Sanjivani College of Engineering, Kopargaon

Department of Electronics & Computer Engineering

(An Autonomous Institute)

Affiliated to Savitribai Phule Pune University

Accredited 'A' Grade by NAAC

SANJIVANI
GROUP OF INSTITUTES

Subject: Database Management Systems & SQL

by

Mr. Nitin Bhopale

Introduction to SQL

Installation of SQL/MySQL, SQL: Characteristics and advantages, SQL Data Types and Literals, DDL, DML, SQL Operators, Tables: Creating, Modifying, Deleting, Views: Creating, Dropping, Updating using Views, Indexes SQL DML Queries: SELECT Query and clauses, Set Operations, Predicates and Joins, OLTP vs OLAP.

- SQL is widely popular because it offers the following advantages:

Allows users to access data in the relational database management systems.

Allows users to describe the data.

Allows users to define the data in a database and manipulate that data.

Allows to embed within other languages using SQL modules, libraries

Allows users to create and drop databases and tables.

Allows users to create view, stored procedure, functions in a database

Allows users to set permissions on tables and views

SQL characteristics:

- SQL is domain specific
- SQL is declarative language (non procedural)
- Keys & constraints
- Operators
- Clauses (group by, order by, having, select, update, from, ..)
- Joins and nested queries

SQL	MySQL
SQL is a query programming language that manages RDBMS.	MySQL is a relational database management system that uses SQL.
SQL does not support any connector.	MySQL comes with an in-built tool known as MySQL Workbench that facilitates creating, designing, and building databases.
SQL follows a simple standard format without many or regular updates.	MySQL has numerous variants and gets frequent updates.
SQL supports only a single storage engine.	MySQL offers support for multiple storage engines along with plug-in storage, making it more flexible.
SQL does not allow other processors or even its own binaries to manipulate data during execution.	MySQL is less secure than SQL, as it allows third-party processors to manipulate data files during execution.

Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Note – Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

Character Strings Data Types

DATA TYPE	Description
char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

DATA TYPE	Description
nchar	Maximum length of 4,000 characters.(Fixed length Unicode)
nvarchar	Maximum length of 4,000 characters.(Variable length Unicode)

	char	nchar	varchar	nvarchar
Character Data Type	Non-Unicode fixed-length	Unicode fixed-length can store both non-Unicode and Unicode characters (i.e. Japanese, Korean etc.)	Non-Unicode variable length	Unicode variable length can store both non-Unicode and Unicode characters (i.e. Japanese, Korean etc.)
Maximum Length	up to 8,000 characters	up to 4,000 characters	up to 8,000 characters	up to 4,000 characters
Character Size	takes up 1 byte per character	takes up 2 bytes per Unicode/Non-Unicode character	takes up 1 byte per character	takes up 2 bytes per Unicode/Non-Unicode character
Storage Size	n bytes	2 times n bytes	Actual Length (in bytes)	2 times Actual Length (in bytes)
Usage	use when data length is constant or fixed length columns	use only if you need Unicode support such as the Japanese Kanji or Korean Hangul characters due to storage overhead	used when data length is variable or variable length columns and if actual data is always way less than capacity	use only if you need Unicode support such as the Japanese Kanji or Korean Hangul characters due to storage overhead
			query that uses a varchar parameter does an index seek due to column collation sets	query that uses a nvarchar parameter does an index scan due to column collation sets

Advantages and Disadvantages of char, nchar, varchar and nvarchar in SQL Server

Data Types	Advantages	Disadvantages
char	Query performance is better since no need to move the column while updating. No need to store the length of string in last two bytes.	If not properly used, it can take more space than varchar since it is fixed length and we don't know the length of string to be stored. It is not good for compression since it embeds space characters at the end.
varchar	Since it is variable length it takes less memory spaces.	Decreases the performance of some SQL queries.
nchar/nvarchar	Supports many client computers that are running different locales.	If not properly used it may use up a lot of extra storage space.

String Literals
String literals are always surrounded by either single quotes (') or double quotes (").

Number Literals
Number literals can be either positive or negative numbers that are exact or floating point values.

Date and Time Literals
Date and time literals can be expressed as either strings or numbers.

Boolean Literals
Boolean literals are values that evaluate to either 1 or 0.

SQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressior
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductID FROM
OrderDetails WHERE Quantity = 10);
```

```
SELECT * FROM Customers
WHERE City = "London" AND Country = "UK";
```

```
SELECT * FROM Products
WHERE Price BETWEEN 50 AND 60;
```

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products
WHERE Products.SupplierID = Suppliers.supplierID
AND Price < 20);
```

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressior
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

```
SELECT * FROM Customers
WHERE City IN ('Paris','London');
```

```
SELECT * FROM Customers
WHERE City LIKE 's%';
```

```
SELECT * FROM Customers
WHERE City NOT LIKE 's%';
```

```
SELECT * FROM Customers
WHERE City = "London" OR Country = "UK";
```

```
SELECT * FROM Products
WHERE Price > SOME (SELECT Price FROM Products
WHERE Price > 20);
```

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressior
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Comparison Operator	Description
=	Equal
<=>	Equal (Safe to compare NULL values)
<>	Not Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal
IN ( )	Matches a value in a list
NOT	Negates a condition
BETWEEN	Within a range (inclusive)
IS NULL	NULL value
IS NOT NULL	Non-NULL value
LIKE	Pattern matching with % and _
EXISTS	Condition is met if subquery returns at least one row

Example - Equality Operator

In MySQL, you can use the = operator to test for equality in a query.
The = operator can only test equality with values that are not NULL.

```
SELECT *
FROM contacts
WHERE last_name = 'Johnson';
```

Example - Inequality Operator

In MySQL, you can use the <> or != operators to test for inequality in a query.

```
SELECT *
FROM contacts
WHERE last_name <> 'Johnson';
```

