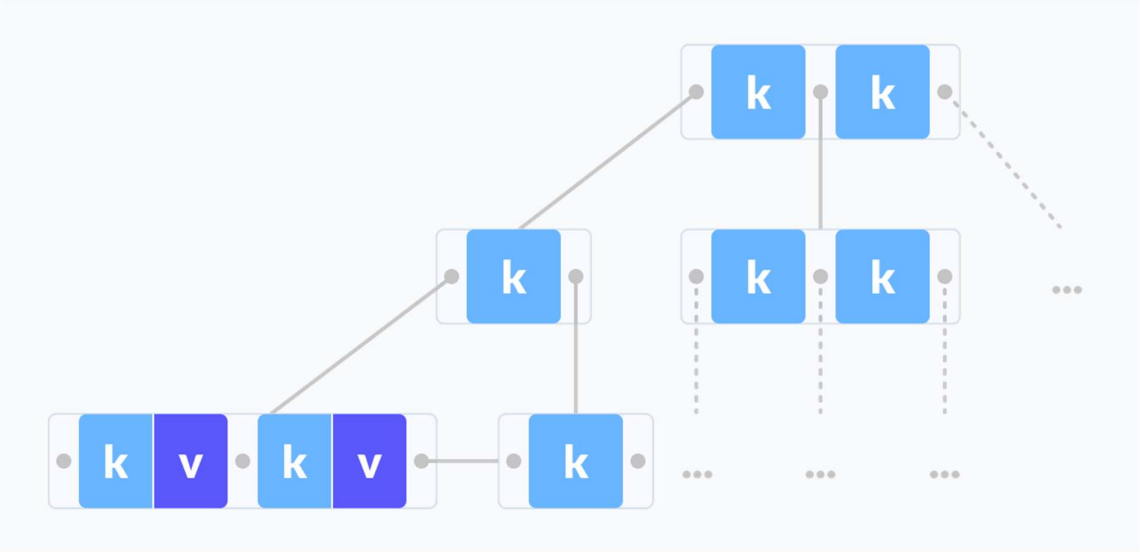
- Properties of a B+ Tree

1. All leaves are at the same level.
2. The root has at least two children.
3. Each node except root can have a maximum of m children and at least $m/2$ children.
4. Each node can contain a maximum of $m - 1$ keys and a minimum of $\lceil m/2 \rceil - 1$ keys.

- Comparison between a B-tree and a B+ Tree



B-tree



B+ tree

The data pointers are present only at the leaf nodes on a B+ tree whereas the data pointers are present in the internal, leaf or root nodes on a B-tree.

The leaves are not connected with each other on a B-tree whereas they are connected on a B+ tree.

Operations on a B+ tree are faster than on a B-tree.

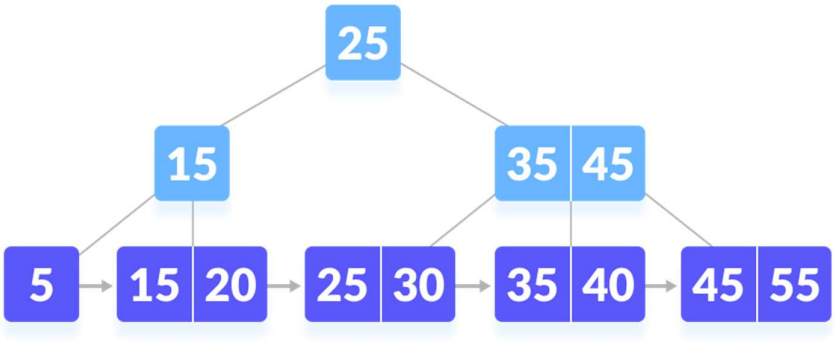
- Searching on a B+ Tree

The following steps are followed to search for data in a B+ Tree of order `m`. Let the data to be searched be `k`.

1. Start from the root node. Compare k with the keys at the root node `[k1, k2, k3,.....km - 1]`.
2. If `k < k1`, go to the left child of the root node.
3. Else if `k == k1`, compare `k2`. If `k < k2`, k lies between `k1` and `k2`. So, search in the left child of `k2`.
4. If `k > k2`, go for `k3, k4,...km-1` as in steps 2 and 3.
5. Repeat the above steps until a leaf node is reached.
6. If k exists in the leaf node, return true else return false.

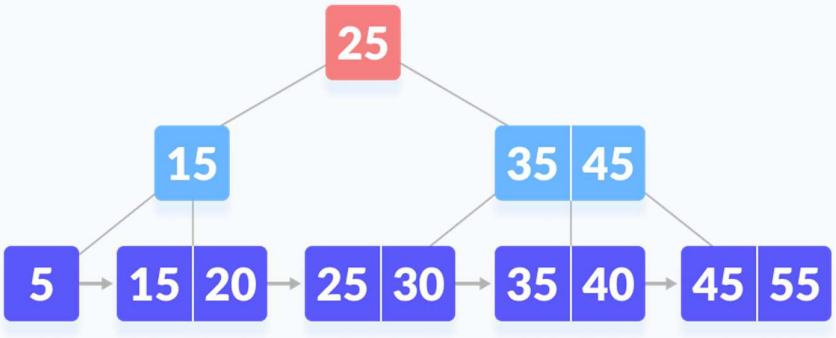
- Searching Example on a B+ Tree

Let us search `k = 45` on the following B+ tree.



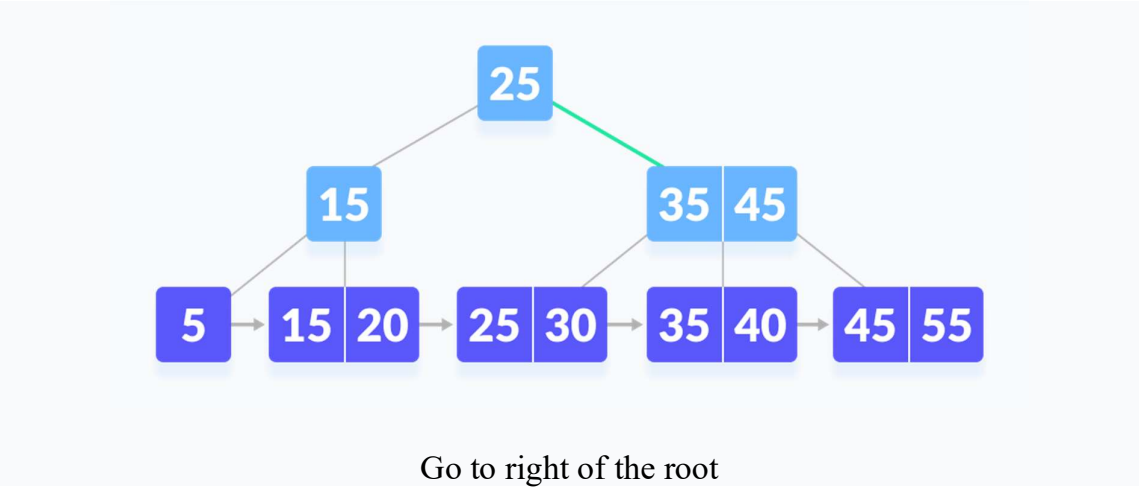
B+ tree

1. Compare k with the root node



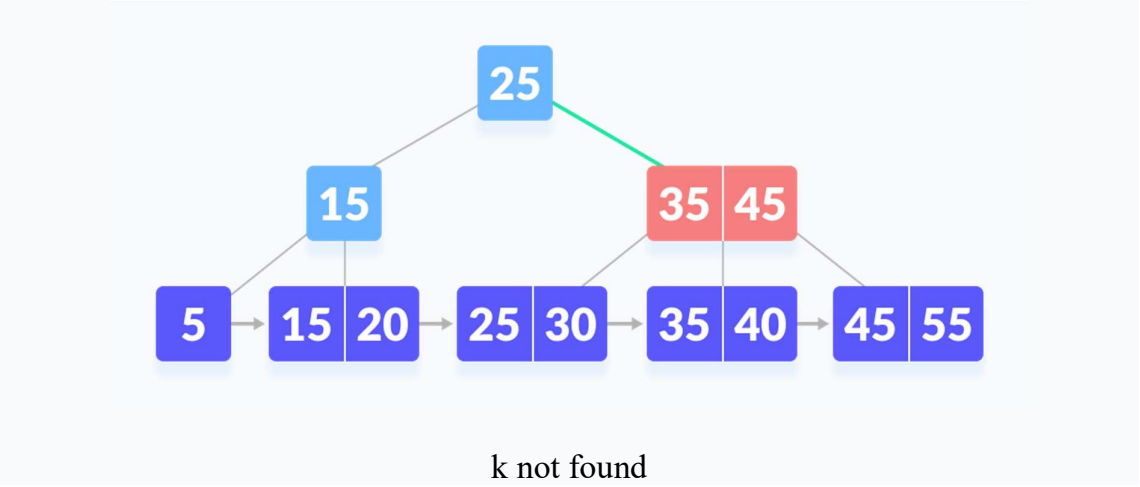
k is not found at the root

2. Since k > 25, go to the right child.



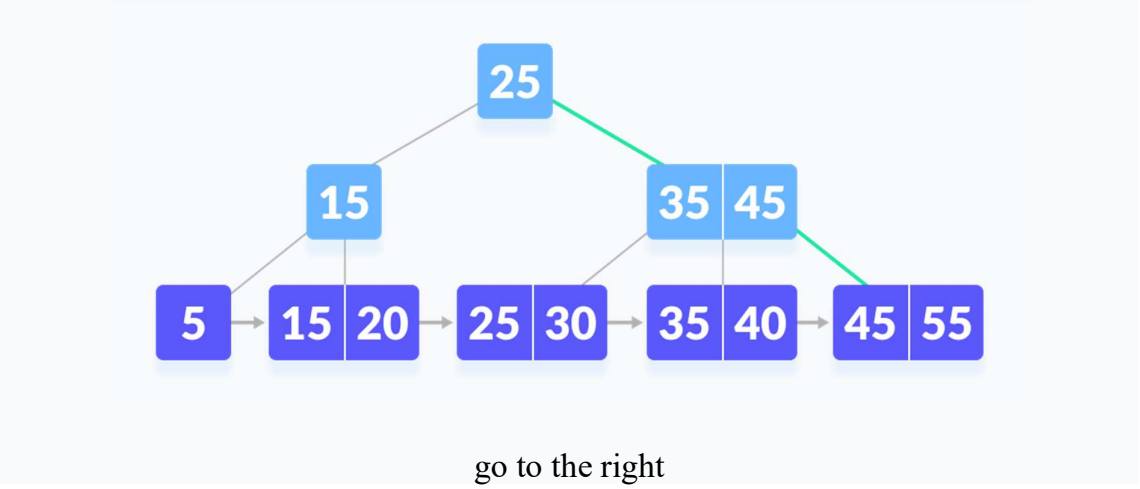
Go to right of the root

3. Compare k with 35. Since k > 30, compare k with 45



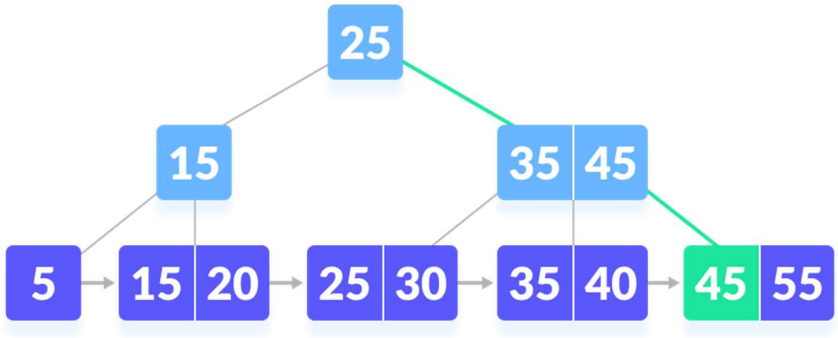
k not found

4. Since k ≥ 45, so go to the right child



go to the right

5. k is found.



k is found

- Insertion Operation

Before inserting an element into a B+ tree, these properties must be kept in mind.

- The root has at least two children.
- Each node except root can have a maximum of m children and at least $m/2$ children.
- Each node can contain a maximum of $m - 1$ keys and a minimum of $\lceil m/2 \rceil - 1$ keys.

The following steps are followed for inserting an element.

1. Since every element is inserted into the leaf node, go to the appropriate leaf node.
2. Insert the key into the leaf node.

Case I

1. If the leaf is not full, insert the key into the leaf node in increasing order.

Case II

1. If the leaf is full, insert the key into the leaf node in increasing order and balance the tree in the following way.
2. Break the node at $m/2th$ position.
3. Add $m/2th$ key to the parent node as well.
4. If the parent node is already full, follow steps 2 to 3.

- Insertion Example

Let us understand the insertion operation with the illustrations below.

The elements to be inserted are 5,15, 25, 35, 45.

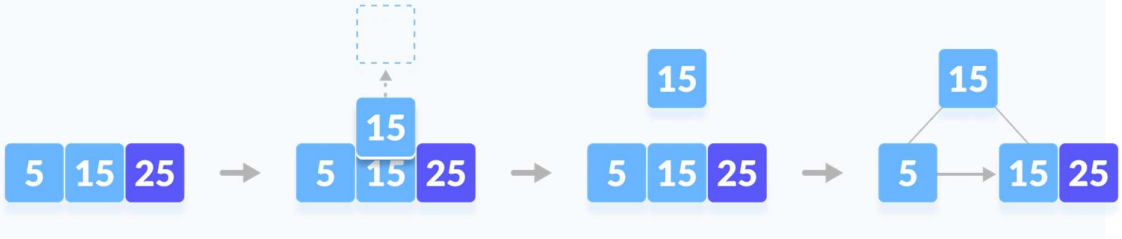
1. Insert 5

Insert 5

2. Insert 15.

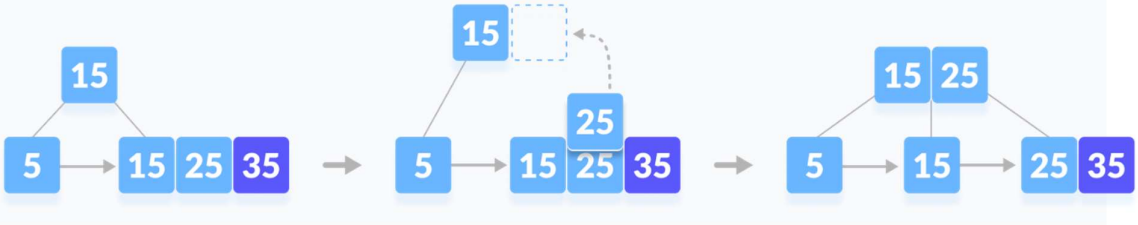
Insert 15

3. Insert 25.



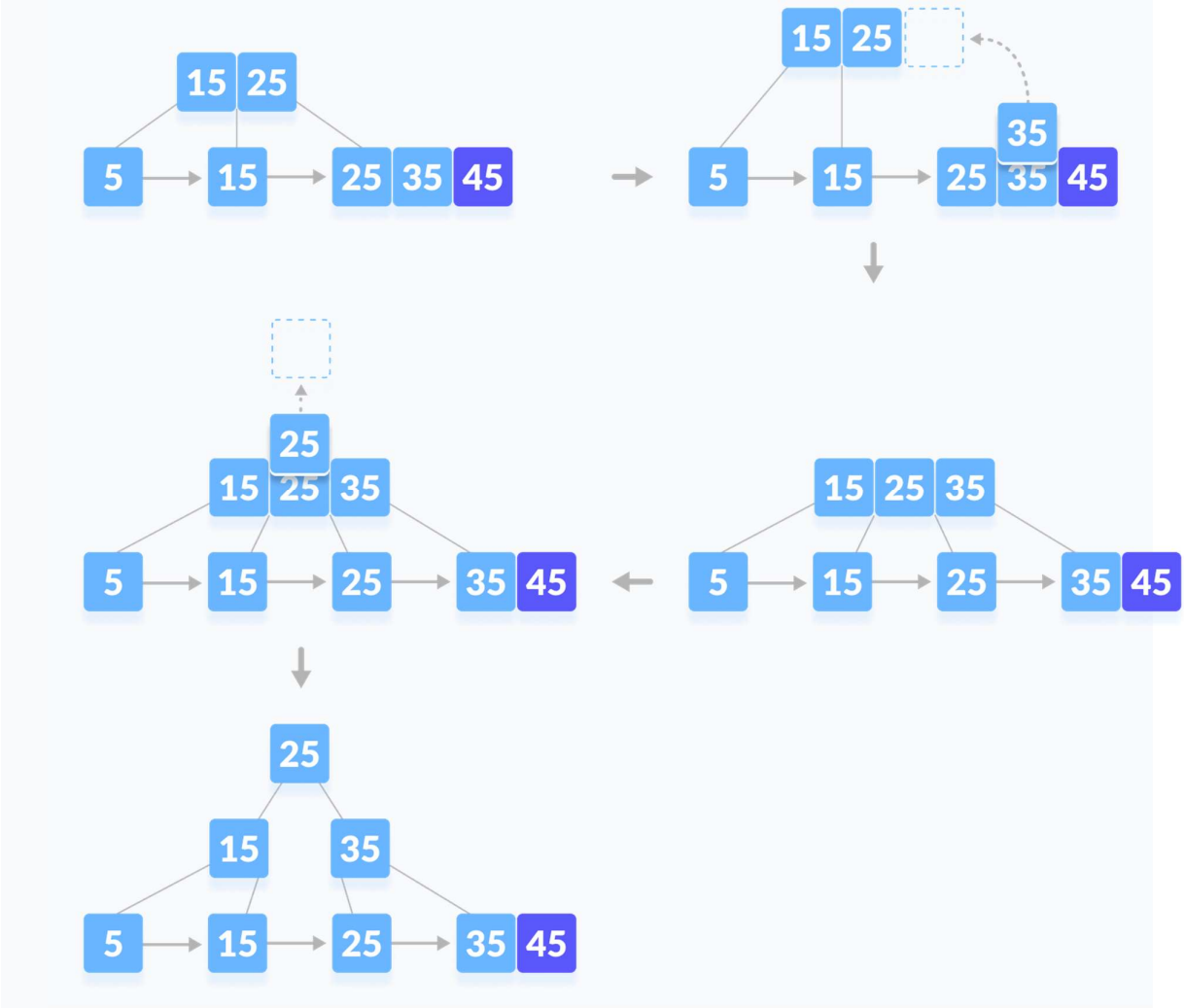
Insert 25

4. Insert 35.



Insert 35

5. Insert 45.



Insert 45

- **Deletion Operation**

Before going through the steps below, one must know these facts about a B+ tree of degree **m**.

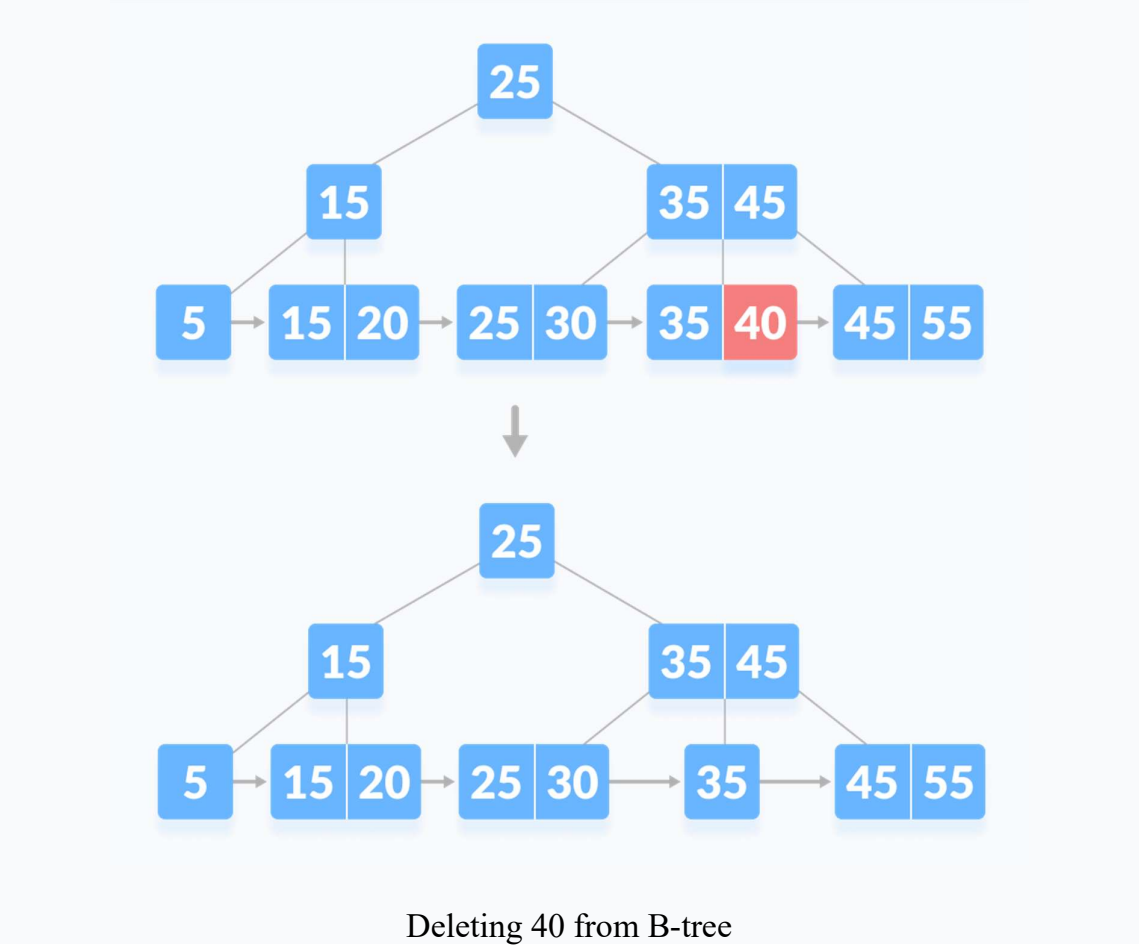
1. A node can have a maximum of m children. (i.e. 3)
2. A node can contain a maximum of $m - 1$ keys. (i.e. 2)
3. A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

4. A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)
   While deleting a key, we have to take care of the keys present in the internal nodes (i.e. indexes) as well because the values are redundant in a B+ tree. Search the key to be deleted then follow the following steps.

- **Case I**

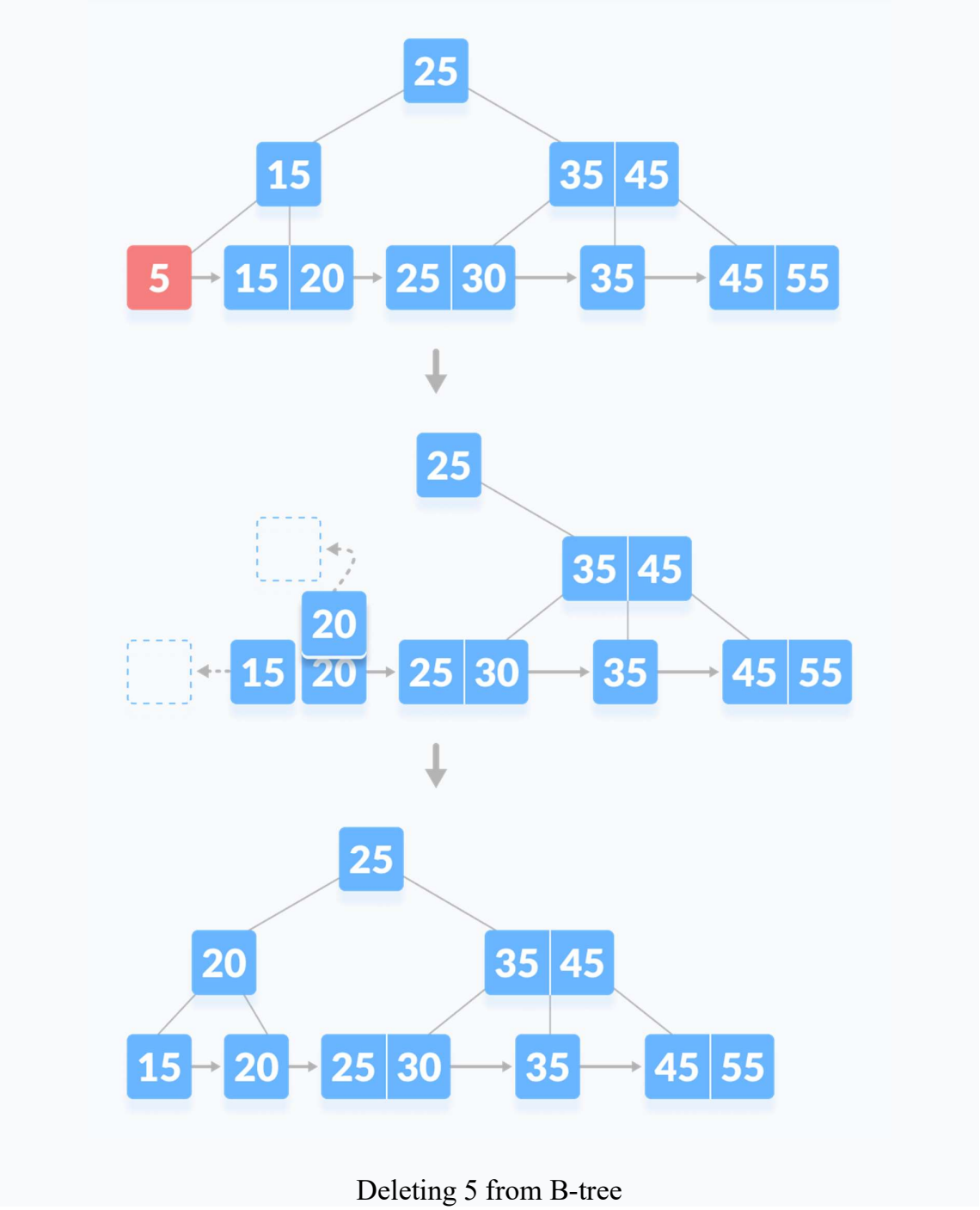
  The key to be deleted is present only at the leaf node not in the indexes (or internal nodes). There are two cases for it:

1. There is more than the minimum number of keys in the node. Simply delete the key.



Deleting 40 from B-tree

2. There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent

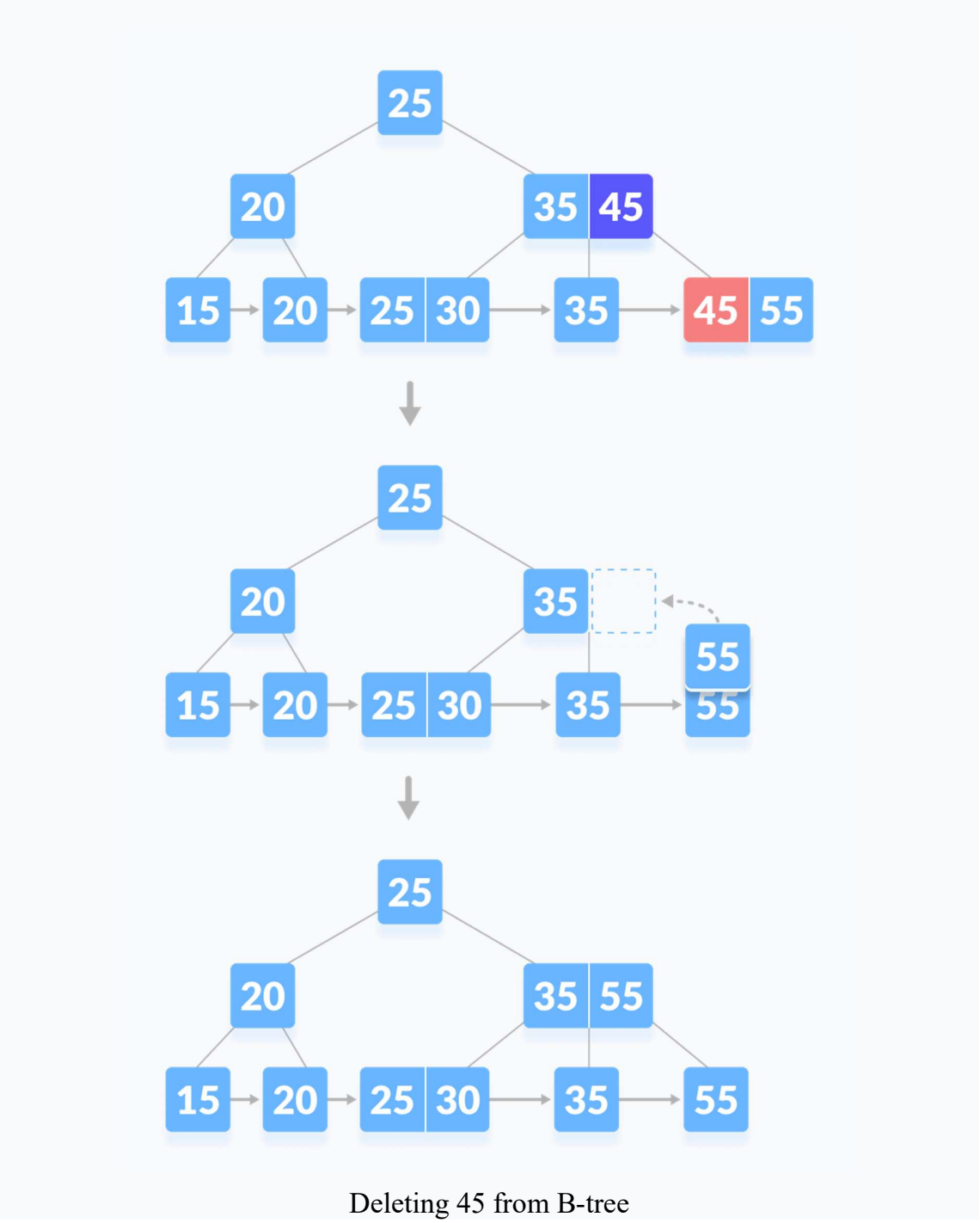


Deleting 5 from B-tree

- **Case II**

The key to be deleted is present in the internal nodes as well. Then we have to remove them from the internal nodes as well. There are the following cases for this situation.

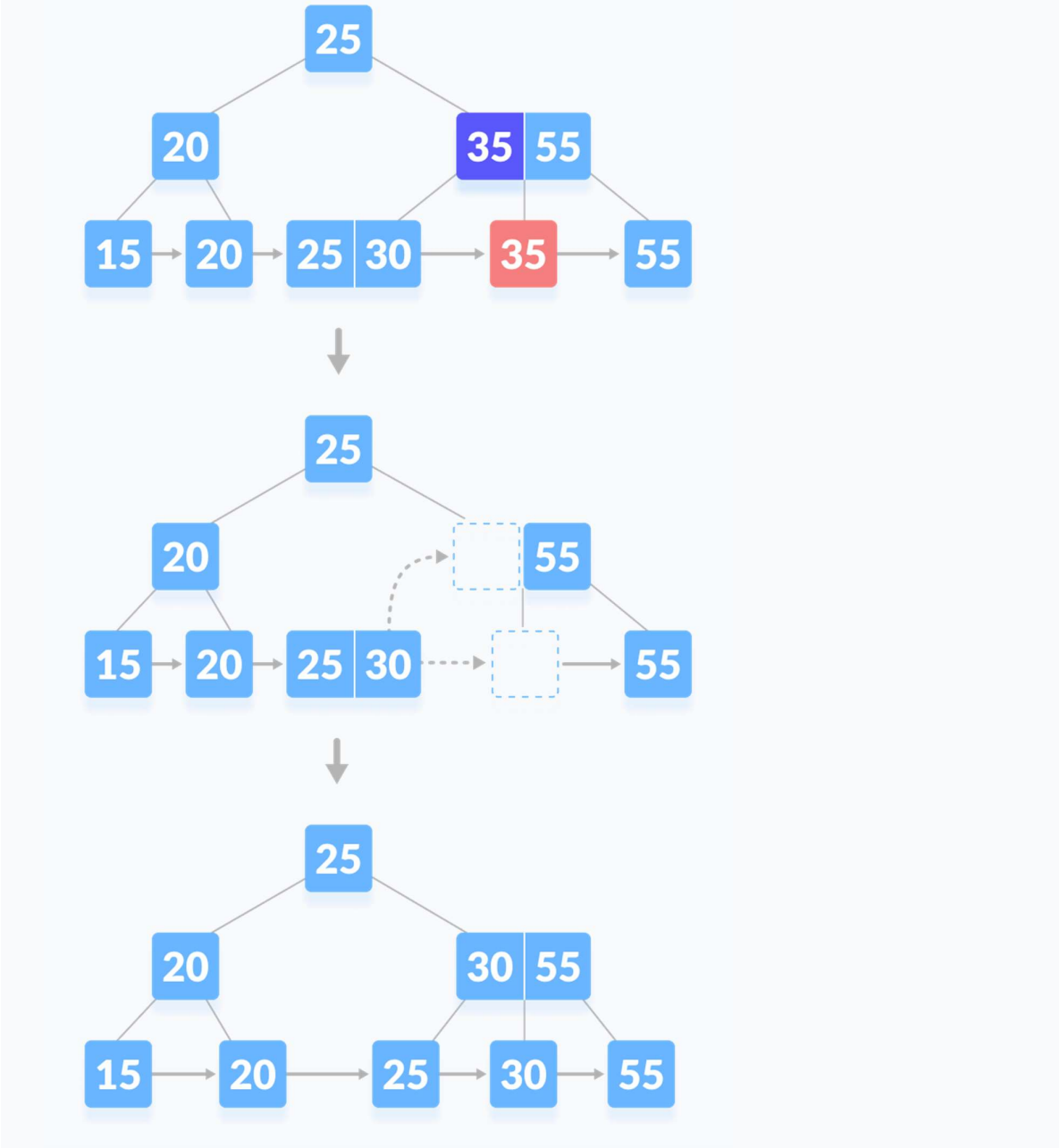
1. If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well.
   Fill the empty space in the internal node with the inorder successor.



Deleting 45 from B-tree

2. If there is an exact minimum number of keys in the node, then delete the key and borrow a key from its immediate sibling (through the parent).

Fill the empty space created in the index (internal node) with the borrowed key.

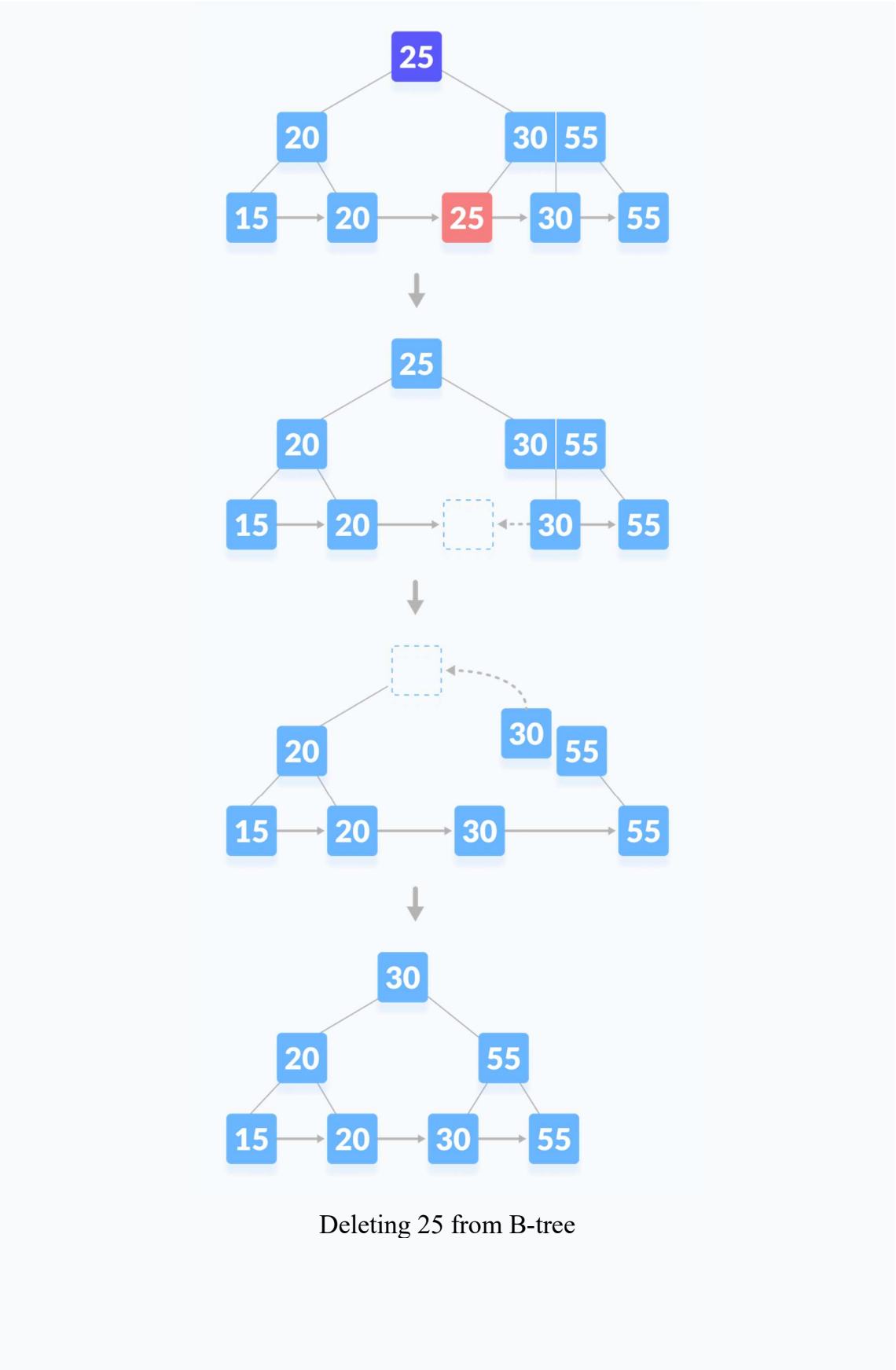


Deleting 35 from B-tree

3. This case is similar to Case II(1) but here, empty space is generated above the immediate parent node.

   After deleting the key, merge the empty space with its sibling.

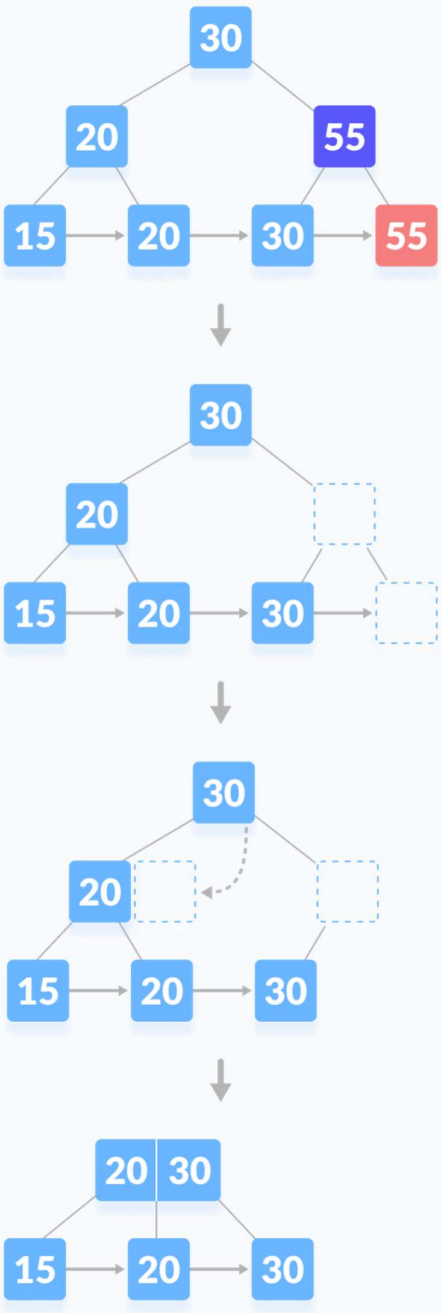
   Fill the empty space in the grandparent node with the inorder successor.



Deleting 25 from B-tree

○ Case III

In this case, the height of the tree gets shrinked. It is a little complicated.Deleting 55 from the tree below leads to this condition. It can be understood in the illustrations below.



Deleting 55 from B-tree

Code:

```
import random

splits = 0
parent_splits = 0
fusions = 0
parent_fusions = 0


class Node(object):

    def __init__(self, parent=None):

        self.keys: list = []
        self.values: list[Node] = []
        self.parent: Node = parent

    def index(self, key):

        for i, item in enumerate(self.keys):
            if key < item:
                return i

        return len(self.keys)

    def __getitem__(self, item):
        return self.values[self.index(item)]

    def __setitem__(self, key, value):
        i = self.index(key)
        self.keys[i:i] = [key]
        self.values.pop(i)
        self.values[i:i] = value

    def split(self):

        global splits, parent_splits
        splits += 1
        parent_splits += 1

        left = Node(self.parent)

        mid = len(self.keys) // 2

        left.keys = self.keys[:mid]
        left.values = self.values[:mid + 1]
        for child in left.values:
```

```
            child.parent = left

        key = self.keys[mid]
        self.keys = self.keys[mid + 1:]
        self.values = self.values[mid + 1:]

        return key, [left, self]



class Leaf(Node):
    def __init__(self, parent=None, prev_node=None, next_node=None):

        super(Leaf, self).__init__(parent)
        self.next: Leaf = next_node
        if next_node is not None:
            next_node.prev = self
        self.prev: Leaf = prev_node
        if prev_node is not None:
            prev_node.next = self

    def __getitem__(self, item):
        return self.values[self.keys.index(item)]

    def __setitem__(self, key, value):
        i = self.index(key)
        if key not in self.keys:
            self.keys[i:i] = [key]
            self.values[i:i] = [value]
        else:
            self.values[i - 1] = value

    def split(self):
        global splits
        splits += 1

        left = Leaf(self.parent, self.prev, self)
        mid = len(self.keys) // 2

        left.keys = self.keys[:mid]
        left.values = self.values[:mid]

        self.keys: list = self.keys[mid:]
        self.values: list = self.values[mid:]

        # When the leaf node is split, set the parent key to the left-most key
of the right child node.
        return self.keys[0], [left, self]
```

```python
class BPlusTree(object):

    root: Node

    def __init__(self, maximum=4):
        self.root = Leaf()
        self.maximum: int = maximum if maximum > 2 else 2
        self.minimum: int = self.maximum // 2
        self.depth = 0

    def find(self, key) -> Leaf:

        node = self.root
        # Traverse tree until leaf node is reached.
        while type(node) is not Leaf:
            node = node[key]

        return node

    def __getitem__(self, item):
        return self.find(item)[item]

    def query(self, key):
        leaf = self.find(key)
        return leaf[key] if key in leaf.keys else None

    def change(self, key, value):

        leaf = self.find(key)
        if key not in leaf.keys:
            return False, leaf
        else:
            leaf[key] = value
            return True, leaf

    def __setitem__(self, key, value, leaf=None):

        if leaf is None:
            leaf = self.find(key)
        leaf[key] = value
        if len(leaf.keys) > self.maximum:
            self.insert_index(*leaf.split())

    def insert(self, key, value):
```

```
        leaf = self.find(key)
        if key in leaf.keys:
            return False, leaf
        else:
            self.__setitem__(key, value, leaf)
            return True, leaf

    def insert_index(self, key, values: list[Node]):
        """For a parent and child node,
                    Insert the values from the child into the values of the
parent."""
        parent = values[1].parent
        if parent is None:
            values[0].parent = values[1].parent = self.root = Node()
            self.depth += 1
            self.root.keys = [key]
            self.root.values = values
            return

        parent[key] = values
        # If the node is full, split the  node into two.
        if len(parent.keys) > self.maximum:
            self.insert_index(*parent.split())




    def show(self, node=None, file=None, _prefix="", _last=True):
        """Prints the keys at each level."""
        if node is None:
            node = self.root
        print(_prefix, "`- " if _last else "|- ", node.keys, sep="",
file=file)
        _prefix += "   " if _last else "|  "

        if type(node) is Node:
            # Recursively print the key of child nodes (if these exist).
            for i, child in enumerate(node.values):
                _last = (i == len(node.values) - 1)
                self.show(child, file, _prefix, _last)

def demo():
    bplustree = BPlusTree()
    flag = True

    while(flag):
```

```
        print("\n******** B+ Tree ********")
        print("1.Insert")
        print("2.Search")
        print("3.Show Tree")
        print("4.Exit")
        n = int(input("Enter Your Choice : "))


        if n == 1:
            value = int(input("Enter Element to Insert : "))
            bplustree[value] = value

        elif n == 2:
            value = int(input("Enter Element to Search : "))
            ans = bplustree.find(value)
            if ans.values:
                print("Value found in the tree")
            else:
                print("Value not found in the tree")

        elif n == 3:
            bplustree.show()
        elif n == 4:
            flag = False
            break
        else:
            flag = False
            break

if __name__ == '__main__':

    demo()
```

Output:

```
PS C:\Users\rudra\OneDrive - Shri Vile Parle Kelavani Mandal\Sem 5\adbms> & C:
e "c:/Users/rudra/OneDrive - Shri Vile Parle Kelavani Mandal/Sem 5/adbms/b+.py

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 12

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 34

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 23

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
```



```
Enter Your Choice : 1
Enter Element to Insert : 23

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 45

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 3
`- [12, 23, 34, 45]

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 55

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
```

```
Enter Your Choice : 1
Enter Element to Insert : 55

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 27

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 78

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 47

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
```

```
******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 1
Enter Element to Insert : 47

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice : 3
`- [34, 47]
   |- [12, 23, 27]
   |- [34, 45]
   `- [47, 55, 78]

******** B+ Tree ********
1.Insert
2.Search
3.Show Tree
4.Exit
Enter Your Choice :
```

Ln 236, Col 1 Spaces: 4 UT

Conclusion: We successfully implement B+ Tree.