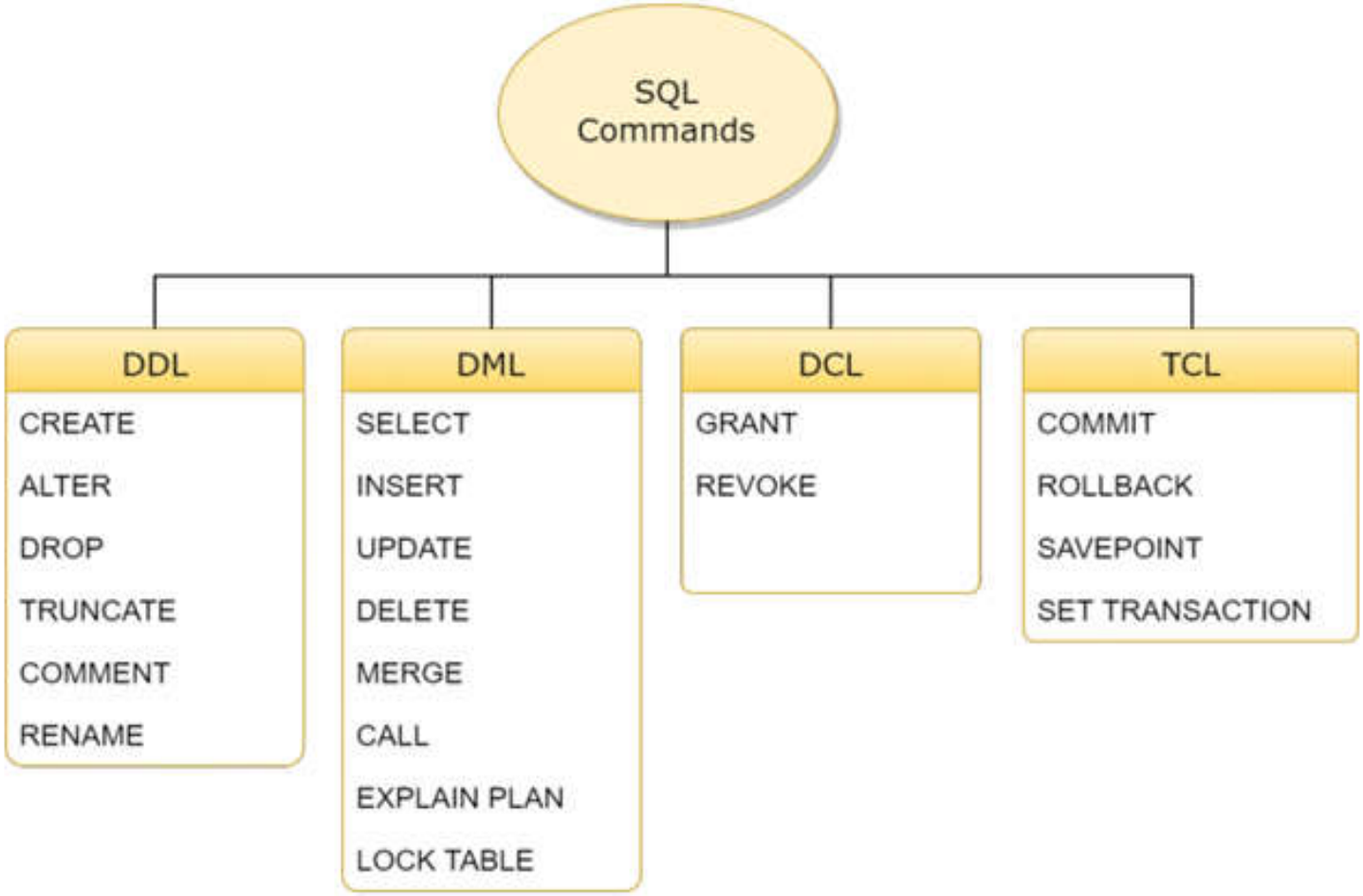

SQL Commands
DDL
CREATE
ALTER
DROP
TRUNCATE
COMMENT
RENAME
DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

Table create

Syntax- CREATE TABLE table_name (column_name column_type);

Example-

```
CREATE TABLE User (
    id int not null auto_increment,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    Salary varchar(255),
    primary key(id)
);
```

Table insert-

Syntax-

insert into table_name (column1, column2,...columnN)VALUES(value1, value2 ,...valueN);

Example-

For single insert records-
insert into User (lastName,firstName,Address,City,Salary)values ('naik', 'jay', `yeola',`nashik', 25000);

For multiple insert records-
insert into User (lastName,firstName,Address,City,Salary) values ('patil', 'ashok', `nirmal pimpri',`shirdi', 45000),('patil', 'ram', 'kopargaon',`nagar', 35000);

Table update-

Syntax

UPDATE table_name SET column_name1 = new-value1, column_name2=new-value2, ... [WHERE Clause]

Example
update user set firstName = 'rohan' where lastName='kulkarni' //Single record or field

or
update user set firstName = 'rohan' where id=3

Table delete

Syntax-

DELETE FROM table_name WHERE condition; //specific data delete

Example-

delete from user where id=3;

Table select

Syntax-

SELECT field_name1, field_name 2,... field_nameN FROM table_name1 [WHERE condition]

Example-

select * from user;

select * from user where city='pune‘

Syntax-Add

ALTER TABLE *table_name* ADD *column_name datatype*;

Example-

ALTER TABLE user ADD Email varchar(255);

Syntax-Modify

ALTER TABLE *table_name* MODIFY COLUMN *column_name datatype*;

Example-

ALTER TABLE user modify Email varchar(125);

Syntax-Drop

ALTER TABLE *table_name* DROP COLUMN *column_name*;

Example-

ALTER TABLE user drop Email;

Syntax-Update

Update table_name set column_name=<value> where [];

Clauses-

1. Where- clause is used with SELECT, INSERT, UPDATE and DELETE clause to filter the results. It specifies a specific position where you have to do the operation.

Syntax- WHERE conditions;

Example- SELECT * FROM student WHERE city = 'pune';

SELECT * FROM student WHERE city = ‘pune’ AND id < 5;

select * from student where salary>=25000 AND salary<=48000;

1. DISTINCT- clause is used to remove duplicate records from the table and fetch only the unique records. The DISTINCT clause is only used with the SELECT statement.

   Syntax- SELECT DISTINCT(expressions)FROM tables [WHERE conditions];

   Example- SELECT DISTINCT (city) FROM student;

1. FROM- clause is used to select some records from a table. It can also be used to retrieve records from multiple tables using JOIN condition.

2. ORDER BY- Clause is used to sort the records in ascending or descending order.

   Syntax-

   SELECT expressions FROM tables [WHERE conditions] ORDER BY expression [ASC | DESC];

   Example- SELECT * FROM student WHERE city = 'pune' ORDER BY salary;

Aggregate function-

1. MySQL count() function is used to return the count of an expression.

```
mysql> SELECT COUNT(emp_name) FROM employees;
```

1. The MIN() & MAX() function in MySQL is used to return the **minimum value and maximum** in a set of values from the table.

```
mysql> SELECT MIN(income) AS Minimum_Income FROM employees;
```

1. **Primary Key-**

MySQL primary key is used to identify each record in a table uniquely.

It cannot be null or empty.

It can contain only one primary key.

It always contains unique value into a column.

Rules-

1. The primary key column value must be unique.
2. Each table can contain only one primary key.
3. The primary key column cannot be null or empty.

1. Foreign Key-

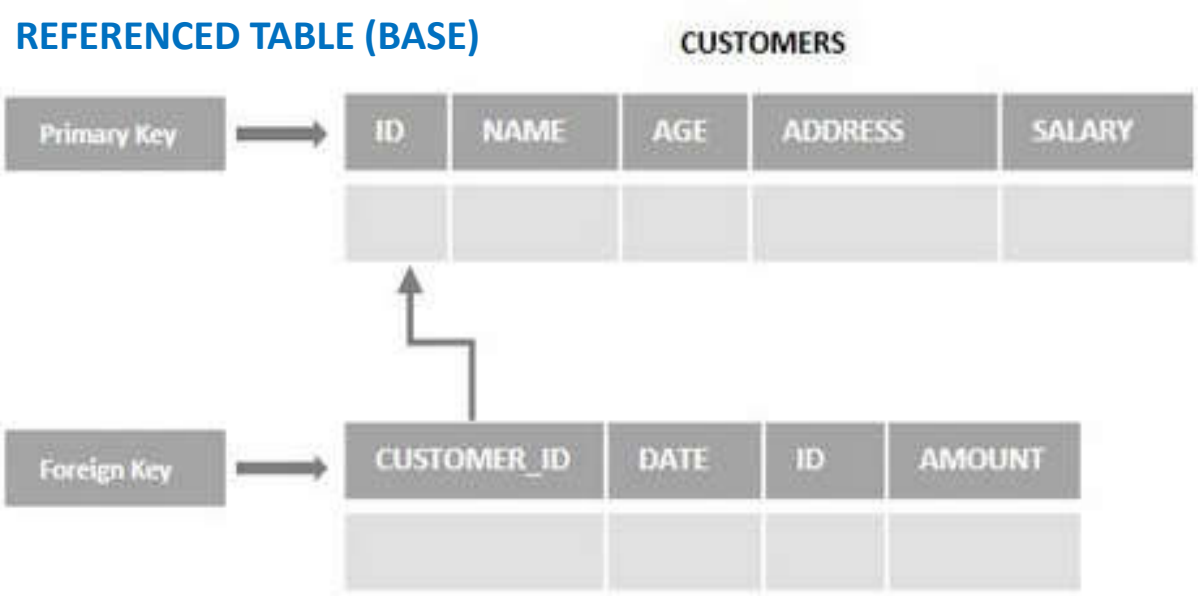
The foreign key is used to link one or more than one table together. It is also known as the **referencing** key.

A foreign key matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of the other table.

Primary Key	Foreign Key
The primary key is always unique.	The foreign key can be duplicated.
The primary key can not be NULL.	The Foreign can be NULL.
A table can contain only one Primary Key.	We can have more than one Foreign Key per table.

Foreign Key & Referential Integrity

REFERENCED TABLE (BASE)



REFERENCING TABLE

ORDERS

- A Foreign Key is used to reduce the redundancy (or duplicates) in the table.
- It helps to normalize (or organize the data in a database) the data in multiple tables.

FOR NEW TABLE CREATION WITH FK

```
CREATE TABLE ORDERS (
    ID INT NOT NULL,
    DATE DATETIME,
    CUSTOMER_ID INT,
    CONSTRAINT FK_CUSTOMER
    FOREIGN KEY(CUSTOMER_ID)
    REFERENCES CUSTOMERS(ID),
    AMOUNT DECIMAL,
    PRIMARY KEY (ID)
);
```

FOR EXISTING TABLE, WITH FK

```
ALTER TABLE ORDERS
ADD CONSTRAINT FK_ORDERS
FOREIGN KEY(ID)
REFERENCES CUSTOMERS(ID);
```

Can’t drop CUSTOMERS before dropping the ORDERS table.