Experiment 3

ADBMS

Ayush Jain 60004200132 B2

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

Theory:

Query Execution Without Optimization

Query 1
Finding Average Marks of students

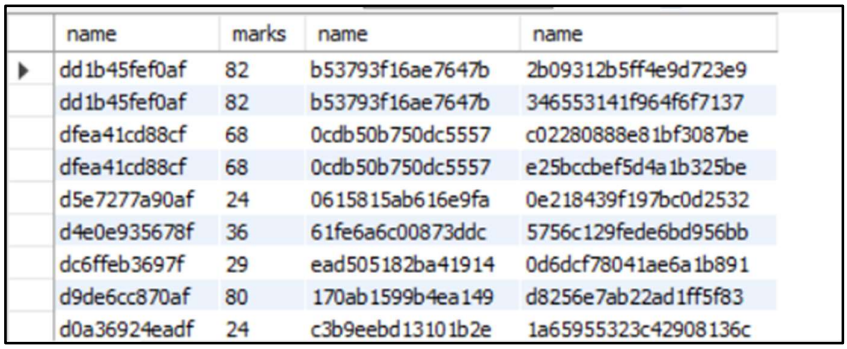
avg(marks)
43.1667
3.0000
33.0000
6.0000
47.0000
68.0000
28.3333
60.0000
63.2500

```
use rudra;
select avg(marks)
from student
group by fav_teacher;
```

Timing (as measured at client side):
Execution time: 0:00:0.01500000

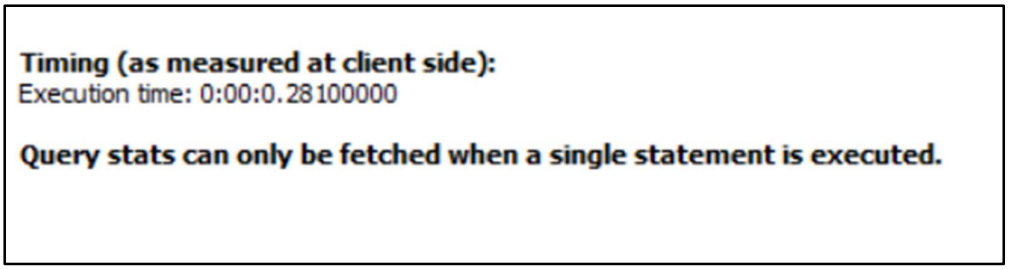
Query stats can only be fetched when a single statement is executed.

Query 2



	name	marks	name	name
▶	dd1b45fef0af	82	b53793f16ae7647b	2b09312b5ff4e9d723e9
	dd1b45fef0af	82	b53793f16ae7647b	346553141f964f6f7137
	dfea41cd88cf	68	0cdb50b750dc5557	c02280888e81bf3087be
	dfea41cd88cf	68	0cdb50b750dc5557	e25bccbef5d4a1b325be
	d5e7277a90af	24	0615815ab616e9fa	0e218439f197bc0d2532
	d4e0e935678f	36	61fe6a6c00873ddc	5756c129fede6bd956bb
	dc6ffeb3697f	29	ead505182ba41914	0d6dcf78041ae6a1b891
	d9de6cc870af	80	170ab1599b4ea149	d8256e7ab22ad1ff5f83
	d0a36924eadf	24	c3b9eebd13101b2e	1a65955323c42908136c

Performing Join to get subjects taught by a student's favourite teacher



Timing (as measured at client side):
Execution time: 0:00:0.28100000

Query stats can only be fetched when a single statement is executed.

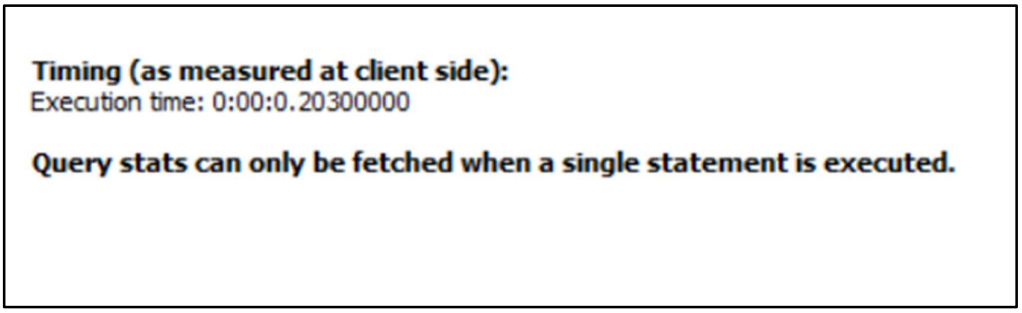
```
use rudra;
select student.name , student.marks , teacher.name , subject.name
from student
inner join teacher on teacher.teacher_id = student.fav_teacher
inner join teaches on teaches.teacher_id = teacher.teacher_id
inner join subject on subject.subject_id = teaches.subject_id
where student.name like "d%e%f";
```

Query 3

Query using Join to find names of subject taught by students favourite teacher where student has marks greater than 75

```
use rudra;
select teacher.name , student.name
from student
inner join teacher on teacher.teacher_id = student.fav_teacher
inner join teaches on teaches.teacher_id = teacher.teacher_id
```

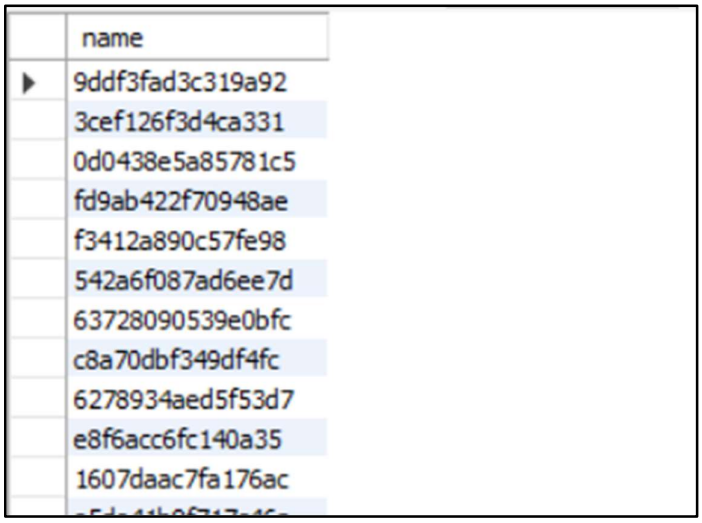
```
inner join subject on subject.subject_id = teaches.subject_id
where student.name like "ef%6%d"
and
student.marks > 75
limit 25
offset 75000;
```



Query 4

Query to find name of the teacher that teaches pre requisite of a subject thought by a teacher.

```
use rudra;
select teacher.name
from teacher
where teacher.teacher_id in(
        select teacher.teacher_id
   from subject
   inner join subject on subject.pre_req = subject.subject_id
   inner join teaches on teaches.subject_id = subject.subject_id
   where
   student.subject_id in (
                select teacher.subject_id
      from student
      inner join teacher on teacher.teacher_id = student.fav_teacher
      inner join teaches on teacher.teacher_id = teaches.teacher_id
        )
)
```



	name
▶	9ddf3fad3c319a92
	3cef126f3d4ca331
	0d0438e5a85781c5
	fd9ab422f70948ae
	f3412a890c57fe98
	542a6f087ad6ee7d
	63728090539e0bfc
	c8a70dbf349df4fc
	6278934aed5f53d7
	e8f6acc6fc140a35
	1607daac7fa176ac

Timing (as measured at client side):
Execution time: 0:00:0.46900000

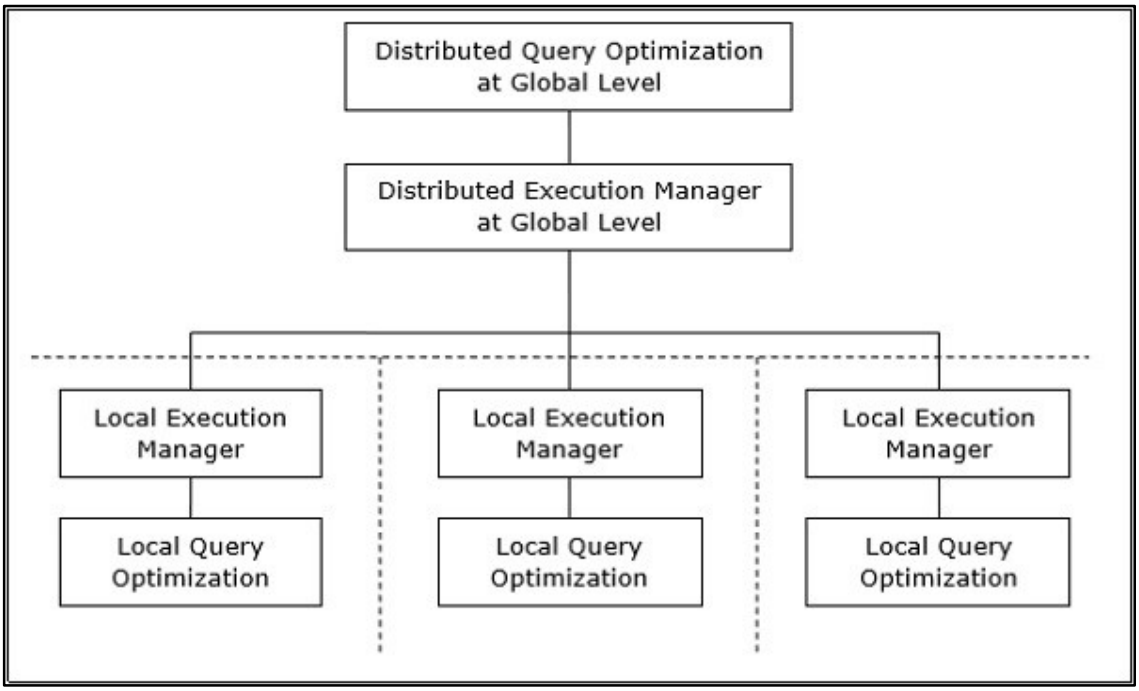
Query stats can only be fetched when a single statement is executed.

Theory:

Distributed Query Processing Architecture

In a distributed database system, processing a query comprises of optimization at both the global and the local level. The query enters the database system at the client or controlling site. Here, the user is validated, the query is checked, translated, and optimized at a global level.

The architecture can be represented as −



Mapping Global Queries into Local Queries

The process of mapping global queries to local ones can be realized as follows −

- The tables required in a global query have fragments distributed across multiple sites. The local databases have information only about local data. The controlling site uses the global data dictionary to gather information about the distribution and reconstructs the global view from the fragments.
- If there is no replication, the global optimizer runs local queries at the sites where the fragments are stored. If there is replication, the global optimizer selects the site based upon communication cost, workload, and server speed.
- The global optimizer generates a distributed execution plan so that least amount of data transfer occurs across the sites. The plan states the location of the fragments, order in which query steps needs to be executed and the processes involved in transferring intermediate results.
- The local queries are optimized by the local database servers. Finally, the local query results are merged together through union operation in case of horizontal fragments and join operation for vertical fragments.

Distributed Query Optimization

Distributed query optimization requires evaluation of a large number of query trees each of which produce the required results of a query. This is primarily due

to the presence of large amount of replicated and fragmented data. Hence, the target is to find an optimal solution instead of the best solution.

The main issues for distributed query optimization are −

- Optimal utilization of resources in the distributed system.
- Query trading.
- Reduction of solution space of the query.

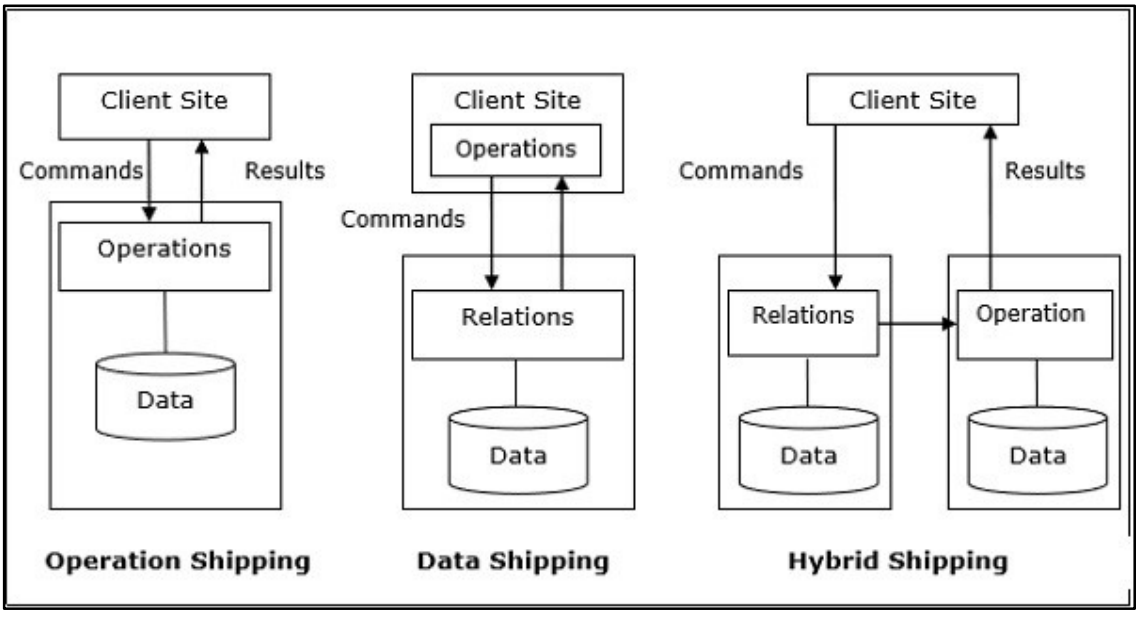
Optimal Utilization of Resources in the Distributed System

A distributed system has a number of database servers in the various sites to perform the operations pertaining to a query. Following are the approaches for optimal resource utilization −

Operation Shipping − In operation shipping, the operation is run at the site where the data is stored and not at the client site. The results are then transferred to the client site. This is appropriate for operations where the operands are available at the same site. Example: Select and Project operations.

Data Shipping − In data shipping, the data fragments are transferred to the database server, where the operations are executed. This is used in operations where the operands are distributed at different sites. This is also appropriate in systems where the communication costs are low, and local processors are much slower than the client server.

Hybrid Shipping − This is a combination of data and operation shipping. Here, data fragments are transferred to the high-speed processors, where the operation runs. The results are then sent to the client site.



Index Optimization:

Index in SQL Server is used to retrieve requested data speedily from database tables. There are two types of index are used in SQL server i.e. clustered and non-clustered indexes. In which, the non-clustered index is better as compare to cluster index because it has index key values and it retrieves data more rapidly as compare to clustered index. However, sometimes non-clustered index can also take a huge amount of time to retrieve data. Therefore, if a user wants then they can optimize SQL indexes to fetch data more rapidly. In this post, we are going to discuss various SQL server index optimization best practices.

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is like what we see in books. Indexing is defined based on its indexing attributes. Indexing can be of the following types

- - Primary Index− Primary index is defined on an ordered data file. The data file is ordered on akey field. The key field is generally the primary key of the relation.
- Secondary Index− Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- Clustering Index− Clustering index is defined on an ordered data file. The data file is ordered on a non-keyfield Ordered Indexing is of two types –
- Dense Index
- Sparse Index

So indexing is basically an efficient way of retrieval of data with low access time on the disc.

Executing previously executed queries again with Index Level Optimization For comparison in the speed of execution

Creating Index on tables:

```
create index student_index on rudra.student(stud_id ,
name,marks,fav_teacher);
create index subject_index on rudra.subject(subject_id,name,pre_req);
create index teacher_inex on rudra.teacher(teacher_id , name);
create index teaches_index on rudra.teaches(teacher_id , subject_id);
```

✔	50	15:59:20	create index student_index on jay.student(stud_id , name,marks,fav_teacher)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.781 sec
✔	51	16:23:55	create index subject_index on jay.subject(subject_id,name,pre_req)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	1.000 sec
✔	52	16:24:31	create index teacher_inex on jay.teacher(teacher_id , name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.719 sec
✔	53	16:25:03	create index teaches_index on jay.teaches(teacher_id , subject_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.156 sec

Query 1

Finding Average Marks of students

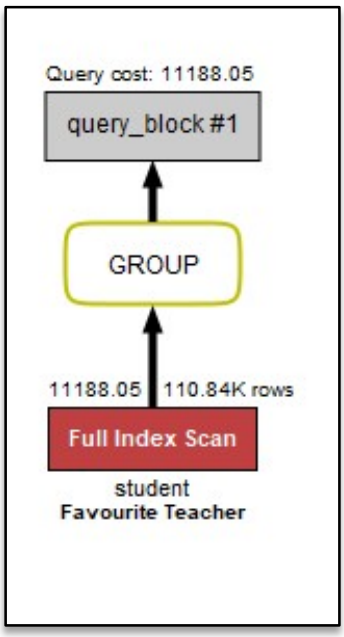
```
1 •  use jay;
2    select avg(marks)
3    from student
4    group by fav_teacher;
5
```

Query Statistics

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

Query stats can only be fetched when a single statement is executed.

Execution Plan



Query 2

Performing Join to get subjects taught by a student's favourite teacher

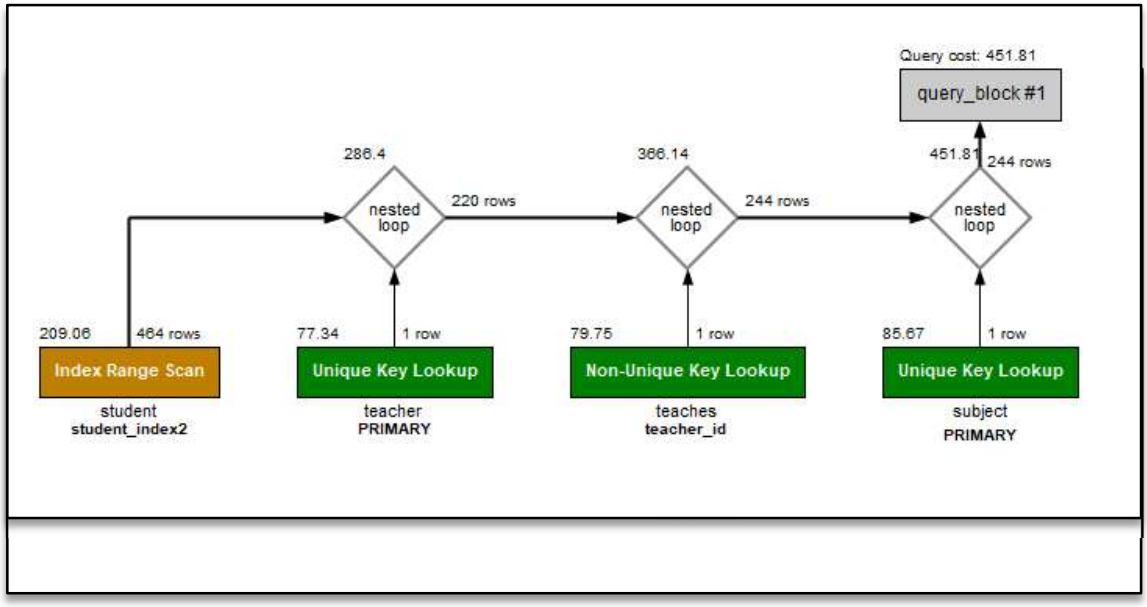
```
2    select student.name , student.marks , teacher.name , subject.name
3    from student
4    inner join teacher on teacher.teacher_id = student.fav_teacher
5    inner join teaches on teaches.teacher_id = teacher.teacher_id
6    inner join subject on subject.subject_id = teaches.subject_id
7    where student.name like "d%e%f";
8
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.21800000

Query stats can only be fetched when a single statement is executed.

Execution Plan



Query 3

Query using Join to find names of subject taught by students favourite teacher where student has marks greater than 75

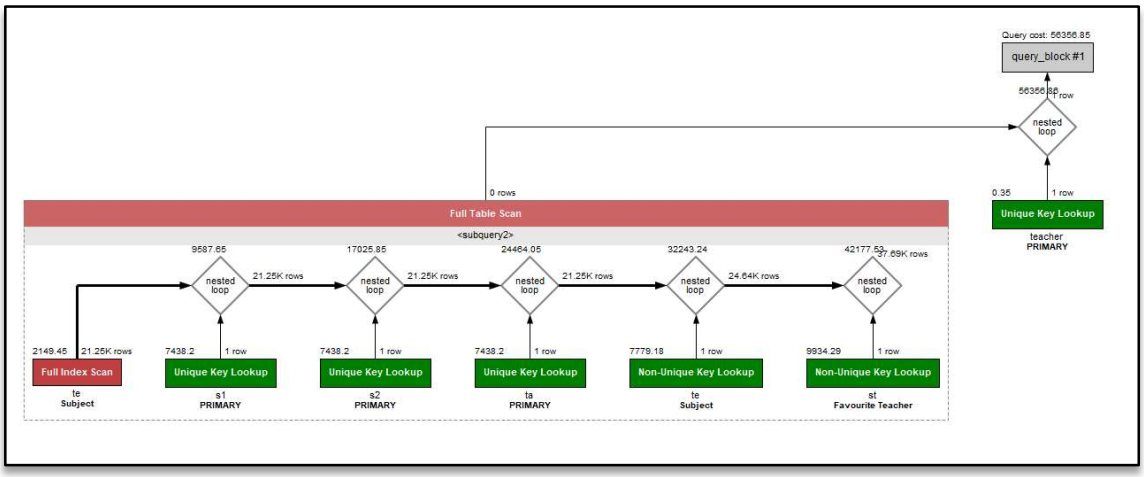
```
1 •  use jay;
2    select teacher.name , student.name
3    from student
4    inner join teacher on teacher.teacher_id = student.fav_teacher
5    inner join teaches on teaches.teacher_id = teacher.teacher_id
6    inner join subject on subject.subject_id = teaches.subject_id
7    where student.name like "ef%6%d"
8    and
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01500000

Query stats can only be fetched when a single statement is executed.

Execution Plan



Query 4

Query to find name of the teacher that teaches pre requisite of a subject thought by a teacher.

```
1  use jay;
2  select teacher.name
3  from teacher
4  where teacher.teacher_id in(
5      select teacher.teacher_id
6      from subject
7      inner join subject on subject.pre_req = subject.subject_id
8      inner join teaches on teaches.subject_id = subject.subject_id
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.36000000

Query stats can only be fetched when a single statement is executed.

Execution Plan

Conclusion: Hence, we learned about database optimization using indices and Table level optimization. We performed some queries and performed before after analysis to conclude that optimization indeed improved runtime of our queries but also substantially increased disk space

Experiment 4

ADBMS

Ayush Jain 60004200132 B2

Aim: Implement Query Monitor (QEP - Query Execution Plan, Query Statistics).

Theory:

The most commonly used SQL command is SELECT statement. SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

Syntax-

SELECT [DISTINCT|ALL] { * | [fieldExpression [AS newName]} FROM tableName [alias] [WHERE condition][GROUP BY fieldName(s)] [HAVING condition] ORDER BY fieldName(s)

- **SELECT** is the SQL keyword that lets the database know that you want to retrieve data.
- **[DISTINCT | ALL]** are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
- **{*| [fieldExpression [AS newName]}** at least one part must be specified, "*" selected all the fields from the specified table name, fieldExpression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
- **FROM** tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.
- **WHERE** condition is optional, it can be used to specify criteria in the result set returned from the query.
- **GROUP BY** is used to put together records that have the same field values.
- **HAVING** condition is used to specify criteria when working using the GROUP BY keyword.
- **ORDER BY** is used to specify the sort order of the result set.

Queries:

1. Select all column from a table
Select * from tablename;
2. Select specific column of list from a table
Select col1,col2 from tablename;
3. Select ….where clause with various operators (<,>,<=,> =,IN,NOT IN,BETWEEN..AND,NOT BETWEEN… AND)
Select col1 from tablename where col2>value1;
Select * from tablename where col2 in(value1,value2,value3)
Select * from tablename where col2 between value1 and value2
Select * from tablename where col2 NOT between value1 and value2

4. Select ….where clause with multiple conditions
Select * from tablename where col2=Value1 and/or col3=value2

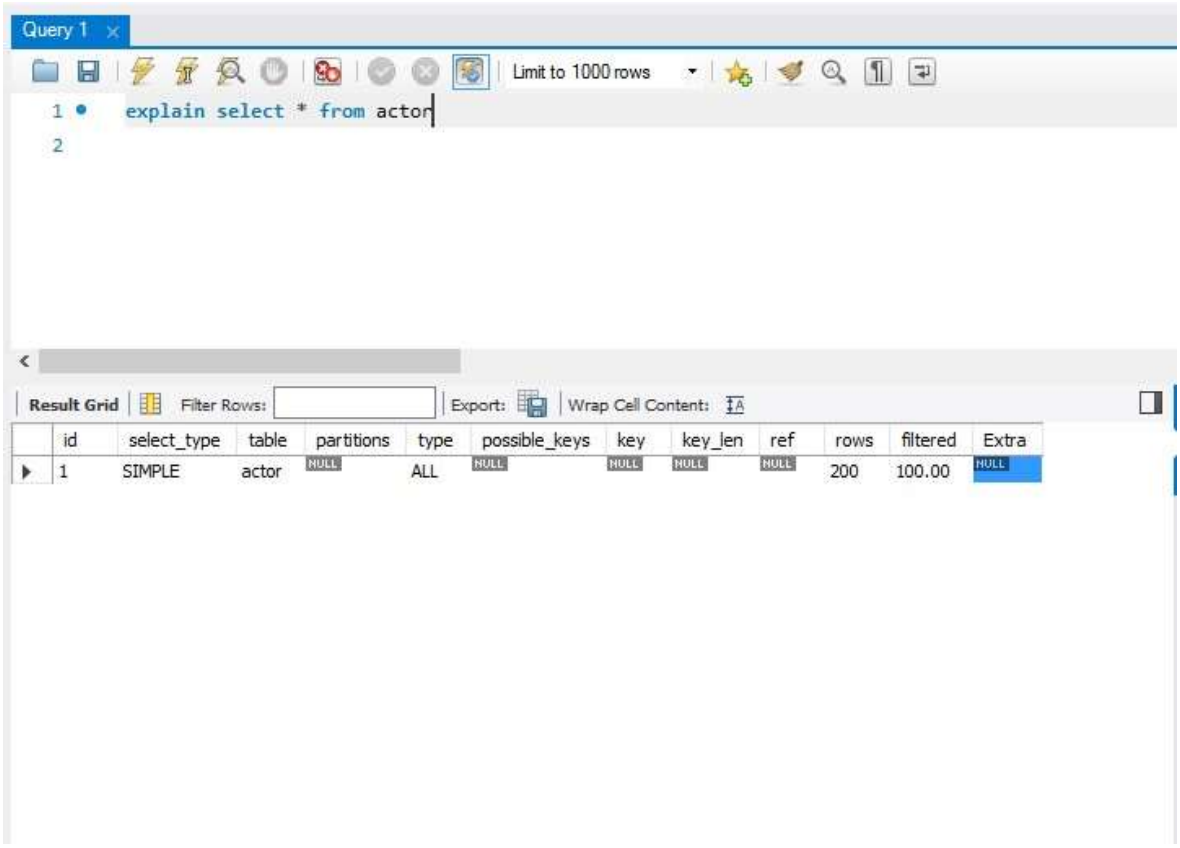
5. Select ….where clause with string matching
- Starts with any character
- ends with any character
- have a specific substring
- character at specific position
- starts with a specific character and having specifically n no of characters
- starts and ends with different character

6. Query your database by applying aggregate function
MIN,MAX,COUNT,AVG,SUM (create alias also)
7. Query your database by applying group by clause using one column and multiple column
8. Query your database by applying order by clause using one column and multiple column
9. Query your database by applying group by, order by and having clause simultaneously
10. Select clause with various date functions
CURDATE(),CURRENT_TIME(),CURRENT_TIMESTAMP(),DATE(),DATEDIFF(),DAY(),DAYNAME(),EXTRACT(),MINUTE(),MONTH(),WEEK(),NOW(),YEAR()

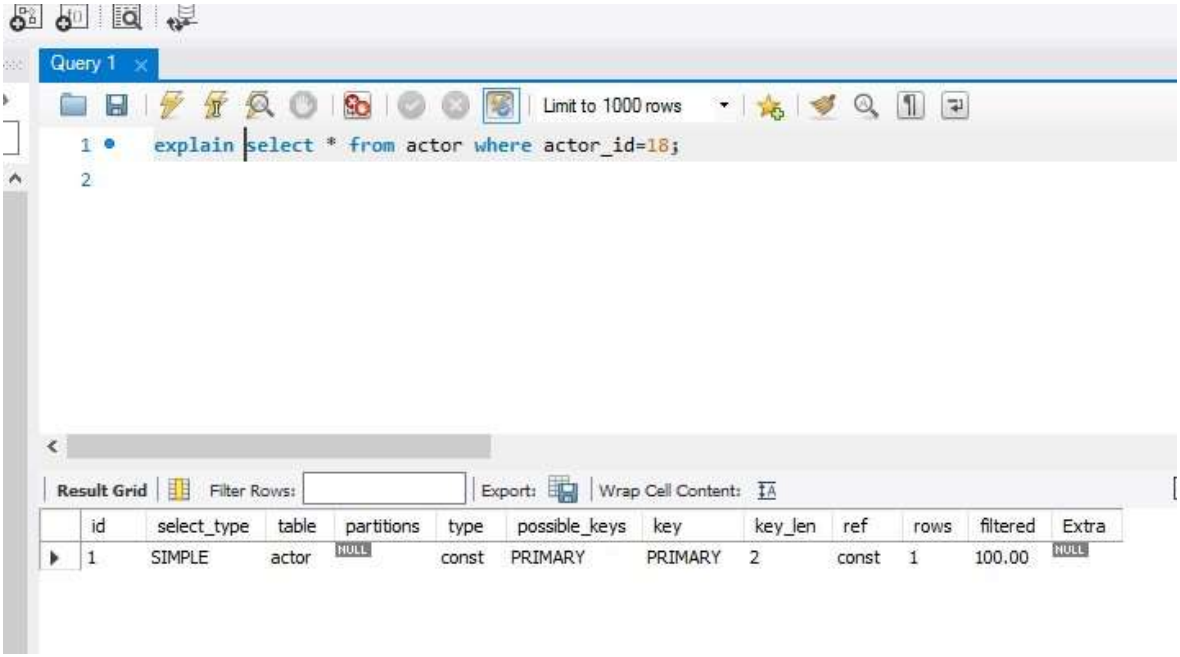
11. Perform set operation on multiple select queries(UNION)

Output:

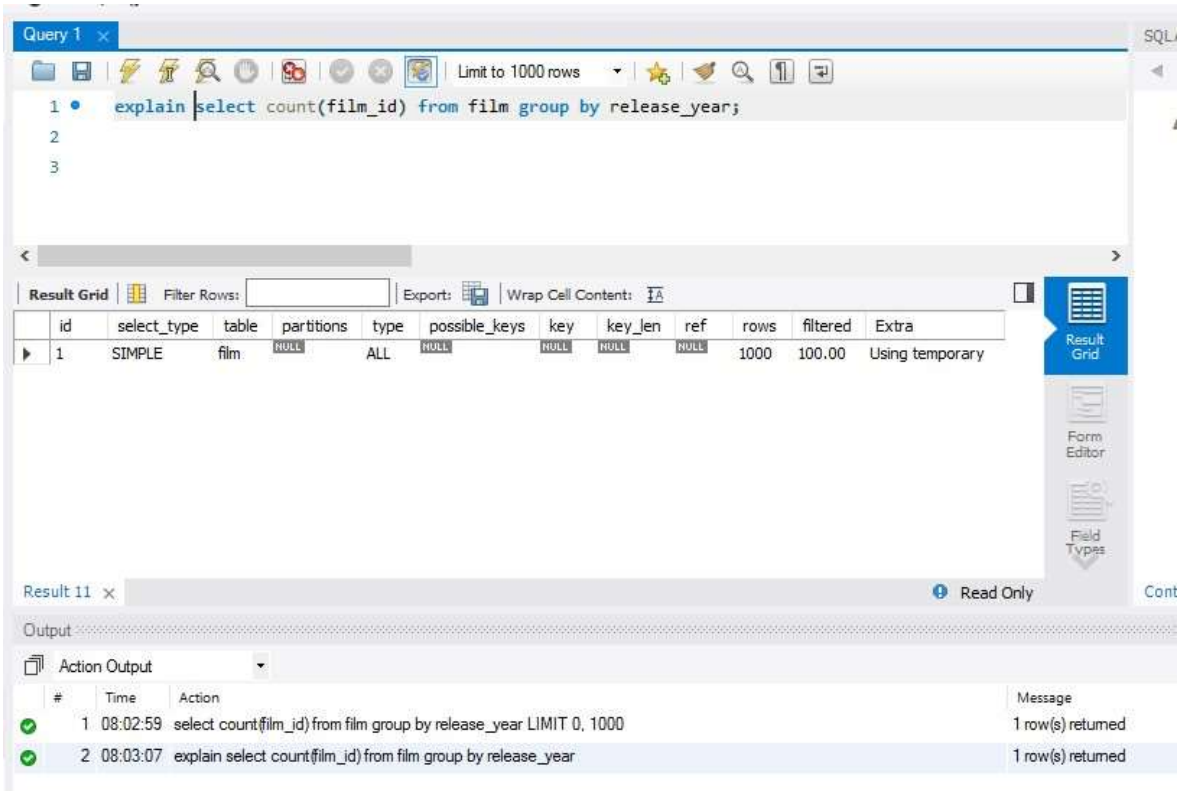
SELECT Query



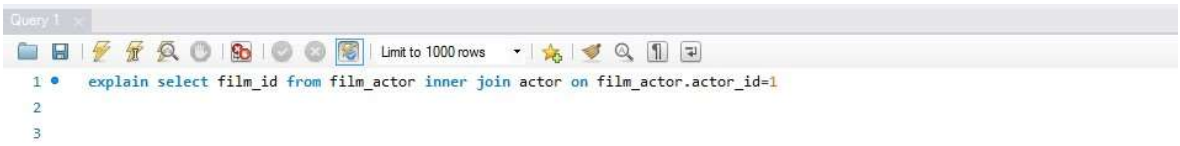
SELECT WITH WHERE

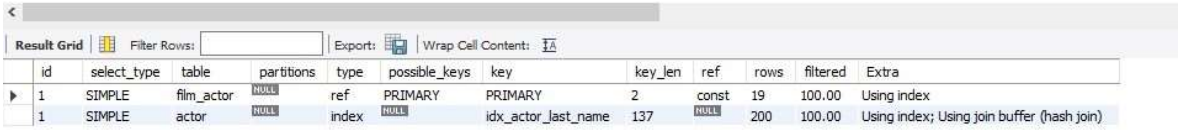


GROUP BY



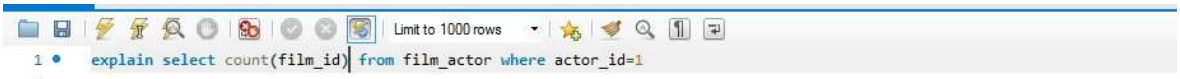
JOIN

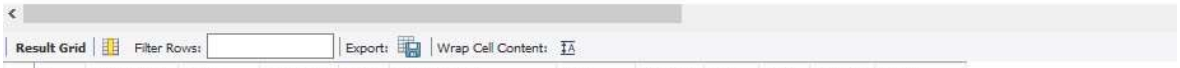




id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film_actor	NULL	ref	PRIMARY	PRIMARY	2	const	19	100.00	Using index
1	SIMPLE	actor	NULL	index	NULL	idx_actor_last_name	137	NULL	200	100.00	Using index; Using join buffer (hash join)

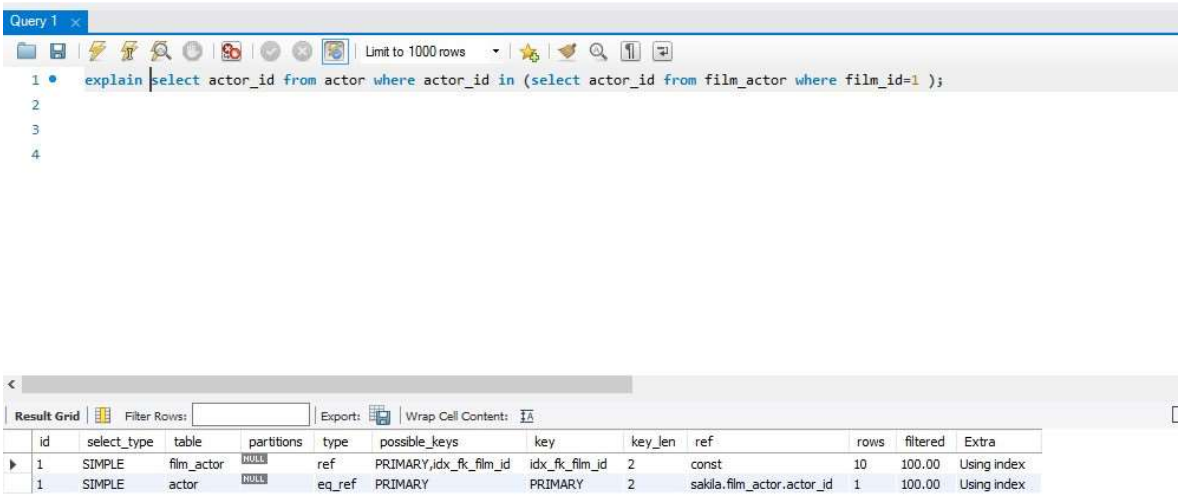
AGGREGATE (COUNT)





id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film_actor	NULL	ref	PRIMARY,idx_fk_film_id	PRIMARY	2	const	19	100.00	Using index

NESTED QUERIES



Conclusion: Thus, we implemented query monitor and saw how **explain** SQL command describes how SQL queries are executed in the database.

Experiment 5

ADBMS

Ayush Jain 60004200132 B2

AIM: Perform Fragmentation (Range,List,Hash and Key) in DDBS Design

Theory:

What is Partitioning Key
Partitioning key is comprised of one or more columns that determine the partition where each row will be stored. Oracle automatically directs insert, update and delete operations to the appropriate partition through the use of partitioning key.

When to partition table: suggestions
· Tables more than 2GB in size can be considered for candidates for partition
· Historical data tables where recent months data is frequently used
· Huge table needs to be distributed across the different types of storage devices

How benefited in Indexes
· Avoid rebuilding the entire index when data is removed
· Can perform maintenance on part of data without invalidating the entire index

Types of partitions: Different strategies

1. Range
· This is the commonly used partition technique and is often used with date as the partitioning key.
· In Range partition the partitions are identified on the range of values of the partitioning key.
· Each partition has a VALUES LESS THAN clause, which specifies a non-inclusive upper bound for the partitions. Any values of the partitioning key equal to or higher than this key are added to the next higher partition.
· MAXVALUE literal can be defied for the highest partition.

2. Hash

· Oracle uses hashing algorithm on the partitioning key for the data mapping to the partitions.

· Hash algorithm evenly distributes the row among partitions, so the partitions would be approximately same size.

· Hash is Ideal for distributing the data across devices.

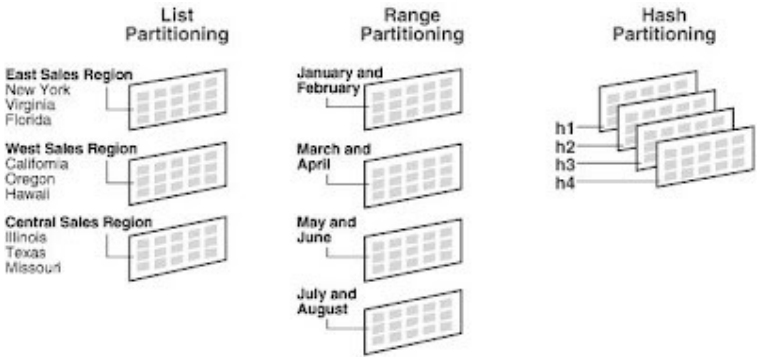
· It is more suits for the non historical data or has no obvious partitioning key

3. List

· More suits for a list of discrete values for the partitioning key

· Can group and organize unordered and unrelated set of data in a natural way

· DEFAULT partition enables you to avoid specifying all possible values for a list-partitioned table by using a default partition, so that all rows that do not map to any other partition do not generate any error.



List, Range and Hash Partitioning

4. Interval

· Introduced in Oracle 11g

· Interval partitions are extensions to range partition

· Technology provides automation for equi-sized range partitions.

· Syntax Keyword: INTERVAL (NUMTOYMINTERVAL(1,'month') numtoyminterval function converts a number to an INTERVAL YEAR TO MONTH literal(‘YEAR’ or ‘MONTH’)

· Using SET INTERVAL option of ALTER TABLE command can extend a range partitioned table to interval partitioned table

OUTPUT

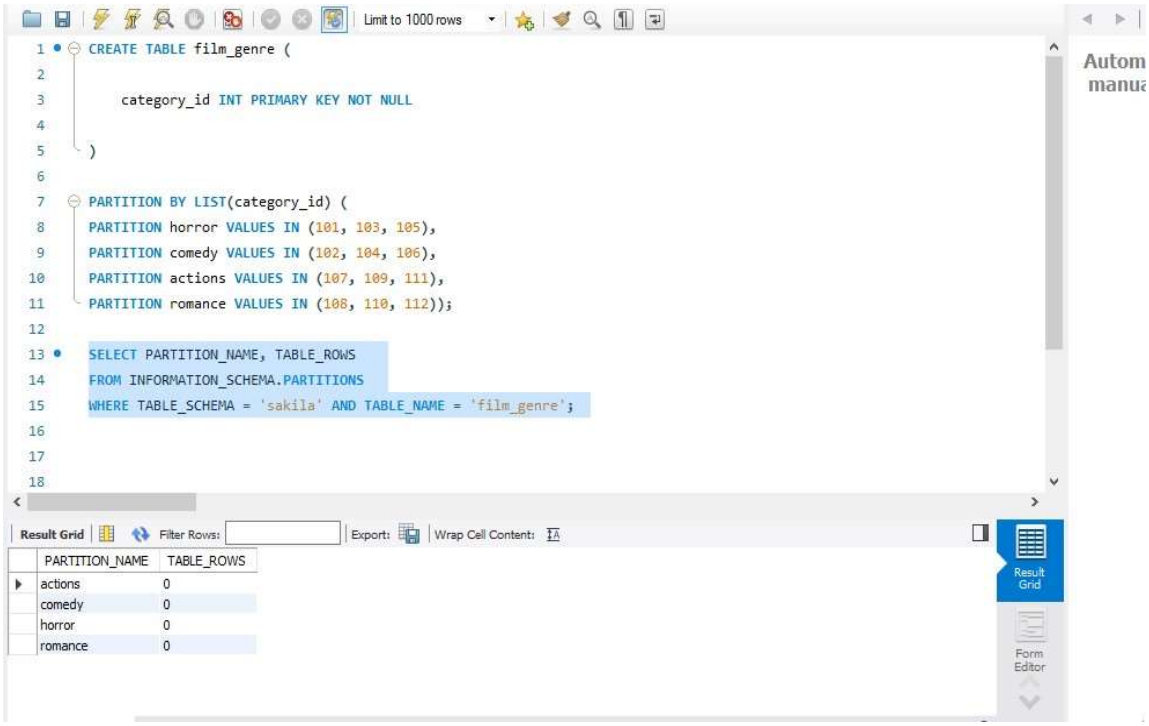
- RANGE