Can’t insert in ORDERS table.

Can’t DELETE from CUSTOMERS if it is in ORDERS table.

S.NO.	Delete	Truncate
1.	The DELETE command is used to delete specified rows(one or more).	While this command is used to delete all the rows from a table.
2.	It is a DML(Data Manipulation Language) command.	While it is a DDL(Data Definition Language) command.
3.	There may be WHERE clause in DELETE command in order to filter the records.	While there may not be WHERE clause in TRUNCATE command.
4.	We can rollback the data even after using DELETE command.	While in this command, we can’t rollback.
5.	DELETE command is slower than TRUNCATE command.	While TRUNCATE command is faster than DELETE command.

Qs: In the student table :

a. Change the name of column "name" to "full_name".

b. Delete all the students who scored marks less than 80.

c. Delete the column for grades.

```sql
ALTER TABLE student
CHANGE name full_name VARCHAR(50);
```

```sql
DELETE FROM student
WHERE marks < 80;
```

```sql
ALTER TABLE student
DROP COLUMN grade;
```

Inner Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

Result

student_id	name	course
102	bob	english
103	casey	science

Left Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *
FROM student as s
LEFT JOIN course as c
ON s.student_id = c.student_id;
```

Result

student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science

Right Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

```
SELECT *
FROM student as s
RIGHT JOIN course as c
ON s.student_id = c.student_id;
```

Result

student_id	course	name
102	english	bob
105	math	*null*
103	science	casey
107	computer science	*null*

SQL Sub Queries

Example

Get names of all students who scored more than class average.

Step 1. Find the avg of class
Step 2. Find the names of students with marks > avg

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

```
SELECT name, marks
FROM student
WHERE marks > (SELECT AVG(marks) FROM student);
```

SQL Sub Queries

Example

Find the names of all students with even roll numbers.

Step 1. Find the even roll numbers
Step 2. Find the names of students with even roll no

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

```
SELECT rollno
FROM student
WHERE rollno % 2 = 0;
```

```
SELECT name, rollno
FROM student
WHERE rollno IN (
    SELECT rollno
    FROM student
    WHERE rollno % 2 = 0);
```

customer_id	customer	mode	city
101	Olivia Barrett	Netbanking	Portland
102	Ethan Sinclair	Credit Card	Miami
103	Maya Hernandez	Credit Card	Seattle
104	Liam Donovan	Netbanking	Denver
105	Sophia Nguyen	Credit Card	New Orleans
106	Caleb Foster	Debit Card	Minneapolis
107	Ava Patel	Debit Card	Phoenix
108	Lucas Carter	Netbanking	Boston
109	Isabella Martinez	Netbanking	Nashville
110	Jackson Brooks	Credit Card	Boston

For the given table, find the total payment according to each payment method.

Predicates

SELECT Company, Phone FROM supplier WHERE State = ‘MH' OR State = ‘AP' OR State = ‘UP' ;

SELECT Company, Phone FROM SUPPLIER WHERE State IN (‘MH', ‘AP', ‘UP') ;

Comparison Predicates

=	Equal
<>	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

Other Predicates

ALL	BETWEEN
DISTINCT	EXISTS
IN	LIKE
MATCH	NOT IN
NOT LIKE	NULL
OVERLAPS	SOME, ANY
UNIQUE	

SQL Set Operation

The SQL Set operation is used to combine two or more SQL SELECT statements.

Types of Set Operation

Union, UnionAll, Intersect, Minus

The SQL Union operation is used to combine the result of two or more SQL SELECT queries.

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

RULES:

1. Number and the orders of columns must be Same.
2. Data Types of the Column must be Same

1. `select fname from faculty UNION select sname from student;`
2.
```
mysql> select fname from faculty
    -> where fname in(select sname from student);
```
3.
```
mysql> select distinct fname from faculty
    -> where fname in (select sname from student);
```
4. Find Student Name who is **not allocated** to any Department.

Fid	Fname	Qualification	Deptid
1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
5	Ravi	Ph.D	3
6	Aarti	M.Tech	2
7	Varun	Ph.D	5

Sid	Sname	Bra
1	Lalit	IT
2	Mahesh	IT
3	Sagar	CSE
4	Arun	CSE
5	Shikha	ECE
6	Anand	ECE
7	Parul	IT
8	John	NUL

Aman
Mohan
Vishal
Priya
Ravi
Aarti
Varun
Lalit
Mahesh
Sagar
Arun
Shikha
Anand
Parul
John

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

ID	NAME
1	Jack
2	Harry
3	Jackson

ID	NAME
3	Jackson
4	Stephan
5	David

- **3. Intersect**
- The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	NAME
3	Jackson

ID	NAME
1	Jack
2	Harry
3	Jackson

ID	NAME
3	Jackson
4	Stephan
5	David

- **4. Minus**
- Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry

ID	NAME
1	Jack
2	Harry
3	Jackson

ID	NAME
3	Jackson
4	Stephan
5	David

Views

It is virtual table which is not stored physically in database, but all operations can be performed on it.(It is stored set of query).

Read only views, Updatable views(only DML command support), Materialised views(copy of remote location data).

Advantages:

- Data presentation
- Enable computing on columns
- Data security
- Complex query can be simplified

Disadvantages:

- Performance decreases
- Dependency on table

1. Create a View that contain records of **only IT and CSE Students.**

```
mysql> create view view1
    -> as
    -> select * from student where Branch in ('IT','CSE');
```

```
+------+--------+--------+-------+
| Sid  | Sname  | Branch | marks |
+------+--------+--------+-------+
|    1 | Lalit  | IT     |    75 |
|    2 | Mahesh | IT     |    90 |
|    3 | Sagar  | CSE    |    92 |
|    4 | Arun   | CSE    |    45 |
|    5 | Shikha | ECE    |    56 |
|    6 | Anand  | ECE    |    78 |
|    7 | Parul  | IT     |    82 |
|    8 | John   | NULL   |    89 |
|    9 | Varun  | ECE    |    70 |
+------+--------+--------+-------+
```

```
mysql> select * from view1;
+------+--------+--------+-------+
| Sid  | Sname  | Branch | marks |
+------+--------+--------+-------+
|    1 | Lalit  | IT     |    75 |
|    2 | Mahesh | IT     |    90 |
|    3 | Sagar  | CSE    |    92 |
|    4 | Arun   | CSE    |    45 |
|    7 | Parul  | IT     |    82 |
+------+--------+--------+-------+
```

2. Create a View that cont
Corresponding **Department Nar**

Fid	Fname	Qualification	Deptid
1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
5	Ravi	Ph.D	3
6	Aarti	M.Tech	2
7	Varun	Ph.D	5

```
mysql> select * from view2;
+--------+-------+
| fname  | dname |
+--------+-------+
| Aman   | IT    |
| Mohan  | IT    |
| Vishal | IT    |
| Priya  | CSE   |
| Aarti  | CSE   |
| Ravi   | ECE   |
| Varun  | ICE   |
+--------+-------+
mysql> create view view2
    -> as
    -> select f.fname,d.dname from
    -> faculty f, department d
    -> where f.deptid = d.deptid;
```

```
mysql> create view view3
    -> as
    -> select sname, marks from student where
    -> marks>(select avg(marks) from student);
```

Sid	Sname	Branch	marks
1	Lalit	IT	75
2	Mahesh	IT	90
3	Sagar	CSE	92
4	Arun	CSE	45
5	Shikha	ECE	56
6	Anand	ECE	78
7	Parul	IT	82
8	John	NULL	89
9	Varun	ECE	70

```
mysql> select * from view3;
```

sname	marks
Mahesh	90
Sagar	92
Anand	78
Parul	82
John	89

3. Create a View that contain student names and marks where each student has **higher marks than average marks** of all the students.**[Use SubQuery]**

1. **Insert Some Records** in the View of Faculty Table
2. **Delete Record of Faculty Id = 7** from View of Faculty
3. **Update** the Dept. Number of **Faculty Id = 6**

```
mysql> select * from faculty;
+------+--------+----------------+--------+
| Fid  | Fname  | Qualification  | Deptid |
+------+--------+----------------+--------+
|    1 | Aman   | B.Tech         |      1 |
|    2 | Mohan  | M.Tech         |      1 |
|    3 | Vishal | M.Tech         |      1 |
|    4 | Priya  | Ph.D           |      2 |
|    5 | Ravi   | Ph.D           |      3 |
|    6 | Aarti  | M.Tech         |      2 |
|    7 | Varun  | Ph.D           |      5 |
+------+--------+----------------+--------+
```