```
1  alter table film_year
2  PARTITION BY RANGE (year(film_years))(
3  PARTITION p0 VALUES LESS THAN (2016),
4  PARTITION p1 VALUES LESS THAN (2017),
5  PARTITION p2 VALUES LESS THAN (2018),
6  PARTITION p3 VALUES LESS THAN (2020));
7
8
9
10 SELECT PARTITION_NAME, TABLE_ROWS
11 FROM INFORMATION_SCHEMA.PARTITIONS
12 WHERE TABLE_SCHEMA = 'sakila' AND TABLE_NAME = 'film_year';
13
```

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

PARTITION_NAME	TABLE_ROWS
p0	2
p1	2
p2	2
p3	0

- **LIST**



```
CREATE TABLE film_genre (

    category_id INT PRIMARY KEY NOT NULL

)

PARTITION BY LIST(category_id) (
PARTITION horror VALUES IN (101, 103, 105),
PARTITION comedy VALUES IN (102, 104, 106),
PARTITION actions VALUES IN (107, 109, 111),
PARTITION romance VALUES IN (108, 110, 112));

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

Result Grid

PARTITION_NAME	TABLE_ROWS
actions	0
comedy	0
horror	0
romance	0

- **HASH**

```
19
20  CREATE TABLE ActorDetail (
21      actor_id INT NOT NULL UNIQUE KEY,
22      actor_name VARCHAR(40)
23  )
24  PARTITION BY KEY()
25  PARTITIONS 2;
26
27
28
29  Insert into ActorDetail values
30   (1,'Salman'),
31   (2,'SRK'),
32   (3,'HERO');
33
34  SELECT PARTITION_NAME, TABLE_ROWS
35  FROM INFORMATION_SCHEMA.PARTITIONS
36  WHERE TABLE_SCHEMA = 'sakila' AND TABLE_NAME = 'ActorDetail';
```

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

PARTITION_NAME	TABLE_ROWS
p0	2
p1	1

- **KEY**

```
19
20  CREATE TABLE ActorDetail (
21      actor_id INT NOT NULL UNIQUE KEY,
22      actor_name VARCHAR(40)
23  )
24  PARTITION BY KEY()
25  PARTITIONS 2;
26
27
28
29  Insert into ActorDetail values
30   (1,'Salman'),
31   (2,'SRK'),
32   (3,'HERO');
33
34  SELECT PARTITION_NAME, TABLE_ROWS
35  FROM INFORMATION_SCHEMA.PARTITIONS
36  WHERE TABLE_SCHEMA = 'sakila' AND TABLE_NAME = 'ActorDetail';
```

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

PARTITION_NAME	TABLE_ROWS
p0	2
p1	1

CONCLUSION:

Partitioning is powerful functionality that allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.

Experiment 6

ADBMS

Ayush Jain 60004200132 B2

Aim: To implement 2 phase and 3phase commit protocol in distributed system

Theory:

Two phase commit

This protocol requires a coordinator. The client contacts the coordinator and proposes a value. The coordinator then tries to establish the consensus among a set of processes (a.k.a Participants) in two phases, hence the name.

1. In the first phase, coordinator contacts all the participants suggests value proposed by the client and solicit their response.

2. After receiving all the responses, the coordinator makes a decision to commit if all participants agreed upon the value or abort if someone disagrees.

3. In the second phase, coordinator contacts all participants again and communicates the commit or abort decision.

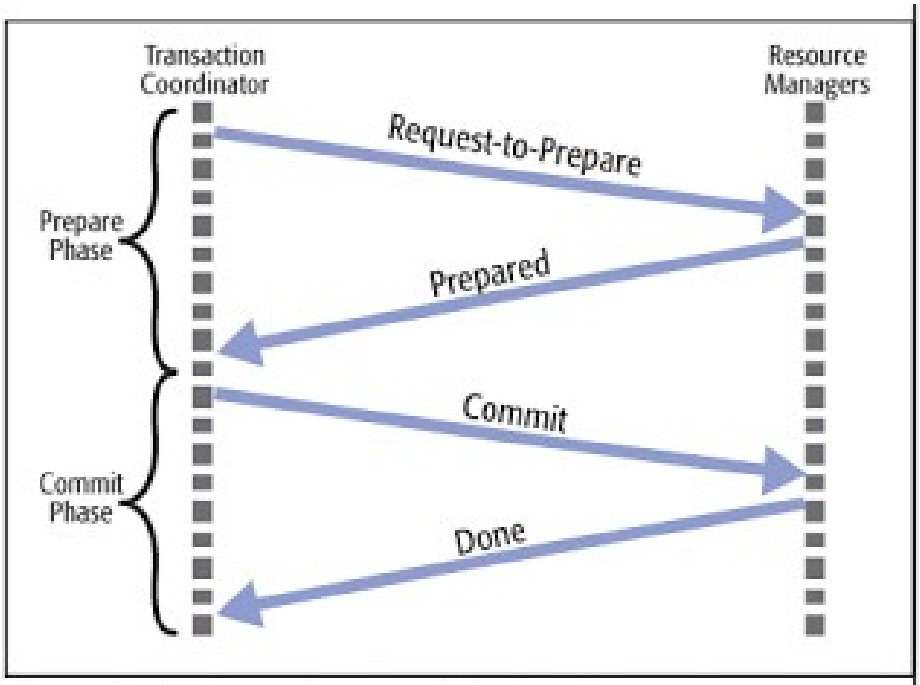


Figure 1 • The two-phase commit protocol

We can see that all the above-mentioned conditions are met. The agreement is there because the participants only make a yes or no decision on the value proposed by the coordinator and don't propose values. Validity is there because the same decision to commit or abort is enforced by the coordinator on all participants. Termination is guaranteed only if all participants communicate the responses to the coordinator. However, this is prone to failures.

When speaking about failures what are the types of failures of a node?

Fail-Stop Model, Nodes just crash and don't recover at all.

Fail Recover Model, Nodes crash, and recover after a certain time and continue executing.

Byzantine failure, Nodes start behaving arbitrarily trying to interrupt the regular behavior.

Now coming to how failures affect 2 Phase commit,

1. Coordinator fails even before initiating phase 1. This literally means the consensus isn't started at all and theoretically the protocol works correctly.

2. Coordinator fails after initiating phase 1. Some nodes have received the message from coordinator initiating a fresh round of 2PC. These nodes might have sent their responses and are blocked waiting for the 2nd phase of 2PC to start. This also means that no future consensus rounds of 2PC can start. One way out of this issue is to have time outs when waiting for responses. So when a node times out waiting for a response from the coordinator it can assume that coordinator is dead and take over the role as coordinator. It can reinitiate phase 1 and contact all other nodes asking them for the consensus based on the value for which this node voted as a participant before the actual coordinator crashed. However if another node crashes before recovery node gathers all messages of phase 1, then the protocol can't proceed. This is because recovery node doesn't know what's the intended decision of the crashed node. If all other participant nodes have agreed to commit but the newly crashed node might have intended to abort. So the recovery node can't call the decision as a commit. This argument applies vice versa also.

3. Similarly, if a participant fails during phase 1 before the coordinator receives a response from the participant, the protocol comes to a grinding halt. The reasoning is similar as point 2, because coordinator doesn't know

the result of failed node and hence can’t proceed to commit or abort the consensus.

4. Similarly, if coordinator fails during phase 2 we would want a node to take over and shepherd the protocol to completion. Another big issue is that if a participant node fails during commit phase the system is left to lurch in the dark because the coordinator doesn’t know whether the participant failed after committing or before committing. Hence coordinator can’t proactively decide whether the transaction is committed.

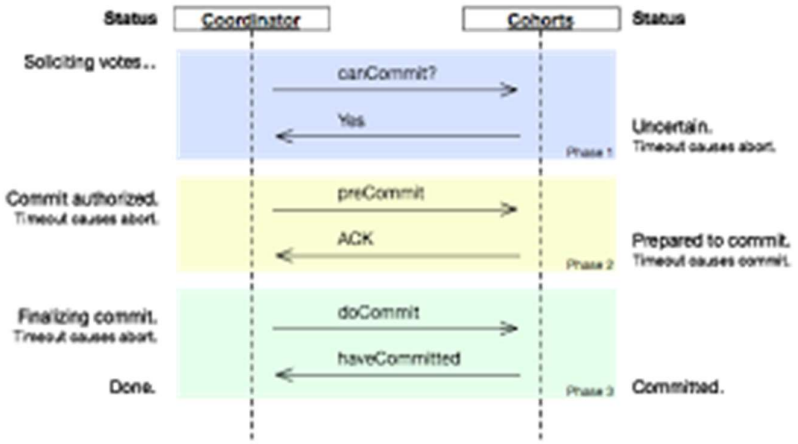
Summarizing one of the biggest disadvantages of two phase commit is that it’s blocking protocol. If a cohort sends an agreement message to the coordinator it holds the resources associated with the consensus till it receives the commit or abort message from the coordinator. The failure of coordinator will prevent the cohorts from recovering from a failure. The same reasoning applies to failure of a participant.

Three phase commit

This is an extension of two-phase commit wherein the commit phase is split into two phases as follows.

a. Prepare to commit, After unanimously receiving yes in the first phase of 2PC the coordinator asks all participants to prepare to commit. During this phase, all participants acquire locks etc, but they don’t actually commit.

b. If the coordinator receives yes from all participants during the prepare to commit phase then it asks all participants to commit.



The pre-commit phase introduced above helps us to recover from the case when a participant failure or both coordinator and participant node failure during commit phase. The recovery coordinator when it takes over after coordinator failure during phase2 of previous 2 pc the new pre-commit comes handy as follows. On querying participants, if it learns that some nodes are in commit phase then it assumes that previous coordinator before crashing has made the decision to commit. Hence it can shepherd the protocol to commit. Similarly, if a participant says that it doesn't receive prepare to commit, then the new coordinator can assume that previous coordinator failed even before it started the prepare to commit phase. Hence it can safely assume no other participant would have committed the changes and hence safely abort the transaction.

Code:

Server.java

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

public class Server{
    boolean closed = false, inputFromAll = false;
    List<ClientThread> thread;
    List<String> data;
    List<String> decision;

    Server() {
        thread = new ArrayList<ClientThread>();
        data = new ArrayList<String>();
        decision= new ArrayList<String>();
    }

    public static void main(String args[])
    {
        Socket clientSocket = null;
        ServerSocket serverSocket = null;
        int port_number = 1111;
        Server server = new Server();
        try
        {
            serverSocket = new ServerSocket(port_number);
        } catch (IOException e) {
            System.out.println(e);
        }
        while (!server.closed)
        {
            try {
                    clientSocket = serverSocket.accept();
                    ClientThread clientThread = new ClientThread(server,
clientSocket);
                    (server.thread).add(clientThread);
                    System.out.println("\nNow Total clients are : " +
(server.thread).size());
                    (server.data).add("NOT_SENT");
                    (server.decision).add("NOT_SENT");
                    clientThread.start();
            } catch (IOException e) { }
        }
        try {
```

```
            serverSocket.close();
        } catch (Exception e1) { }
    }
}

class ClientThread extends Thread
{
    DataInputStream is = null;
    String line;
    String destClient = "";
    String name;
    PrintStream os = null;
    Socket clientSocket = null;
    String clientIdentity;
    Server server;

    public ClientThread(Server server, Socket clientSocket)
    {
        this.clientSocket = clientSocket;
        this.server = server;
    }

    @SuppressWarnings("deprecation")
    public void run()
    {
        try {
            is = new DataInputStream(clientSocket.getInputStream());
            os = new PrintStream(clientSocket.getOutputStream());
            os.println("Enter your name.");
            name = is.readLine();
            clientIdentity = name;
            os.println("Welcome " + name + " to this 2 Phase Application.\nYou
will receive a vote Request now...");
            os.println("Send Ready or Not Ready after local transaction..");
            while (true)
            {
                line = is.readLine();
                if (line.equalsIgnoreCase("NOT READY"))
                {
                    System.out.println("\nFrom '" + clientIdentity
                            + "' : NOT READY\n\nSince NOT READY we will not
wait for inputs from other clients.");
                    System.out.println("\nAborted....");

                    for (int i = 0; i < (server.thread).size(); i++) {
                        ((server.thread).get(i)).os.println("GLOBAL_ABORT");
                        ((server.thread).get(i)).os.close();
                        ((server.thread).get(i)).is.close();
```

```
                    }
                    break;
                }
                if (line.equalsIgnoreCase("READY"))
                {
                    System.out.println("\nFrom '" + clientIdentity + "' :
READY");
                    if ((server.thread).contains(this))
                    {
                        (server.data).set((server.thread).indexOf(this),
"READY");
                        for (int j = 0; j < (server.data).size(); j++)
                        {
                            if
(!(((server.data).get(j)).equalsIgnoreCase("NOT_SENT")))
                            {
                                server.inputFromAll = true;
                                continue;
                            }
                            else{
                                server.inputFromAll = false;
                                System.out.println("\nWaiting for inputs from
other clients.");
                                break;
                            }
                        }
                        if (server.inputFromAll)
                        {
                            System.out.println("All Ready..");
                        }
                    }
                }

                os.println("VOTE_REQUEST\nPlease enter COMMIT or ABORT to
proceed : ");
                line = is.readLine();
                if (line.equalsIgnoreCase("ABORT"))
                {
                    System.out.println("\nFrom '" + clientIdentity
                            + "' : ABORT\n\nSince ABORT we will not wait for
inputs from other clients.");
                    System.out.println("\nAborted....");

                    for (int i = 0; i < (server.thread).size(); i++) {
                        ((server.thread).get(i)).os.println("GLOBAL_ABORT");
                        ((server.thread).get(i)).os.close();
                        ((server.thread).get(i)).is.close();
                    }
```

```
                    break;
                }
                if (line.equalsIgnoreCase("COMMIT")){
                    System.out.println("\nFrom '" + clientIdentity + "' :
COMMIT");
                    if ((server.thread).contains(this))
                    {
                        (server.decision).set((server.thread).indexOf(this),
"COMMIT");
                        for (int j = 0; j < (server.decision).size(); j++)
                        {
                            if
(!(((server.decision).get(j)).equalsIgnoreCase("NOT_SENT")))
                            {
                                server.inputFromAll = true;
                                continue;
                            }
                            else{
                                server.inputFromAll = false;
                                System.out.println("\nWaiting for inputs from
other clients.");
                                break;
                            }
                        }
                        if (server.inputFromAll){
                            System.out.println("\n\nCommited....");
                            for (int i = 0; i < (server.thread).size(); i++)
                            {
                                ((server.thread).get(i)).os.println("GLOBAL_CO
MMIT");
                                ((server.thread).get(i)).os.close();
                                ((server.thread).get(i)).is.close();
                            }
                            break;
                        }
                    }
                }
            }
            server.closed = true;
            clientSocket.close();
        } catch (IOException e) { }

    }
}
```

Client.java

```
import java.io.*;
import java.net.*;
public class Client implements Runnable
{
    static Socket clientSocket = null;
    static PrintStream os = null;
    static DataInputStream is = null;
    static BufferedReader inputLine = null;
    static boolean closed = false;
    public static void main(String[] args)
    {
        int port_number=1111;
        String host="localhost";
        try {
            clientSocket = new Socket(host, port_number);
            inputLine = new BufferedReader(new InputStreamReader(System.in));
            os = new PrintStream(clientSocket.getOutputStream());
            is = new DataInputStream(clientSocket.getInputStream());
        } catch (Exception e)
        {   System.out.println("Exception occurred : "+e.getMessage()); }


        if (clientSocket != null && os != null && is != null)
        {
            try
            {
                new Thread(new Client()).start();
                while (!closed)
                {
                    os.println(inputLine.readLine());
                }
                os.close();
                is.close();
                clientSocket.close();
            } catch (IOException e)
            {
                System.err.println("IOException:  " + e);
            }
        }
    }
    @SuppressWarnings("deprecation")
    public void run()     {
        String responseLine;
        try
        {
            while ((responseLine = is.readLine()) != null)
```

```
            {
                System.out.println("\n"+responseLine);
                if (responseLine.equalsIgnoreCase("GLOBAL_COMMIT")==true ||
responseLine.equalsIgnoreCase("GLOBAL_ABORT")==true )
                {
                    break;
                }
            }
            closed=true;
        }
        catch (IOException e)
        {
            System.err.println("IOException:  " + e);
        }
    }
}
```

Output:

Main Server with 3 client Rudra, Preet and Dhruv

And their activities are noted

```
PS C:\Users\rudra\OneDrive\Desktop\College\Sem 5\adbms\Exp> java Server

Now Total clients are : 1

From 'Rudra' : READY
All Ready..

Now Total clients are : 2

From 'Dhruv' : READY
All Ready..

Now Total clients are : 3

From 'Preet' : READY
All Ready..

From 'Preet' : COMMIT

Waiting for inputs from other clients.

From 'Preet' : COMMIT

Waiting for inputs from other clients.

From 'Rudra' : ABORT

Since ABORT we will not wait for inputs from other clients.

Aborted....
```

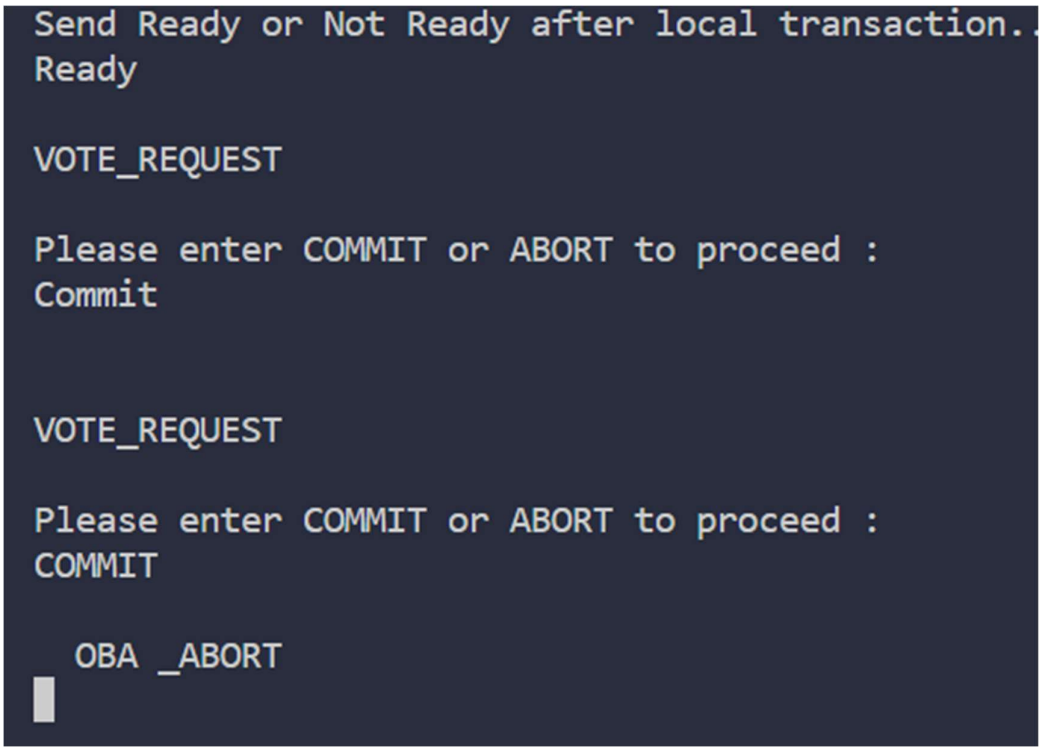
Client 1 Rudra

```
Enter your name.
Rudra

Welcome Rudra to this 2 Phase Application.

You will receive a vote Request now...

Send Ready or Not Ready after local transaction..
Ready

VOTE_REQUEST

Please enter COMMIT or ABORT to proceed :
Abort

GLOBAL_ABORT

PS C:\Users\rudra\OneDrive\Desktop\College\Sem 5\adbms\Exp>
```

Client 2 Dhruv

```
Enter your name.
Dhruv

Welcome Dhruv to this 2 Phase Application.

You will receive a vote Request now...

Send Ready or Not Ready after local transaction..
Ready

VOTE_REQUEST

Please enter COMMIT or ABORT to proceed :

  OBA _ABORT
```

Client 3 Preet



Conclusion: In summary, the 2PC protocol is a blocking Two-Phase commit protocol. The 3PC protocol is a non-blocking Three-Phase commit protocol. However, the 3PC protocol does not recover in the event the network in segmented situation. So a new way was suggested using the enhanced three-phase commit (E3PC) protocol to eliminate this issue. The E3PC protocol requires at least three round trips to complete. It needs a minimum of 3 round trip times that would have a long latency to complete each transaction.

Experiment 7

ADBMS

Ayush Jain 60004200132 B2

Aim: Query Execution on an XML database

Theory:

What is XML (Extensible Markup Language)?

XML (Extensible Markup Language) is used to describe data. The XML standard is a flexible way to create information formats and electronically share structured data via the public internet, as well as via corporate networks.

XML is a markup language based on Standard Generalized Markup Language (SGML) used for defining markup languages.

XML's primary function is to create formats for data that is used to encode information for documentation, database records, transactions and many other types of data. XML data may be used for creating different content types that are generated by building dissimilar types of content -- including web, print and mobile content -- that are based on the XML data.

Like Hypertext Markup Language (HTML), which is also based on the SGML standard, XML documents are stored as American Standard Code for Information Interchange (ASCII) files and can be edited using any text editor.

What is XML used for?

XML's primary function is to provide a "simple text-based format for representing structured information," according to the World Wide Web Consortium (W3C), the standards body for the web, including for the following:

- underlying data formats for applications such as those in Microsoft Office;
- technical documentation;
- configuration options for application software;
- books;

- transactions; and
- invoices.

XML enables sharing of structured information among and between the following:

- programs and programs;
- programs and people; and
- locally and across networks.

W3C defines the XML standard and recommends its use for web content. While XML and HTML are both based on the SGML platform, W3C has also defined the XHTML and XHTLM5 document formats that mirror, respectively, the HTML and HTML5 standards for web content.

How does XML work?

XML works by providing a predictable data format. XML is strict on formatting; if the formatting is off, programs that process or display the encoded data will return an error.

For an XML document to be considered *well-formed* -- that is, conforming to XML syntax and able to be read and understood by an XML parser -- it must be valid XML code. All XML documents consist of elements; an element acts as a container for data. The beginning and end of an element are identified by opening and closing tags, with other elements or plain data within.

XML works by providing properly formatted data that can be reliably processed by programs designed to handle XML inputs. For example, technical documentation may include a <warning> element similar to that shown in the following snippet of XML code:

XML entities

XML elements can also contain predefined entities, which are used for special reserved XML characters. Custom entities can be defined to insert a predefined string of characters for inclusion in an XML file.

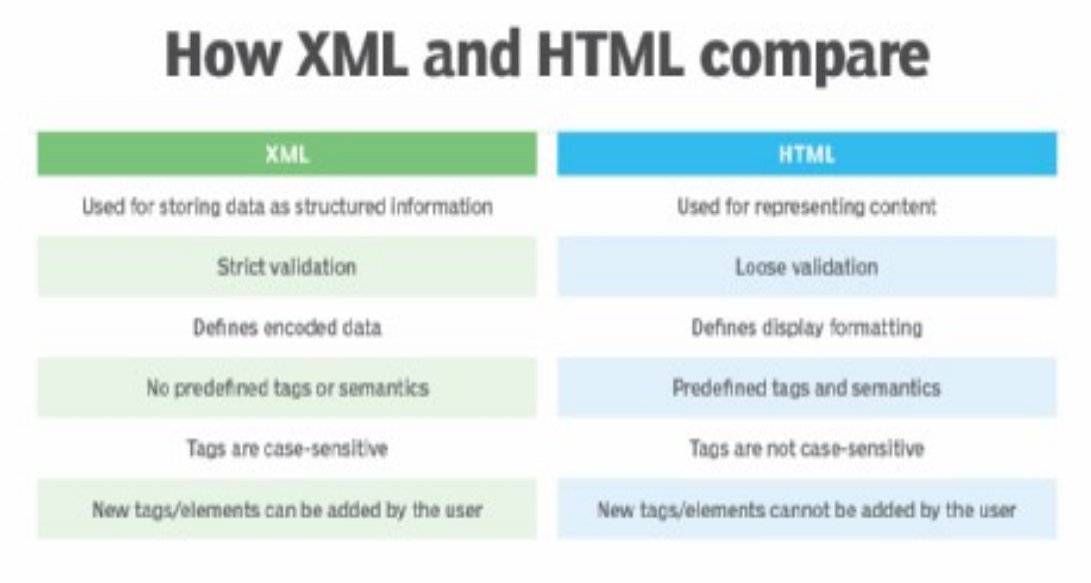
The five standard predefined XML entities are the following:

1. **< --** The less than symbol (<), also known as the *open angle bracket*, is normally used in XML to indicate the start of an XML tag. This entity is used when the open angle bracket is part of the content of the XML file.

2. > -- The greater than symbol (>), also known as the *close angle bracket*, is normally used in XML to indicate the end of an XML tag. This entity is used when the close angle bracket is part of the content of the XML file.
3. & -- The ASCII ampersand symbol (&) is reserved in XML for indicating the start of an XML entity. This entity is used when an ampersand occurs within an XML element.
4. " -- The ASCII double quote character (") is used in XML element tags to identify optional attribute values of the element. For example, an <emphasis> tag might include options for emphasizing some text, such as bold, italic or underline. This entity is used when a double quote character appears in the contents of an XML element.
5. ' -- The ASCII single quote character ('), also known as an *apostrophe*, is used in XML element tags to identify option attributes of the element. For example, an <emphasis> tag might include options for emphasizing some text, such as bold, italic or underline. This entity is used when a single quote or apostrophe appears in the contents of an XML element.

XML entities take the form of &name; where the entity name begins with the ampersand symbol and ends with a semicolon. Custom entities can be single characters or complex XML elements. For example, boilerplate language for technical documentation or legal contracts can be reduced to a single entity. However, when using entities, the XML author must ensure that inserting the entity into an XML file will produce well-formed XML data.

What are the differences between XML and HTML?



Code:

HTML

```
<html>

<head>
    <style>
        table,
        th,
        td {
            border: 1px solid black;
            border-collapse: collapse;
            margin: 5px;
        }

        th,
        td {
            padding: 5px;
        }

        input {
            margin-bottom: 5px;
        }
    </style>
```

```
</head>

<body>

    <button type="button" onclick="loadXMLDoc()">Show Employee
Information</button>
    <br><br>

    <table id="data-table"></table>

    <script>

        function loadXMLDoc() {
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200) {
                    myFunction(this);
                }
            };
            xmlhttp.open("GET", "Employee.xml", true);
            xmlhttp.send();
        }
        function myFunction(xml) {
            var i;
            var xmlDoc = xml.responseXML;
            var table =
"<tr><th>ID</th><th>Name</th><th>Department</th><th>Salary</th><th>Designation
</th><th>Age</th></tr>";
            var x = xmlDoc.getElementsByTagName("employee");

            for (i = 0; i < x.length; i++) {

                table += "<tr><td>" +
                    x[i].id +
                    "</td><td>" +
                    x[i].getElementsByTagName("name")[0].childNodes[0].nodeVal
ue +
                    "</td><td>" +
                    x[i].getElementsByTagName("department")[0].childNodes[0].n
odeValue +
                    "</td><td>" +
                    x[i].getElementsByTagName("salary")[0].childNodes[0].nodeV
alue +
                    "</td><td>" +
                    x[i].getElementsByTagName("designation")[0].childNodes[0].
nodeValue +
                    "</td><td>" +
```

```
                    x[i].getElementsByTagName("age")[0].childNodes[0].nodeValu
e +
                    "</td>";


            }
            document.getElementById("data-table").innerHTML = table;

        }

    </script>

</body>

</html>
```

XML

```xml
<?xml version="1.0"?>
<employees>
   <employee id="emp01">
      <name>Jhon</name>
      <department>Depertment 1</department>
      <salary>200000</salary>
      <designation>Senior Developer</designation>
      <age>39</age>
   </employee>
   <employee id="emp02">
      <name>Browm</name>
      <department>Depertment 3</department>
      <salary>50000</salary>
      <designation>Data Analyst</designation>
      <age>30</age>
   </employee>
   <employee id="emp03">
      <name>Jonathon</name>
      <department>Depertment 2</department>
      <salary>90000</salary>
      <designation>Junior Developer</designation>
      <age>39</age>
   </employee>
   <employee id="emp04">
      <name>Kevin</name>
      <department>Depertment 4</department>
      <salary>40000</salary>
      <designation>Graphic designer</designation>
      <age>43</age>
   </employee>
   <employee id="emp05">
      <name>Eliza</name>
      <department>Depertment 5</department>
      <salary>80000</salary>
      <designation>DataBase Administrator</designation>
      <age>39</age>
   </employee>
   <employee id="emp06">
      <name>Nancy</name>
      <department>Depertment 3</department>
      <salary>70000</salary>
      <designation>HR</designation>
      <age>39</age>
   </employee>

</employees>
```

Output:

Show Employee Information

ID	Name	Department	Salary	Designation	Age
emp01	Jhon	Depertment 1	200000	Senior Developer	39
emp02	Browm	Depertment 3	50000	Data Analyst	30
emp03	Jonathon	Depertment 2	90000	Junior Developer	39
emp04	Kevin	Depertment 4	40000	Graphic designer	43
emp05	Eliza	Depertment 5	80000	DataBase Administrator	39
emp06	Nancy	Depertment 3	70000	HR	39

Conclusion: Thus, we have studied how to represent XML data into HTML

Experiment 8

ADBMS

Ayush Jain 60004200132 B2

Aim: Data handling using JSON (eg. Display user information from JSON file downloaded from Mobile)

Theory:

Android provides several ways of dealing with app data within the system or local storage. We are going to be dealing with app-specific storage of data in directory available in Internal or External Storage of system.

App-specific Storage

- Internal Storage : Sensitive data, No other application access it.
- External Storage : Other application can access it like Images.

What we are going to do ?

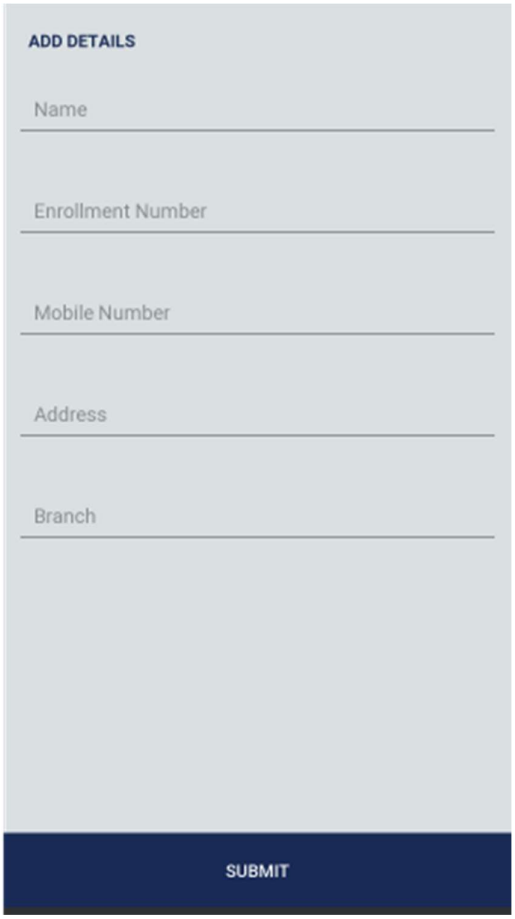
We will generate a JSON file, which will be stored in Internal storage of application. From android application user will add(**WRITE**) data, which will be converted into JSON format(JSON Object) and then stored in JSON file.

We will access(**READ**) the data from JSON file and converted into app usable format like string, arrays etc.

We will also **UPDATE** the data from JSON file and save it back to JSON file.

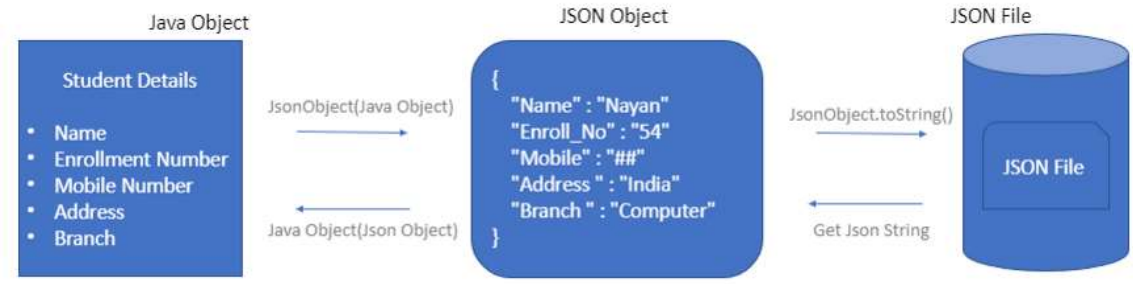
We will also perform the **DELETION** operation on JSON file data/Objects.

1. **Write Data into JSON File :-**



Get Data From Application

Data will be taken in terms of Java Object and transferred to JSON File.



Flow of Data to JSON parsing

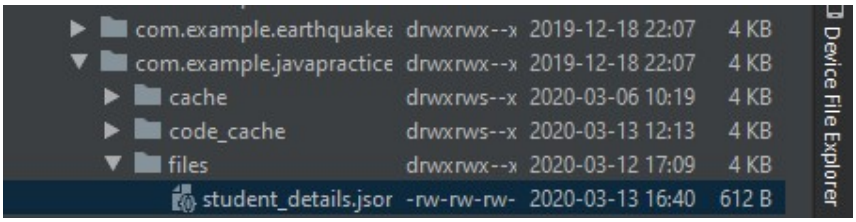
Java-object will be passed to **JsonObject**, which will convert the java object into JsonObject which means the value is now associated with a key because JSON works as Key-Value pairs.

```
JSONObject jsonObject = new JSONObject();
jsonObject.put("Name", Name);
jsonObject.put("Enroll_No", Enrollment Number);
jsonObject.put("Mobile", Mobile);
jsonObject.put("Address", Address);
jsonObject.put("Branch", Branch);
return jsonObject;
```

Now we will store this JsonObject to our JSON File available at Internal Storage, For this we need to **define the path** and then we will store the JSON object as a String into .Json file.

```
// Convert JsonObject to String Format
String userString = JsonObject.toString();// Define the
File Path and its Name
File file = new File(context.getFilesDir(),FILE_NAME);
FileWriter fileWriter = new FileWriter(file);
BufferedWriter bufferedWriter = new
BufferedWriter(fileWriter);
bufferedWriter.write(userString);
bufferedWriter.close();
```

At this point, Data has entered into JSON file. How can I see where the data is transferred into Android Studio → **Device File Explorer**.



Contect.getFilesDir will store into ‘files’ folder

JSON File will have data stored Like this :-

```
{
"Name":"Ram Varma",
"Enroll_no":"160760120546",
"Mobile":"8989898989",
"Address":"Sahyog Socirty",
"Branch":"EC"
}
```

2. **Read Data From JSON FIle** :-

Now we have to access the data which is available in JSON file.

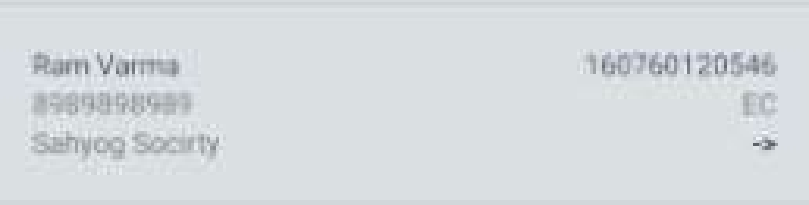
```
File file = new
File(context.getFilesDir(),FILE_NAME);FileReader
fileReader = new FileReader(file);
BufferedReader bufferedReader = new
BufferedReader(fileReader);
StringBuilder stringBuilder = new StringBuilder();
String line = bufferedReader.readLine();
while (line != null){
    stringBuilder.append(line).append("\n");
    line = bufferedReader.readLine();
}
bufferedReader.close();// This responce will have Json
Format String
String responce = stringBuilder.toString();
```

This responce is available in String Json Format,but we have to access it in Java Object form so that we can apply it wherever we want in our application.

So we have to get the data available in responce(String) using **Key and assign those value to our Java Object.**

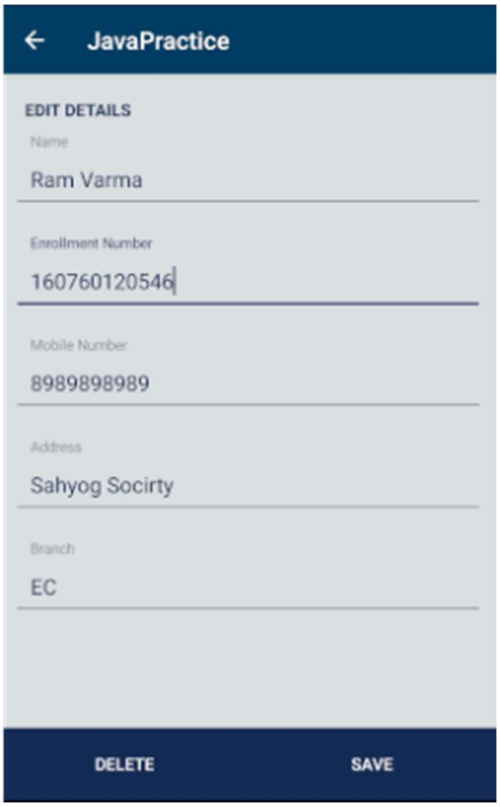
```
JSONObject jsonObject  = new
JSONObject(responce);//Java Object
JavaObject javaObject =
new JavaObject(jsonObject.get("name").toString(),
        jsonObject.get("enroll_no").toString(),
        jsonObject.get("mobile").toString(),
        jsonObject.get("address").toString(),
        jsonObject.get("branch").toString());return
javaObject;
```

Now We can access this javaObject and have value which was stored in JSON file.



javaObject data displayed on List

3. **Update and Delete the Data to JSON FIle :-**



Edit the given data and **save** it to JSON File

Now This updated Java Object again perform **WRITE** operation(repeat step 1) on JSON file and **Edited data will be displayed** when we read(repeat step 2) the JSON object from file.

Code:

```
package com.codinginflow.jsonparsingexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
```

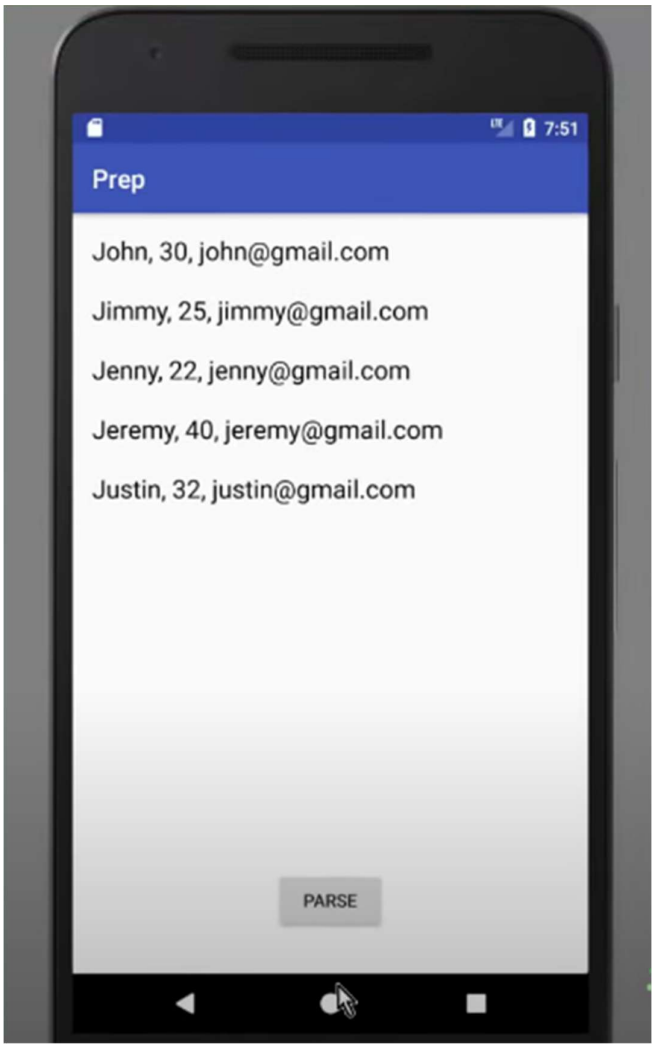
```
import com.android.volley.toolbox.Volley;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class MainActivity extends AppCompatActivity {
    private TextView mTextViewResult;
    private RequestQueue mQueue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mTextViewResult = findViewById(R.id.text_view_result);
        Button buttonParse = findViewById(R.id.button_parse);

        mQueue = Volley.newRequestQueue(this);

        buttonParse.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                jsonParse();
            }
        });
    }

    private void jsonParse() {

        String url = "https://api.myjson.com/bins/kp9wz";

        JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET,
url, null,
                new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject response) {
                        try {
                            JSONArray jsonArray =
response.getJSONArray("employees");

                            for (int i = 0; i < jsonArray.length(); i++) {
                                JSONObject employee =
jsonArray.getJSONObject(i);

                                String firstName =
employee.getString("firstname");
                                int age = employee.getInt("age");
```

```
                                String mail = employee.getString("mail");