```
mysql> select * from view4;
+------+--------+----------------+--------+
| Fid  | Fname  | Qualification  | Deptid |
+------+--------+----------------+--------+
|    1 | Aman   | B.Tech         |      1 |
|    2 | Mohan  | M.Tech         |      1 |
|    3 | Vishal | M.Tech         |      1 |
|    4 | Priya  | Ph.D           |      2 |
|    6 | Aarti  | M.Tech         |      2 |
+------+--------+----------------+--------+
```

```
mysql> create view view4 as select * from faculty where deptid in (1,2);
```

```
mysql> insert into view4 values(10,'Temp1','Ph.D',1);
```

```
mysql> insert into view4 values(11,'Temp2','Ph.D',3);
```

```
mysql> select * from faculty;
```

1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
5	Ravi	Ph.D	3
6	Aarti	M.Tech	2
7	Varun	Ph.D	5
10	Temp1	Ph.D	1
11	Temp2	Ph.D	3

```
mysql> select * from view4;
```

Fid	Fname	Qualification	Deptid
1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
6	Aarti	M.Tech	2
10	Temp1	Ph.D	1

```
mysql> delete from view4 where fid = 7;
```

```
mysql> select * from view4;
+------+--------+---------------+--------+
| Fid  | Fname  | Qualification | Deptid |
+------+--------+---------------+--------+
|    1 | Aman   | B.Tech        |      1 |
|    2 | Mohan  | M.Tech        |      1 |
|    3 | Vishal | M.Tech        |      1 |
|    4 | Priya  | Ph.D          |      2 |
|    6 | Aarti  | M.Tech        |      2 |
|   10 | Temp1  | Ph.D          |      1 |
+------+--------+---------------+--------+
```

```
mysql> select * from faculty;
+------+--------+---------------+--------+
| Fid  | Fname  | Qualification | Deptid |
+------+--------+---------------+--------+
|    2 | Mohan  | M.Tech        |      1 |
|    3 | Vishal | M.Tech        |      1 |
|    4 | Priya  | Ph.D          |      2 |
|    5 | Ravi   | Ph.D          |      3 |
|    6 | Aarti  | M.Tech        |      2 |
|   10 | Temp1  | Ph.D          |      1 |
|   11 | Temp2  | Ph.D          |      3 |
+------+--------+---------------+--------+
```

```
mysql> update view4
    -> set deptid = 3 where fid = 6;
```

- If view is created for more than 1 table then it CANNOT BE UPDATED
- VIEW OVER VIEW CAN BE CREATED.
- Views cannot be updated if query contains GROUP BY, ORDER BY, WITH.

```
mysql> select * from faculty;
+------+--------+---------------+--------+
| Fid  | Fname  | Qualification | Deptid |
+------+--------+---------------+--------+
|    2 | Mohan  | M.Tech        |      1 |
|    3 | Vishal | M.Tech        |      1 |
|    4 | Priya  | Ph.D          |      2 |
|    5 | Ravi   | Ph.D          |      3 |
|    6 | Aarti  | M.Tech        |      3 |
|   10 | Temp1  | Ph.D          |      1 |
+------+--------+---------------+--------+
```

```
mysql> select * from view4;
+------+--------+---------------+--------+
| Fid  | Fname  | Qualification | Deptid |
+------+--------+---------------+--------+
|    2 | Mohan  | M.Tech        |      1 |
|    3 | Vishal | M.Tech        |      1 |
|    4 | Priya  | Ph.D          |      2 |
|   10 | Temp1  | Ph.D          |      1 |
+------+--------+---------------+--------+
```

Reason for view4 table display ?

```
mysql> create view view4 as select * from faculty where deptid in (1,2);
```

- Check for **drop view nameofview;**
- ASSIGNING THE ROLE AND GRANT THE ROLE ???????

ALTER ADD COLUMN **WILL NOT BE WORKING ON THE VIEW BUT RATHER IT WILL WORK ON BASE TABLE ONLY…..**

If there is not null constraint on the column of original table, then insertion will not be possible on the Views.

Find the faculties with deptid 1 /3

Fid	Fname	Qualification	Deptid
1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
5	Ravi	Ph.D	3
6	Aarti	M.Tech	2

```
mysql> select fname from faculty where deptid=1 or deptid=3;
+--------+
| fname  |
+--------+
| Aman   |
| Mohan  |
| Vishal |
| Ravi   |
+--------+
mysql> select fname from faculty where deptid IN (1,3);
+--------+
| fname  |
+--------+
| Aman   |
| Mohan  |
| Vishal |
| Ravi   |
+--------+
```

Find the faculties with qualifications as BTECH OR PHD

```
mysql> select * from faculty where qualification IN ('B.Tech','Ph.D');
+------+-------+---------------+--------+
| Fid  | Fname | Qualification | Deptid |
+------+-------+---------------+--------+
|    1 | Aman  | B.Tech        |      1 |
|    4 | Priya | Ph.D          |      2 |
|    5 | Ravi  | Ph.D          |      3 |
+------+-------+---------------+--------+
```

Find Student Details who are **not from IT or CSE Branch**

```
mysql> select * from student where branch NOT IN('IT','CSE');
+------+--------+--------+-------+
| Sid  | Sname  | Branch | marks |
+------+--------+--------+-------+
|    5 | Shikha | ECE    |    56 |
|    6 | Anand  | ECE    |    78 |
+------+--------+--------+-------+
```

Syntax:

Select Col1, Col2, …… aggregate_function(Col)
From TableName where condition
Group By Col1, Col2,…
Having Condition

1. Find **Total Number of Students** in each branch
2. Find **Total Marks** of Each Branch
3. Find **Average Marks** of Each Branch having marks branch only **CSE or IT**

```
select branch,count(*) from student GROUP BY branch;

select branch,sum(marks) from student GROUP BY branch;

select branch,sum(marks) from student GROUP BY branch having branch IN ('CSE','IT');
```

Faculty

Fid	Fname	Qualification	Deptid
1	Aman	B.Tech	1
2	Mohan	M.Tech	1
3	Vishal	M.Tech	1
4	Priya	Ph.D	2
5	Ravi	Ph.D	3
6	Aarti	M.Tech	2
7	Varun	Ph.D	5

Department

deptid	dname
1	IT
2	CSE
3	ECE
4	EEE

Student

Sid	Sname	Branch	marks
1	Lalit	IT	75
2	Mahesh	IT	90
3	Sagar	CSE	92
4	Arun	CSE	45
5	Shikha	ECE	56
6	Anand	ECE	78
7	Parul	IT	82
8	John	NULL	89

1. Find the Name of **Faculty** Working in **IT Department**
2. Find the Name of **Faculty** Working in **IT or CSE Department**

```
select fname from faculty where deptid IN (select deptid from department where dname = 'IT');
```

```
select fname from faculty where deptid IN (select deptid from department where dname IN ('IT','C
```

3. Find **Average Marks** of Students who are not from **Department 1 or 3**

```
mysql> select branch, avg(marks) from student
    -> where branch NOT IN(select dname from
    -> department where deptid NOT IN(1,3))
    -> ;
```

Student

Sid	Sname	Branch	marks
1	Lalit	IT	75
2	Mahesh	IT	90
3	Sagar	CSE	92
4	Arun	CSE	45
5	Shikha	ECE	56
6	Anand	ECE	78
7	Parul	IT	82
8	John	NULL	89

Department

deptid	dname
1	IT
2	CSE
3	ECE
4	EEE

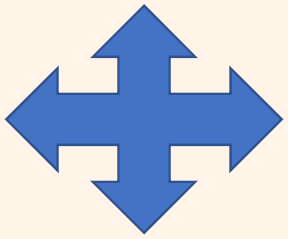
```
select * from student where EXISTS (select * from department where deptid=1);
```

Sid	Sname	Branch	marks
1	Lalit	IT	75
2	Mahesh	IT	90
3	Sagar	CSE	92
4	Arun	CSE	45
5	Shikha	ECE	56
6	Anand	ECE	78
7	Parul	IT	82
8	John	NULL	89

```
mysql> select * from student where NOT EXISTS (select * from department where deptid=1);
Empty set (0.00 sec)
```

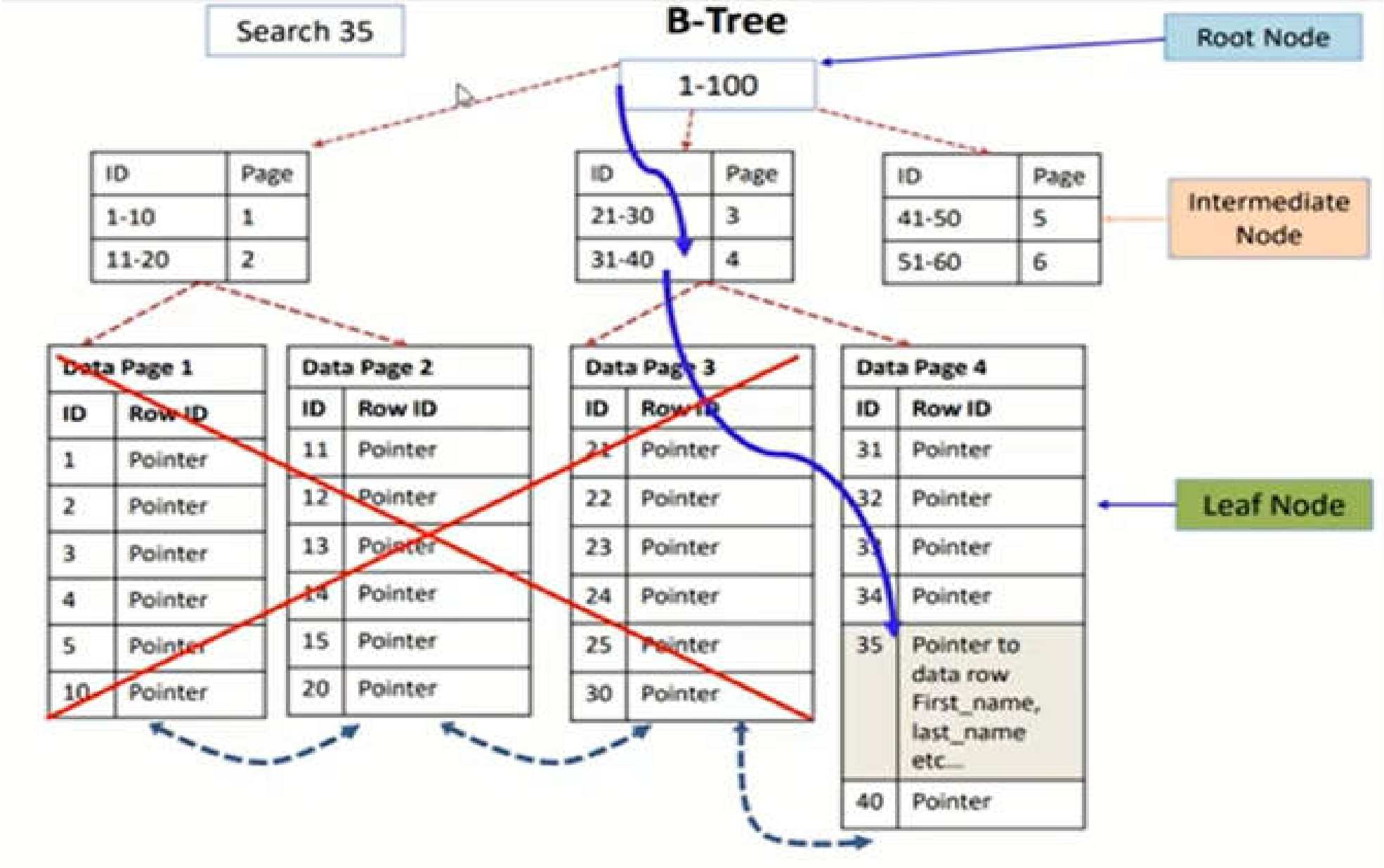
Indexes

(to be applied on read operations)

ORDERED DATA	PRIMARY INDEX (SPARSE=> no. of blocks in HDD = no of entries in Index TABLE *carries Anchor* eg: rn, name in HDD) TIME COMP: Log N+1	CLUSTERED INDEX (SPARSE=>for extra data in the next block *use block hanker, eg: dno, dname in HDD*) TIME COMP :logN+1+1…
UNORDERED DATA	SECONDARY INDEX DENSE	SECONDARY INDEX DENSE
	KEY	NON KEY

AT MOST 1 INDEX PER BLOCK FOR PRIMARY AND CLUSTERED INDEX, TO REDUCE THE I/O COST.

SECONDARY INDEX
INDEX TABLE
Key Pointer
1
4
7
10
Anchor
EID ENAME EPAN
1 A 22
2 B 11
3 C 66
4 G 33
5 A 77
6
7
8
9
10
INDEX TABLE
Key Pointer
11
22
33
66
Based on key EPAN
SECONDARY index followed since the DENSE and
Select * from employee where EPAN=?
no of entries in HDD=
no of entries in Index TABLE
Primary index followed since the sparse and
select * from employee where EID=?
no of entries in HDD=
no of entries in Index TABLE
A A
B B
C C
INDEX TABLE
Key Pointer
A
B
C
Based on Non key ENAME
(intermediate state)BLOCK OF RECORD POINTER

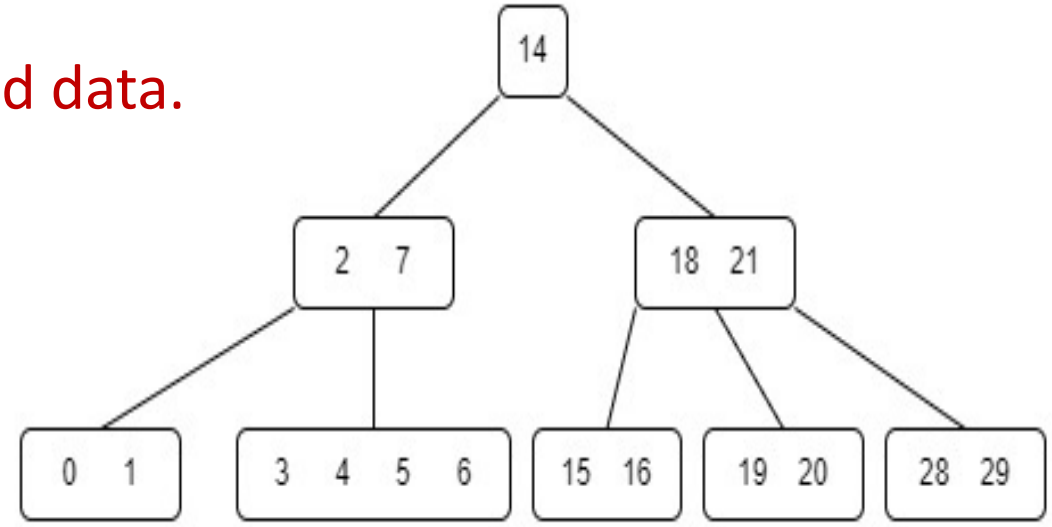

Search 35
B-Tree
Root Node
1-100
ID
Page
1-10
1
11-20
2
21-30
3
31-40
4
41-50
5
51-60
6
Intermediate Node
Data Page 1
ID
Row ID
1
Pointer
2
Pointer
3
Pointer
4
Pointer
5
Pointer
10
Pointer
Data Page 2
11
12
13
14
15
20
Data Page 3
22
23
24
25
30
Data Page 4
31
32
34
35
Pointer to data row First_name, last_name etc...
40
Leaf Node

B (Balanced) Trees

- Every node in a B-Tree contains at most m children.
- Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
- The root nodes must have at least 2 nodes.
- All leaf nodes must be at the same level.
- It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

Properties of B tree

- Every node in a B Tree will hold a maximum of m children and **(m-1)** keys, since the order of the tree is m.
- Every node in a B tree, except root and leaf, can hold at least **m/2** children
- The root node must have minimum two children(one key).
- All the paths in a B tree must end at the same level, i.e. the leaf nodes must be at the same level.
- B tree always maintains sorted data.



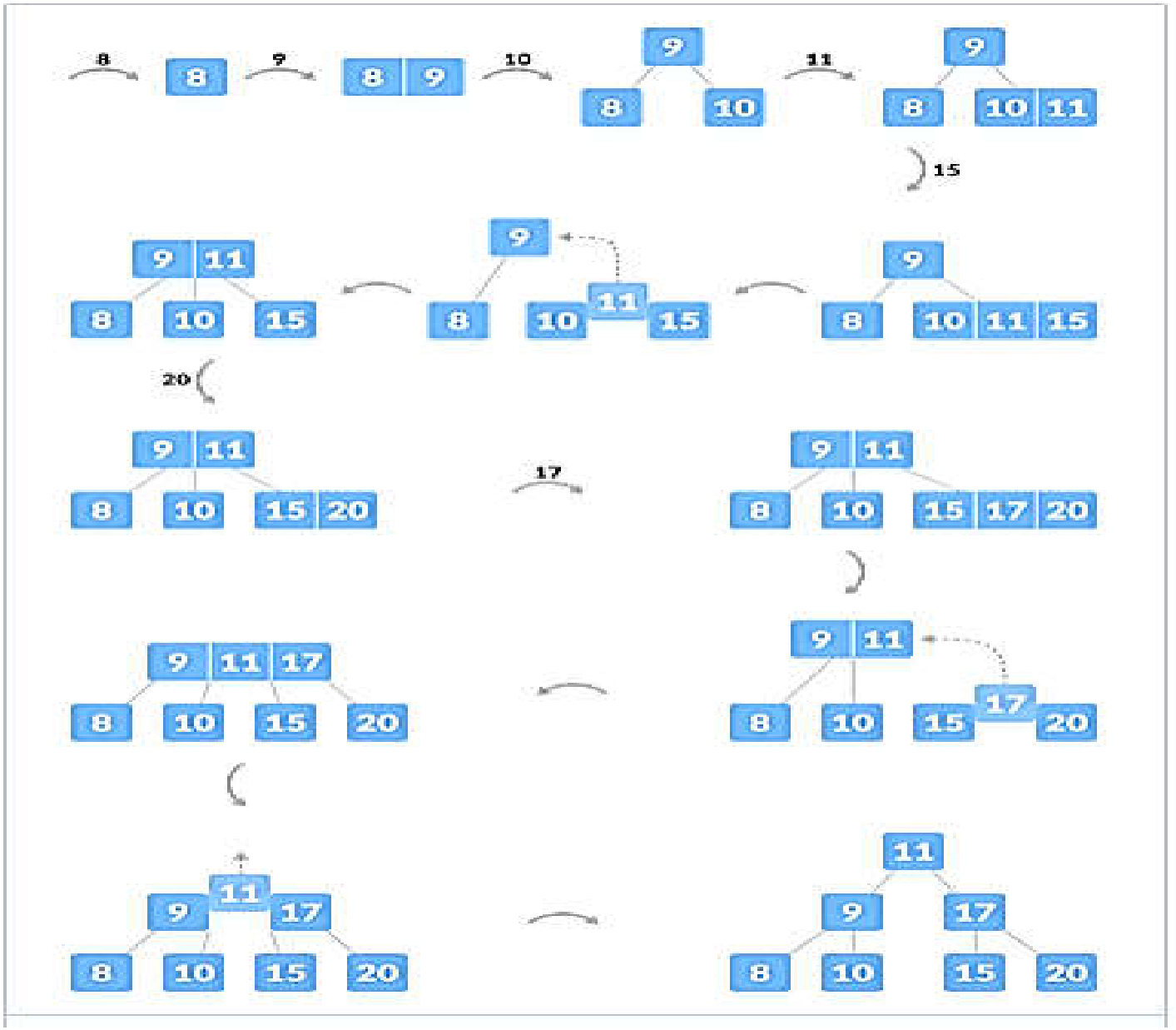
Why do you need a B-tree data structure?

- The need for B-tree arose with the rise in the need for lesser time in accessing physical storage media like a hard disk. The secondary storage devices are slower with a larger capacity. There was a need for such types of data structures that minimize the disk access.
- Other data structures such as a binary search tree, avl tree, red-black tree, etc can store only one key in one node. If you have to store a large number of keys, then the height of such trees becomes very large, and the access time increases.
- However, B-tree can store many keys in a single node and can have multiple child nodes. This decreases the height significantly allowing faster disk accesses.

Insertion Operation in B Tree

- If the tree is empty, allocate a root node and insert the key.
- Update the allowed number of keys in the node.
- Search the appropriate node for insertion.
- If the node is full, follow the steps below.
- Insert the elements in increasing order.