                                mTextViewResult.append(firstName + ", " +
String.valueOf(age) + ", " + mail + "\n\n");
                            }
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                error.printStackTrace();
            }
        });

        mQueue.add(request);
    }
}
```

Json

```
{
  "employees": [
    {
      "firstname": "John",
      "age": 30,
      "mail": "john@gmail.com"
    },
    {
      "firstname": "Jimmy",
      "age": 25,
      "mail": "jimmy@gmail.com"
    },
    {
      "firstname": "Jenny",
      "age": 22,
      "mail": "jenny@gmail.com"
    },
    {
      "firstname": "Jeremy",
      "age": 40,
      "mail": "jeremy@gmail.com"
    },
    {
      "firstname": "Justin",
      "age": 32,
      "mail": "justin@gmail.com"
    }
  ]
}
```

Output:



Conclusion: After serializing and deserialization JSON data obtained from google it can be concluded that javascript methods like stringify and parse can be use to handle JSON data: After serializing and deserialization JSON data obtained from google it can be concluded that javascript methods like stringify and parse can be use to handle JSON data

Experiment 9

ADBMS

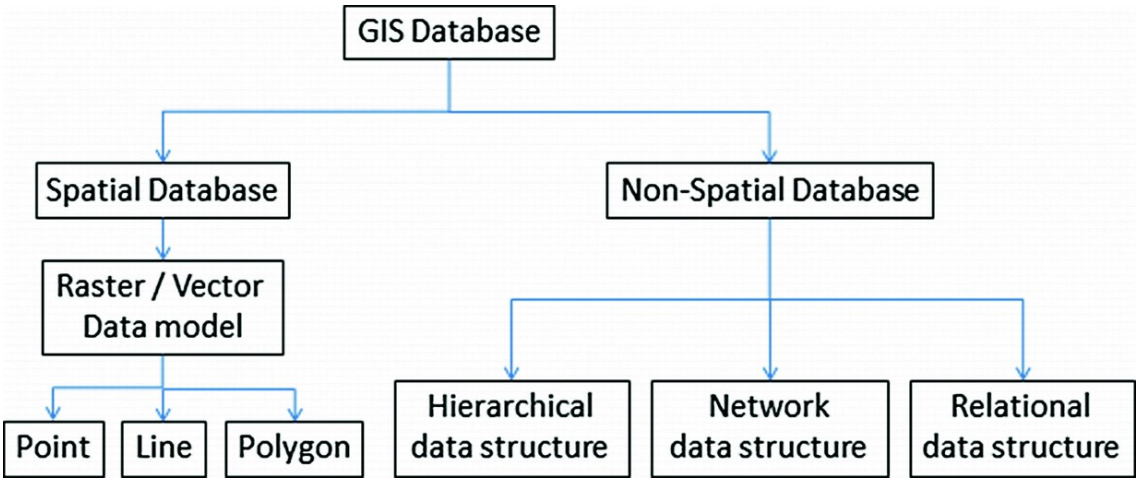
Ayush Jain 60004200132 B2

Aim: Processing of Spatial and temporal data

Theory:

A. Spatial Database:

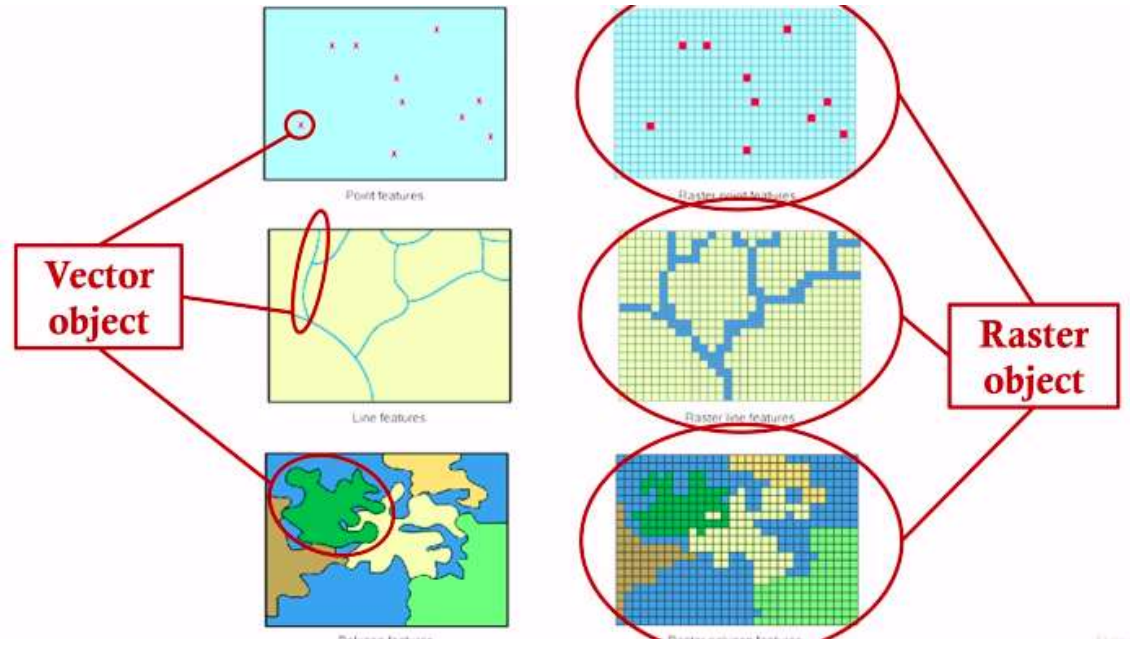
A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space. These data are often associated with geographic locations and features, or constructed features like cities. Data on spatial databases are stored as coordinates, points, lines, polygons and topology. Some spatial databases handle more complex data like three-dimensional objects, topological coverage and linear networks.Aside from the indexes, spatial databases also offer spatial data types in their data model and query language. These databases require special kinds of data types to provide a fundamental abstraction and model the structure of the geometric objects with their corresponding relationships and operations in the spatial environment. Without these kind of data types, the system would not be able to support the kind of modeling a spatial database offers.



- **Vector**

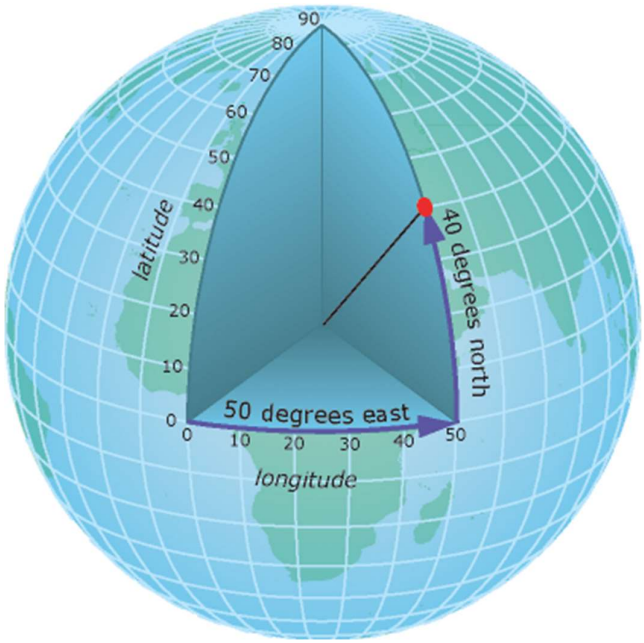
Vector data is graphical representations of the real world. There are three main types of vector data: points, lines, and polygons. Connecting points create lines, and connecting lines that create an enclosed area create polygons. Vectors present generalizations of objects or features on the Earth's surface. Vector data and the file format known as shapefiles (.shp) are sometimes used interchangeably. Vector data is most often stored in .shp files.

- **Raster**
  Raster data is data that is presented in a grid of pixels. Each pixel within a raster has a value, whether it be a colour or unit of measurement. This communicates information about the element in question. Rasters typically refer to imagery. However, in the spatial world, this may specifically refer to orthoimagery which are photos taken from satellites or other aerial devices. Raster data quality varies depending on resolution and your task at hand.



Secondary Spatial Data
Geographic Coordinate System
To identify exact locations on the surface of the Earth, a geographic coordinate system is used. Normally, an x and y-axis are used in mathematical systems, but in geography, the axes are referred to as lines of latitude (horizontal lines that run north-south) and longitude (vertical lines that run east-west). Each axis represents the angle at which that line is oriented with respect to the center of the Earth, and so the units are measured in degrees (°). There is more on projections and coordinate systems below!

Georeferencing and Geocoding



Georeferencing and geocoding are different. However, they are similar processes. Both involve fitting data to the appropriate coordinates of the real world. Georeferencing is the process of assigning coordinates to vectors or rasters so they can be oriented accurately on a model of the Earth's surface. The data used in geocoding are addresses and location descriptors (city, country, etc.). Each of these locations is given the coordinates of reference for that location on the surface of the Earth.

B. Temporal data:

A Temporal Database is a database with **built-in support for handling time sensitive data**. Usually, databases store information only about current state, and not about past states. For example in a employee database if the address or salary of a particular person changes, the database gets updated, the old value is no longer there. However for many applications, it is important to maintain the past or historical values and the time at which the data was updated. That is, the knowledge of evolution is required. That is where temporal databases are useful. It stores information about the past, present and future. Any data that is time dependent is called the temporal data and these are stored in temporal databases.

Temporal Databases store information about states of the real world across time. Temporal Database is a database with built-in support for handling data involving time. It stores information relating to past, present and future time of all events.

Temporal data types
Temporal data types that are supported by MySQL are as follows:

- TIME
- DATE
- DATETIME
- TIMESTAMP
- YEAR

TIME
The TIME data type can express the time of day or duration, and has a range of '-838:59:59' to '838:59:59'. MySQL displays TIME values in 'HH:MM:SS' format, but TIME columns can be assigned using either strings or numbers.

DATE
The supported range is '100001-01' to '9999-12-31'
MySQL retrieves DATE values in 'YYYY-MM-DD' format, but DATE columns can be assigned using either strings or numbers. For example, the date January 21,2001 would look like: '2001-01-21'

DATETIME
This data type allows date and time combinations. The supported range is '1000-01-01' to '9999-12-31 23:59:59'.
MySQL, retrieves DATETIME values in 'YYYY-MM-DD HH:MM:SS' format.

TIMESTAMP
This data type can be used to express datetime values ranging from '1970-01-01 00:00:00' to partway through the year 2037. The TIMESTAMP returns a string in the format 'YYYY-MM-DD HH:MM:SS' with a display width fixed at 19 characters. To obtain the values as a number, add +0 to the timestamp column. A TIMESTAMP column is useful for recording the date and time of an INSERT or UPDATE operation.

YEAR
The YEAR data type is assigned using the syntax:
YEAR [(2|4)]
where the year can be in a two-digit o four-digit format.
The default is the four-digit format. In four-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MySQL retrieves YEAR values in YYYY format.

The following terms stand for:

- YYYY – year
- MM – month
- DD – day of the month
- hh – hour
- mm – minute
- ss – second
- Reservation Systems: Date and time of all reservations is important.

Queries executed in Mongo Shell-

1.Find location of a coordinate in MongoDB

```
world_data> db.wonders.findOne();
{
  _id: ObjectId("638f3303d74932f1d15321c4"),
  name: 'Eiffel Tower',
  position: { type: 'Point', coordinates: [ 2.294481, 48.85837 ] }
}
world_data> db.wonders.createIndex({ position: "2dsphere" });
position_2dsphere
```

```
world_data> db.wonders.find({ position: { $nearSphere: { $geometry: { type: "Point", coordinates: [ 2.2, 48.8 ] }, $maxDistance: 10000
 } } });
[
  {
    _id: ObjectId("638f3303d74932f1d15321c4"),
    name: 'Eiffel Tower',
    position: { type: 'Point', coordinates: [ 2.294481, 48.85837 ] }
  }
]
```

```
world_data> db.wonders.findOne({ position: { $nearSphere: { $geometry: { type: "Point", coordinates: [ 103, 13 ] } } } });
{
  _id: ObjectId("638f3303d74932f1d15321cd"),
  name: 'Angkor Wat',
  position: { type: 'Point', coordinates: [ 103.867424, 13.412938 ] }
}
```

2.Find locations within a rectangular region

```
world_data> db.wonders.find({ position: { $geoWithin: { $box: [ [66.94, 35.121], [89.351, 5.675]] } } });
[
  {
    _id: ObjectId("638f3303d74932f1d15321c5"),
    name: 'Taj Mahal',
    position: { type: 'Point', coordinates: [ 78.042158, 27.175074 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321c8"),
    name: 'Mount Everest Base Camp',
    position: { type: 'Point', coordinates: [ 86.925027, 28.008735 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321d5"),
    name: 'Mount Everest',
    position: { type: 'Point', coordinates: [ 86.922623, 27.987851 ] }
  }
]
```

3.Find locations within a specific circular region