- Now, there are elements greater than its limit. So, split at the median.
- Push the median key upwards and make the left keys as a left child and the right keys as a right child.
- If the node is not full, follow the steps below.
- Insert the node in increasing order.

Insert to form B Tree
8,9,10,11,15,20,17
With Order =3



Formulas for reference

- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys ($\lceil \frac{m}{2} \rceil$) – 1 = 1
- Maximum Children = 4
- Minimum Children ($\lceil \frac{m}{2} \rceil$) = 2

CASE: MORE THAN MAX KEYS

INSERT 9

Order = 4

Index = 5,3,21,9,1,13,2,7,10,12,4,8

1 Search for appropriate position

2 Insert the key, and check for rules

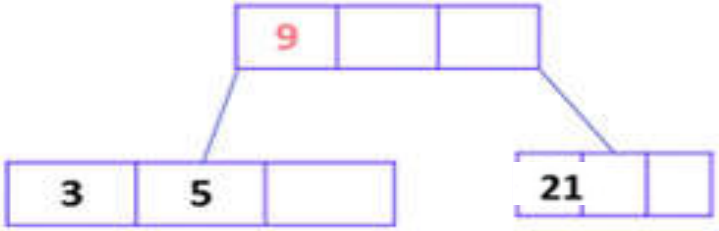
Greater than 3?

Yes

Split the node

LEFT BIAS TREE

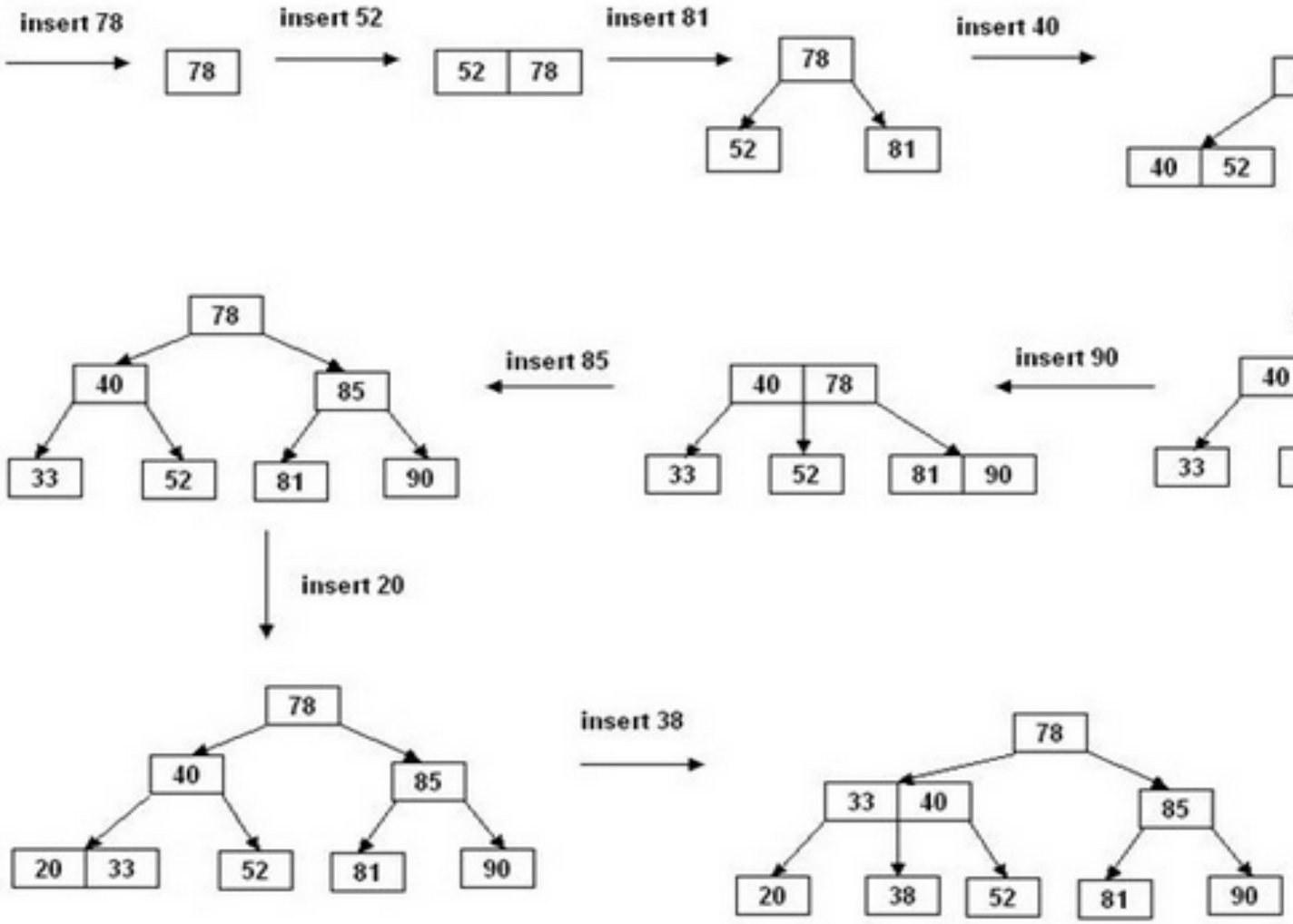
RIGHT BIAS TREE



Formulas for reference

- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys $(\lceil \frac{m}{2} \rceil) - 1 = 1$
- Maximum Children = 4
- Minimum Children $(\lceil \frac{m}{2} \rceil) = 2$

Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



Formulas for reference

- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys ($\lceil \frac{m}{2} \rceil$) – 1 = 1
- Maximum Children = 4
- Minimum Children ($\lceil \frac{m}{2} \rceil$) = 2

Insertion operation

The insertion operation for a B Tree is done similar to the Binary Search Tree but the elements are inserted into the same node until the maximum keys are reached. The insertion is done using the following procedure –