```
world_data> db.wonders.find({ position: { $geoWithin: { $center: [ [42.23, 32.849], 40] } } });
[
  {
    _id: ObjectId("638f3303d74932f1d15321c5"),
    name: 'Taj Mahal',
    position: { type: 'Point', coordinates: [ 78.042158, 27.175074 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321c6"),
    name: 'Mount Kilimanjaro',
    position: { type: 'Point', coordinates: [ 37.35378, -3.067768 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321c7"),
    name: 'Burj Khalifa',
    position: { type: 'Point', coordinates: [ 55.270764, 25.197197 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321c9"),
    name: 'Hagia Sophia',
    position: { type: 'Point', coordinates: [ 28.97953, 41.00527 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321ce"),
    name: 'Petra',
    position: { type: 'Point', coordinates: [ 35.441954, 30.328611 ] }
  },
  {
    _id: ObjectId("638f3303d74932f1d15321d0"),
    name: "St. Basil's Cathedral",
    position: { type: 'Point', coordinates: [ 37.617634, 55.754044 ] }
  }
]
```

Conclusion: Hence, we learned about the concepts of spatial and temporal data. We also implemented Geospatial query in MongoDB by using 2d Sphere index

Experiment 10

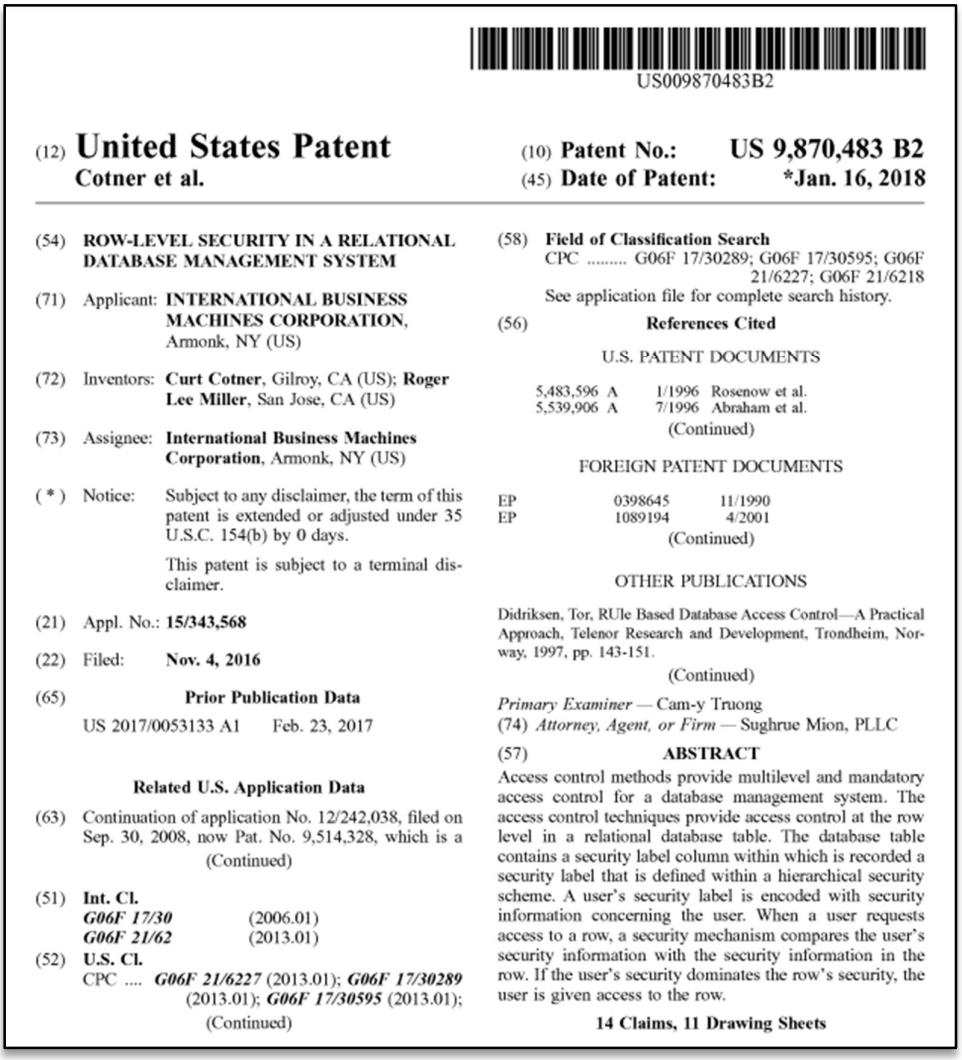
ADBMS

Ayush Jain 60004200132 B2

Aim: Case Study

Case Study on Database security on Row-level security on relational Database Management System

- Patent Information



US009870483B2

(12) **United States Patent**
Cotner et al.

(10) **Patent No.: US 9,870,483 B2**
(45) **Date of Patent: *Jan. 16, 2018**

(54) **ROW-LEVEL SECURITY IN A RELATIONAL DATABASE MANAGEMENT SYSTEM**

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**, Armonk, NY (US)

(72) Inventors: **Curt Cotner**, Gilroy, CA (US); **Roger Lee Miller**, San Jose, CA (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/343,568**

(22) Filed: **Nov. 4, 2016**

(65) **Prior Publication Data**

US 2017/0053133 A1 Feb. 23, 2017

Related U.S. Application Data

(63) Continuation of application No. 12/242,038, filed on Sep. 30, 2008, now Pat. No. 9,514,328, which is a (Continued)

(51) **Int. Cl.**
G06F 17/30 (2006.01)
G06F 21/62 (2013.01)

(52) **U.S. Cl.**
CPC ***G06F 21/6227*** (2013.01); ***G06F 17/30289*** (2013.01); ***G06F 17/30595*** (2013.01);
(Continued)

(58) **Field of Classification Search**
CPC G06F 17/30289; G06F 17/30595; G06F 21/6227; G06F 21/6218
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,483,596 A 1/1996 Rosenow et al.
5,539,906 A 7/1996 Abraham et al.
(Continued)

FOREIGN PATENT DOCUMENTS

EP 0398645 11/1990
EP 1089194 4/2001
(Continued)

OTHER PUBLICATIONS

Didriksen, Tor, RUle Based Database Access Control—A Practical Approach, Telenor Research and Development, Trondheim, Norway, 1997, pp. 143-151.
(Continued)

Primary Examiner — Cam-y Truong
(74) *Attorney, Agent, or Firm* — Sughrue Mion, PLLC

(57) **ABSTRACT**

Access control methods provide multilevel and mandatory access control for a database management system. The access control techniques provide access control at the row level in a relational database table. The database table contains a security label column within which is recorded a security label that is defined within a hierarchical security scheme. A user's security label is encoded with security information concerning the user. When a user requests access to a row, a security mechanism compares the user's security information with the security information in the row. If the user's security dominates the row's security, the user is given access to the row.

14 Claims, 11 Drawing Sheets

- **Introduction**

With the growth of the World-Wide Web ("web") and e-business solutions, database security and privacy are becoming increasingly critical. Hosting a web site on a server, referred to as web hosting, is another trend that magnifies the importance of database security. The web server includes a relational database storing a customer's data in many related tables. A web hosting company is motivated to store data from many customers in a single database management system to minimize its expenses. However, an increasing number of customers need a higher degree of security than is available with database management systems conventionally used by hosting companies, especially when the database management system is used to host more than one customer's web site and data.

Some customers need mandatory access controls in which all access to a data item, such as a database row, is controlled. Many customers also need to use a hierarchical security scheme that simultaneously supports multiple levels of access control. These concepts of mandatory access controls and hierarchical security schemes are well known. They are described, for example, in a Department of Defense standard DoD 5200.28-STD, *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985, which is incorporated by reference herein.

Conventional relational databases, such as the database described in U.S. Pat. No. 5,751,949 to Thomson et al., provide security based on tables and views of those tables. Views can be used to limit access to selected rows and columns within one or more database tables. For example, in Thomson et al., views are used to join data tables with a security table containing user authorization information. Certain users, however, such as system administrators can bypass views and access tables directly, thereby circumventing the access control provided by views. Also, it is often cumbersome for the database administrator and application programmer to construct views that have the desired level of granularity. Although views can be effective for read-only access, views are more difficult to define for updating, inserting and deleting.

Triggers, database constraints and stored procedures are often needed for update controls.

Although many applications need row-level security within a relational database so that individual user access can be restricted to a specific set of rows, there is a need to make the security control mandatory. With mandatory access control, users, application programmers and database administrators are unable to bypass the row-level security mechanism.

- **Problems with conventional Database Management System for Security**

Certain conventional database management systems (DBMS) provide some capabilities to limit access to rows within the database. However, those conventional systems rely on a database administrator to create views that restrict access to the desired rows. The application programmer must then use those special views to enforce the security controls. Often, the application programmer must populate session variables with values that the views use to control access to the data rows. Although such conventional systems do allow a programmer to control access to the data, those conventional systems suffer from several disadvantages.

For example, conventional DBMSs use views to control access to a database. Using views to control access to the database is cumbersome for the database administrator and the application programmer to implement. For example, it is common to have to create separate views for each security level (e.g., a TOP_SECRET VIEW, a SECRET VIEW, etc.) in the security scheme. Using views to control access is also error prone, since it is easy to implement

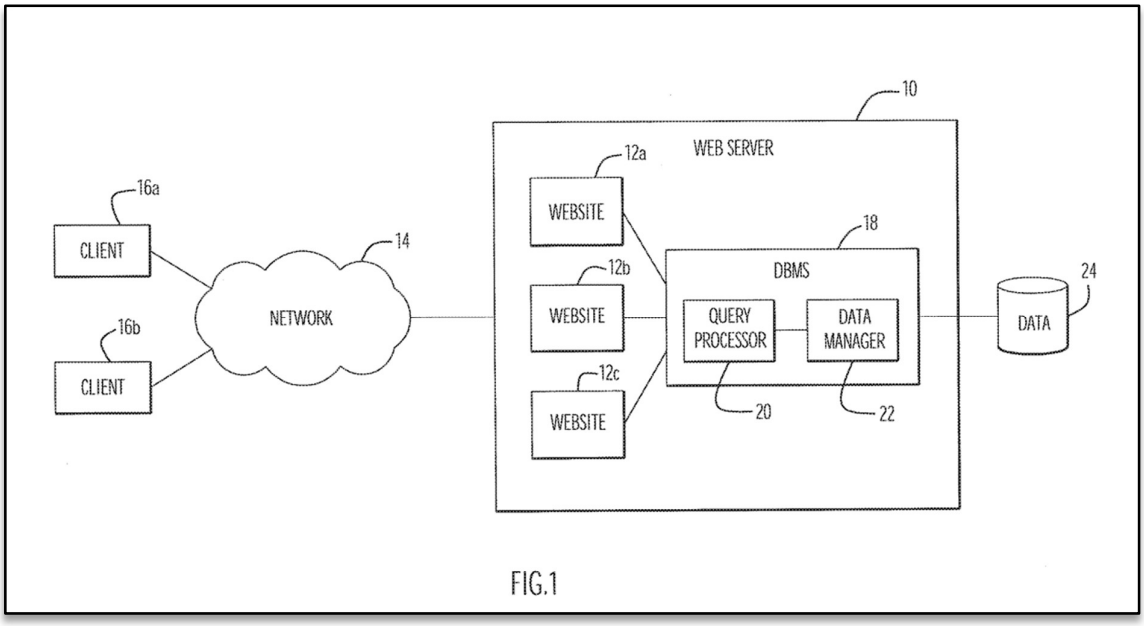


FIG.1

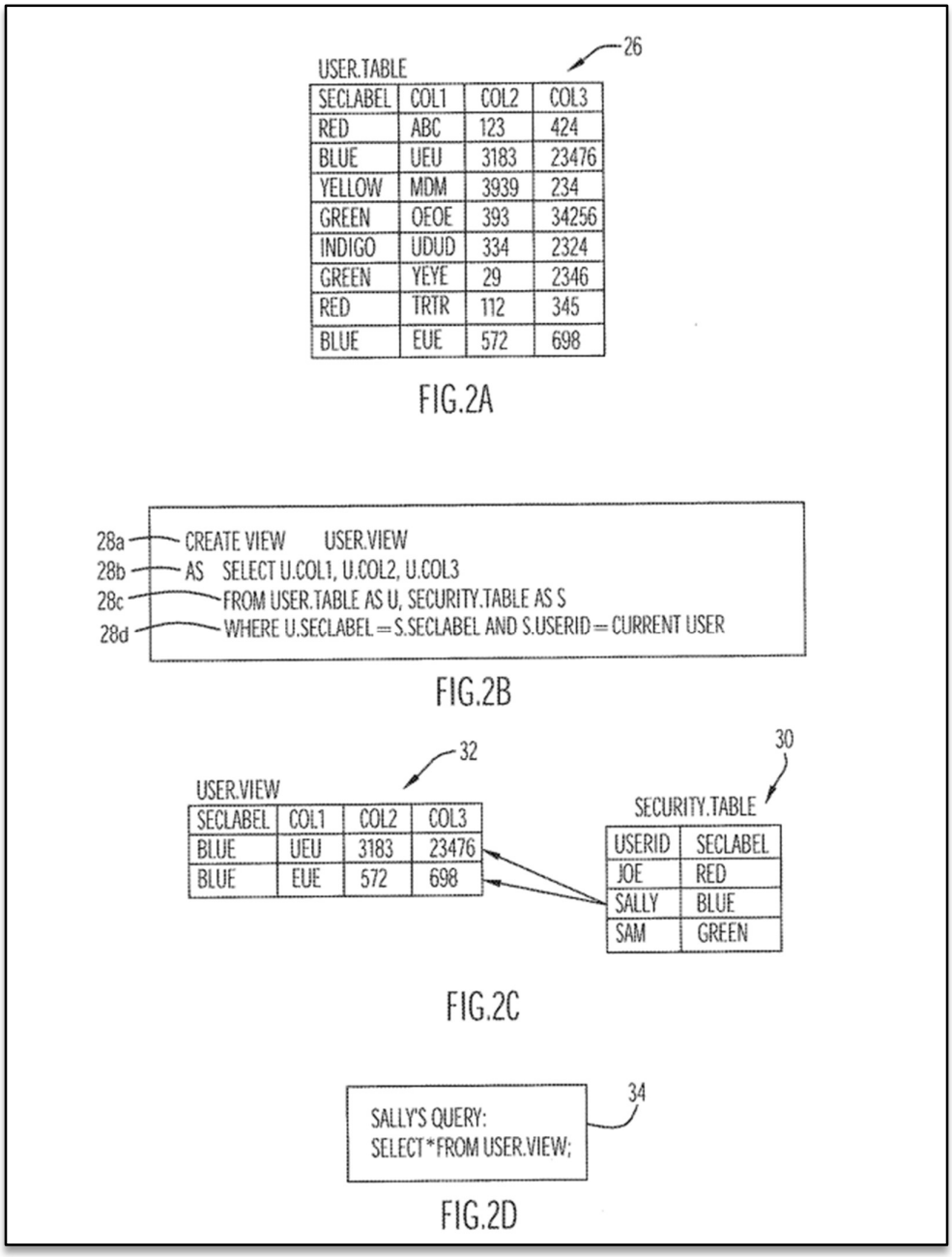
the views incorrectly and inadvertently permit access to the wrong data rows. Also, some security schemes are difficult to express as additional predicates on the view or user query. There is a need to automatically enforce access to the data such that no changes are required to the user application logic or views. The security policy achieved using views is only discretionary rather than mandatory. A person with database administrator authority is able to view the data in the database without using the special views that implement the security mechanism. There is a need to provide mandatory security controls that prevent unauthorized access by end users, application programmers, and database administrators. In order to limit the universe of users with access

to data in the database, the system security administrator should be the only individual with unbridled access to the data.

FIG. 1 shows a system level diagram of a conventional database application, involving a web server 10 hosting a plurality of web sites, namely web site 12 a, web site 12 b and web site 12 c. The web server is connected to a data communications network 14, such as the Internet, that provides connectivity with a plurality of clients, such as clients 16 a and 16 b. The web server includes a single DBMS 18 that services each of the web sites. The DBMS manages data used by the plurality of web sites. Since many web sites are hosted by the web server, yet are served by a single DBMS, security must be provided to prevent unauthorized access of data from one web site by a person using another web site hosted by web server 10. In conventional web servers, the DBMS includes a query processor 20 and a data manager 22. The DBMS 18 is connected to a data repository 24 that holds data managed by the DBMS.

The query processor 20 processes requests containing queries received at a web site from a client. For example, a typical query might be a Structured Query Language (SQL) query that is received at a web site. The SQL query is passed to the query processor 20 for parsing and execution by the DBMS. Based on the query, the query processor 20 controls the data manager 22 to interact with the data repository 24 to handle the appropriate data satisfying the query.

FIG. 2A shows an example of a conventional user table (USER.TABLE) 26 held within the DBMS shown in FIG. 1. The table contains various columns of data, labeled Col1, Col2, and Col3 shown in FIG. 2A. The user table also includes a security label (SECLABEL) column. Each row is associated with a specific security label. Here, the security labels are the names of various colors such as red, blue, yellow, green, for example. Each color name



represents a particular set of security privileges associated with the user table row. For example, a security label of "red" may have one set of access privileges associated with that label, whereas another security label, such as security label blue, is associated with another set of privileges.

Access to USER.TABLE, in a conventional DBMS, is controlled by a database view. A system administrator creates a view of the related tables within the database in order to restrict access based on a user's security label.

The system administrator creates a table by using, for example, SQL statements as shown in FIG. 2B. Here a system administrator creates a view called USER.VIEW 28 a in which the view is selected from three columns, namely, U.COL1, U.COL2, U.COL3, as shown in line 28 b of FIG. 2. Those columns are selected from the USER.TABLE shown in FIG. 2A. A security table (SECURITY.TABLE) 30 relates a user ID (USERID) with a security label (SECLABEL) such as security labels “red”, “blue”, or “green.” This is shown in line 28 c of FIG. 2B. Line 28 d of FIG. 2B requires that the user security label equals the security label defined in the security table, and that the user ID in the security table equals the current user. This limits access to only those rows in USER.TABLE of FIG. 2A that have a security label equal to the current user's security label. Although this conventional access control scheme provides a degree of access control, it does not support a hierarchical security scheme.

FIG. 2C shows a relationship between a security table 30 and the resulting view when the SQL statements of FIG. 2B are applied to USER.VIEW 32. If the view of FIG. 2B is applied for user “SALLY” to access the USER.TABLE 26, the result is shown in FIG. 2C. Here, in FIG. 2C, when the user SALLY requests access to the USER.TABLE, the view of FIG. 2B limits SALLY's view of the table 26 to the USER.VIEW 32 shown in FIG. 2C. When the view is applied to table 26, the USER.TABLE 26 is joined with SECURITY.TABLE 30 to produce USER.VIEW 32.

FIG. 2D shows an example query 34 requested by the user SALLY. SALLY's query includes a select SQL clause selecting all rows from USER.VIEW. The conventional DBMS system operates by applying the view shown in FIG. 2B to the USER.TABLE 26 shown in FIG. 2A. As shown in the SECURITY.TABLE 30 of FIG. 2C, SALLY's security label is “blue”, and accordingly, the resulting user view 32 includes only those rows of USER.TABLE 26 having a security label equal to the security label “blue”. In this manner, a conventional DBMS limits the user's access to only certain rows. However, this conventional access control technique does not support hierarchical security schemes, and it requires the use of views to limit access.

- **Row-Level Security solution to overcome the problem of security with conventional database management system**

1. Each end user of the database management system is assigned a SECURITY_LABEL. That label identifies a security level for the user within a multilevel security scheme and defines certain privileges for accessing data in the database. The security label also identifies security categories within that security level that the user is allowed to access. An example of a security category is a software development project on which the user is authorized to work. For example, a given user might be allowed to view data designated by certain security levels, such as the security levels: TOP SECRET, SECRET, and UNCLASSIFIED. That user also can be permitted to access data that pertains to certain categories, such as, for example, projects ABC, DEF, and XYZ. The value stored in the security label is encoded in a manner that expresses the security level and category information to the security system. An example of such an encoding is the label SECRETABC, where "SECRET" specifies the security level and "ABC" specify security categories A, B and C, which could be identifiers of projects on which the user is assigned to work. The user's security label can be determined using different techniques. For example, the user's security label can be determined with a lookup using the relational DBMS catalog; by making a security call to an external security manager; or by a call to a trusted installation exit routine, for example. It will be understood that other techniques to determine the user's security label can be employed.

2. Each row within a secure table is associated with a security label, which can be a column within that security table. For example, that column can have a predetermined name (e.g. SECURITY_LABEL) or it can be identified through an SQL clause when the table is defined (e.g. AS SECURITY LABEL clause on the CREATE TABLE column definition). It will be understood that other techniques can be used to associate a security label with a row. The SECURITY_LABEL column in the row identifies the security level of the data contained in the row, as well as security categories to which the row applies. For example, the row might contain data having a security level of "SECRET", that pertains to projects ABC and XYZ (security categories). The value stored in the SECURITY_LABEL is encoded in a manner that expresses the security level and category information to the security system.

3. A mandatory security enforcement mechanism controls read access to the secure data rows. That mechanism is activated automatically when a relational database table is known to include a SECURITY_LABEL column. This read security enforcement mechanism compares the user's security label to

the row's security label to determine whether access should be allowed. Read access is allowed only if the user's security dominates the row's security, in which both of the following conditions are true:
a. The security level indicated by the user's security label is greater than or equal to the security level indicated by the row's security label.
b. The security categories associated with the row's security label are a proper subset of the security categories associated with the user's security label.
Write access is controlled separately, so that users can force compliance with the general rule of not reading from rows having higher security levels than the user or writing to rows having lower security levels than the user.
The read and write access security mechanisms can use several various techniques to enforce the access scheme, such as by using a lookup with the relational DBMS catalog; using a security call to an external security manager; or using a call to a trusted installation exit routine, for example.

4. A mandatory security enforcement mechanism controls write access to the secure data rows. This mechanism is activated automatically when a relational database table is known to include a specific column name. The mechanism determines which security label is recorded in the updated data rows to be written in the database. This write access security mechanism forces each of those updated rows to contain one of the following possible values.
a. A security label that is the same as the user's security label is used as the security label for the updated row.
b. If a user is specially authorized, that user is allowed to update rows using a row security label that has a lower level than the user's current security label indicates. The write access security mechanism verifies that the user's security dominates the row's security, such that all of the following conditions are true, before allowing the row to be updated.

- i) The user is specially authorized to write data in a row having a security label designated for a security level lower than a security level associated with the user's security label.
- ii) The security label specified for the row has a security level that is less than or equal to the security level associated with the user's security label.
- iii) The security categories for the security label specified for the row are a proper subset of the security categories associated with the user's security label. That is, all the security categories associated with the row's label are also associated with the user's security label.

The write access security mechanism uses several different techniques to enforce the access scheme. For example, it can perform a lookup with the

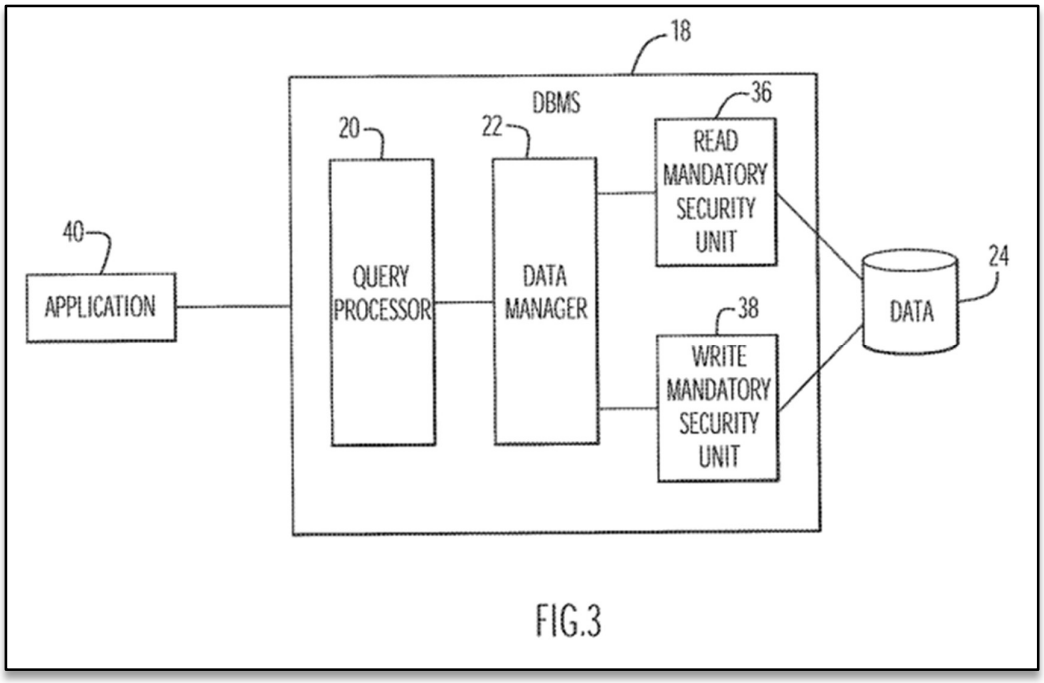


FIG.3

relational DBMS catalog; make a security call to an external security manager; or make a call to a trusted installation exit routine.

Illustration Using Figure

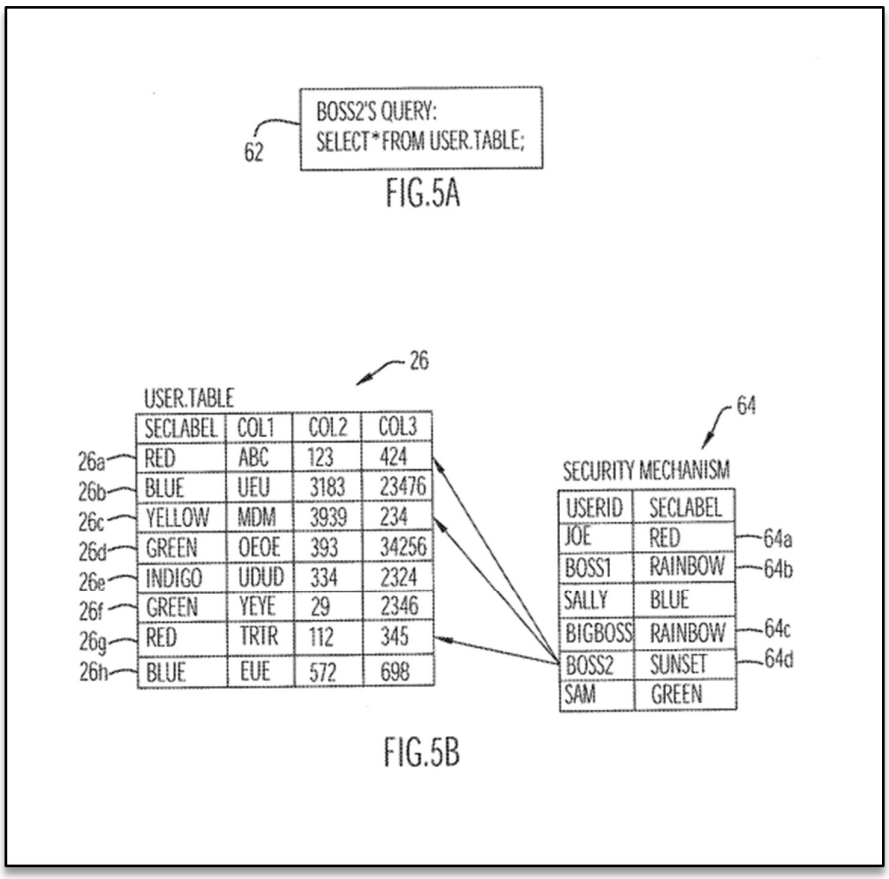
A DBMS that supports mandatory access control at the row level is illustrated in FIG. 3. Here, the DBMS **18**, in addition to a query manager **20** and a data manager **22**, also includes a read mandatory security unit **36** and a write mandatory security unit **38**. An exemplary embodiment of DBMS **18** is IBM's DB2® for the z/OS® operating system. Each mandatory security unit is coupled to the data manager **22** and to the data storage unit **24**. Both of the mandatory security units operate to control access by a user, or application program **40** such as a web site, to the data according to a hierarchical security scheme.

By placing the mandatory security units between the data manager **22** and the data repository **24**, mandatory access control is achieved. If the data manager **22** attempts to read a row of data from the data storage unit **24**, the request is directed through the read mandatory security unit **36**.
That security unit compares a user's security label passed by the data manager with a security label associated with the requested row of data in the data

storage unit **24**. If the conditions discussed above are met, access to the row is granted. That is, the read mandatory security unit **36** determines, from the user's security label, the user's security level and security categories. It also determines, from the row's security label, the row security level and row security categories. If the user's security level is greater than or equal to the security level for the row, and if the security categories associated with the row are a proper subset of the security categories associated with the user's security level, then read access is allowed. Since every attempted read access to the data passes through the read mandatory security unit, mandatory access control is achieved.

Similarly, when writing an updated row to the database, the write mandatory security unit **38** receives from the data manager **22** the request to store the row in the data storage unit **24**. The write mandatory security unit **38** ensures that the conditions discussed above are met before allowing the row to be updated in the data storage unit **24**. That is, the write mandatory security unit ensures that the user's security label indicates both that the user's security level and security categories correspond with the security level and categories indicated by the security label of the row to be updated.

Mandatory Access Control Example



In a DBMS employing the mandatory security access controls described here, a user may query the DBMS tables directly and mandatory access control is automatically performed. This allows a query, such as query 62 shown in FIG. 5A, to be applied to the DBMS without having to use a view to control access as in a conventional DBMS. Here, a query from "BOSS 2," shown in FIG. 5A, includes a SELECT clause 62. The SELECT clause is intended to select all data from the user table "USER.TABLE." If the DBMS does not include any access control, the contents of the entire table would be returned regardless of the user's security level. However, when this query is applied to the DBMS having the mandatory access controls, shown in FIG. 3, the only rows of data that are returned are the rows that the user is authorized to access.

This control aspect is illustrated in FIG. 5B. Here, the user table 26 includes rows that each bear a security label, namely a security label "red," "blue," "yellow," etc. When the query shown in FIG. 5A, originating from a user or an application 40, is received by the DBMS the query processor 20 processes the query and sends requests to the data manager 22 to select all rows of the USER.TABLE 26. However, the read security unit 36 operates to limit the rows of the user table that are returned based on a security mechanism 64. A variety of security mechanisms can be used, such as a table as shown in FIG. 5B, that relates a user ID to a security label for the user. Although, it will be understood

that other security mechanisms can be used to determine a security label associated with a user.

- **Claims Made in the Patent**

Claim 1: A Computer-implemented method of controlling access to a relational database consist of:

- determining user security information from the user security label;
- determining row security information for each row to be accessed by the database operation being performed by the request, from the security label column in the respective row;
- comparing the user security information and the row security information, the user security information and the row security information being unchanged during a period when the database operation is performed;
- retrieving one or more rows of data from the table in the relational database based on a result of the comparing the user security information and the row security information; and
- returning, to the user computer, only the one or more rows for which the user is determined to have authorization to access for performing the database operation on the rows

Claim 2: The method of claim 1, wherein the request contains one or more queries of one or more tables.
Claim 3: The method of claim 1, wherein the table containing the rows of data contains access control information for limiting user access to the relational database.
Claim 4: The method of claim 1, wherein the database operation is a query.
Claim 5: The method of claim 1, wherein the database operation involves a row update.
Claim 6: The method of claim 1, wherein said determining row security information includes checking a cache for row security information corresponding to respective row security label.
Claim 7: The method of claim 1, wherein the user security label is one of plurality of security labels arranged in a hierarchy of security levels.
Claim 8: The method of claim 1, wherein the user is authorized to modify the row only when the user security information is equal to the row security information.
Claim 9: A computer-implemented method of controlling access to data in at least one row of a relational database, wherein said at least one row is associated with row-level access control information, the method comprising:

- determining row security information for each row to be accessed by the database operation being performed by the request, from the security label column in the respective row;
- comparing the user security level and the row security level, the user security level and the row security level being unchanged during a period when the database operation is performed;
- retrieving one or more rows of data from the table in the relational database based on a result of the comparing the user security level and the row security level; and returning data from one or more rows in the relational database to the user computer, if the row security level associated with one or more rows is at least a subset of the user security level, to perform the database operation on the one or more rows.

Claim 10: The method of claim 9, wherein the request is a Structured Query Language (SQL) query.

Claim 11: The method of claim 9, wherein the relational database comprises a plurality of tables and the received request includes a request for access to at least one of the plurality of tables.

Claim 12: The method of claim 9, wherein the subset of the user security level corresponds to all security levels equal to or lower than the security level of the user.

Claim 13: A computer-implemented method of controlling a user's access to data in rows of a relational database, the method comprising:

- determining whether the user is authorized to operate on a row of the relational database that satisfies the request by comparing the first access level stored in the security label column of the table and a second access level and determining whether privileges associated with the first access level are included in the privileges associated with the second access level included in the user security label, the first access level and the second access level being unchanged during a period when the database operation is performed;
- retrieving one or more rows of data from the table in the relational database based on a result of the comparing the first access level and the second access level; and
- returning data from the one or more rows only if the user is determined to be authorized to perform the database operation on the one or more rows.

Claim 14: The method of claim 13, wherein the determining whether the privileges associated with the first access level are included in the privileges associated with the second access level comprising: determining whether the first access level is equal to or lower than the second access level.

Conclusion: Thus, we have successfully studied the DataBase Security Case Study.