Step 1 – Calculate the maximum $(\mathbf{m}-\mathbf{1})$ and, minimum $\left(\left\lceil\frac{\mathbf{m}}{\mathbf{2}}\right\rceil-\mathbf{1}\right)$ number of keys a node can hold, where m is denoted by the order of the B Tree.

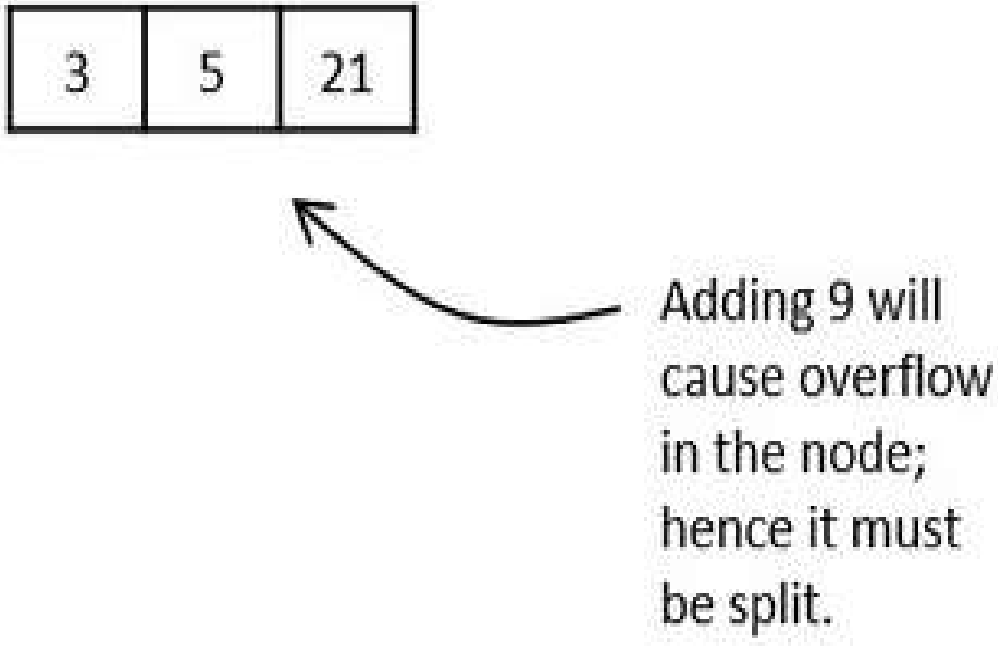
Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree

- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys $\left(\left\lceil\frac{m}{2}\right\rceil\right) - 1 = 1$
- Maximum Children = 4
- Minimum Children $\left(\left\lceil\frac{m}{2}\right\rceil\right) = 2$

Formulas for reference

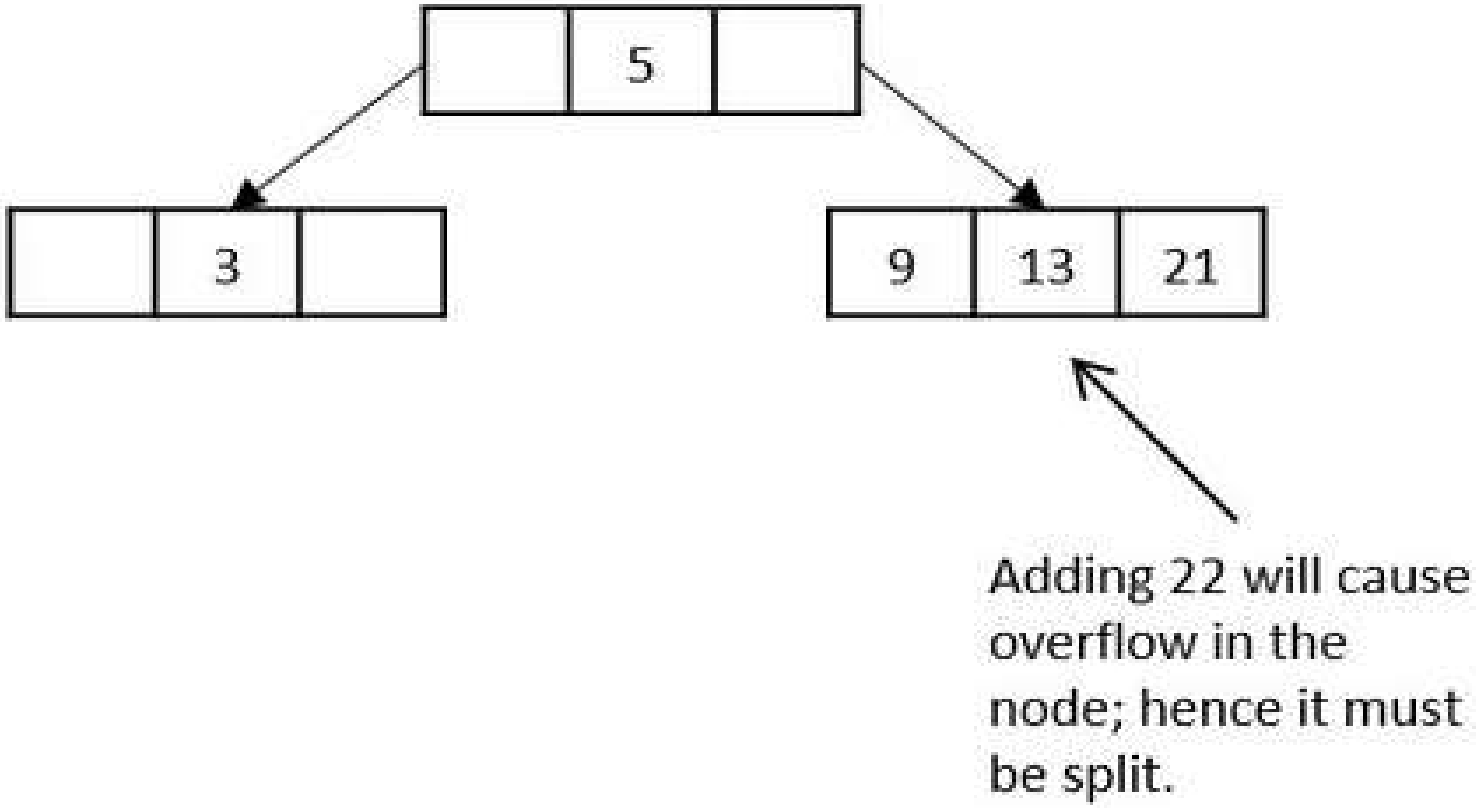
Step 2 – The data is inserted into the tree using the binary search insertion and once the keys reach the maximum number, the node is split into half and the median key becomes the internal node while the left and right keys become its children.

Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree

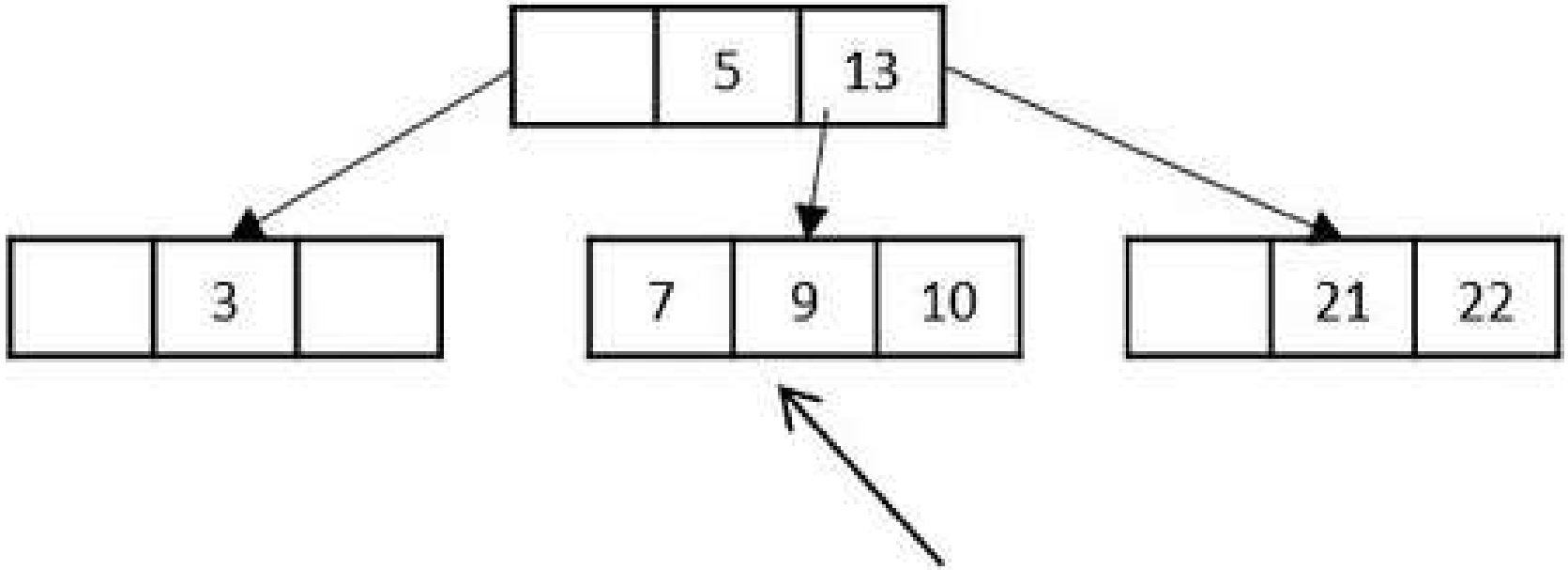


Step 3 – All the leaf nodes must be on the same level.

Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree

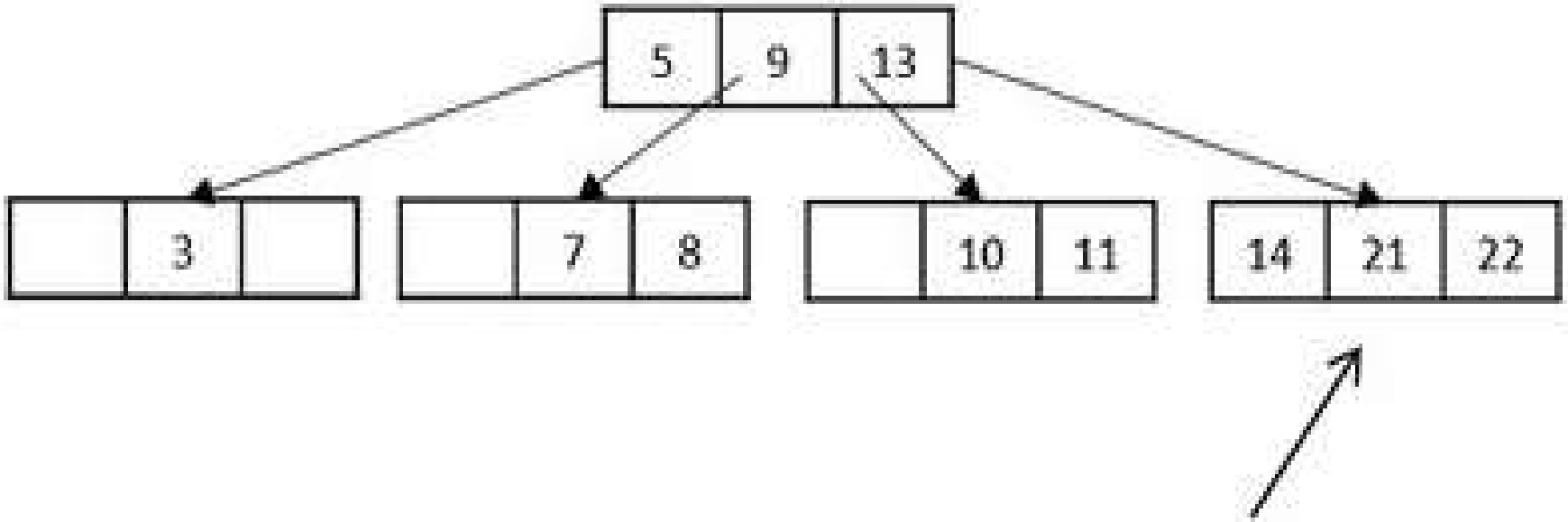


Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree



Adding 11 will cause overflow in the node; hence it must be split.

Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree



Obtain the B tree for the following insertions

- 1,2,3,4,5,6,7,8,9,10 if ORDER m=4
- 10,20,40,50,60,70,80 if ORDER m=4
- 10,20,30,40,50,60,70,80 if ORDER m=3
- 5,3,21,9,1,13,2,7,10,12,4,8 if ORDER m=4

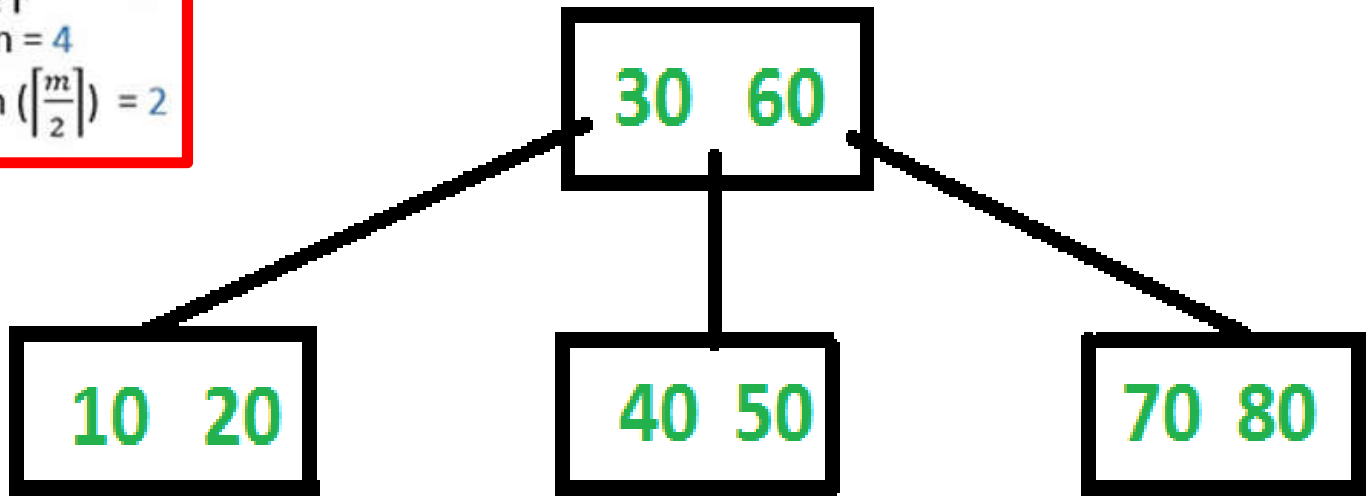
Formulas for reference

- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys $(\lceil \frac{m}{2} \rceil) - 1 = 1$
- Maximum Children = 4
- Minimum Children $(\lceil \frac{m}{2} \rceil) = 2$

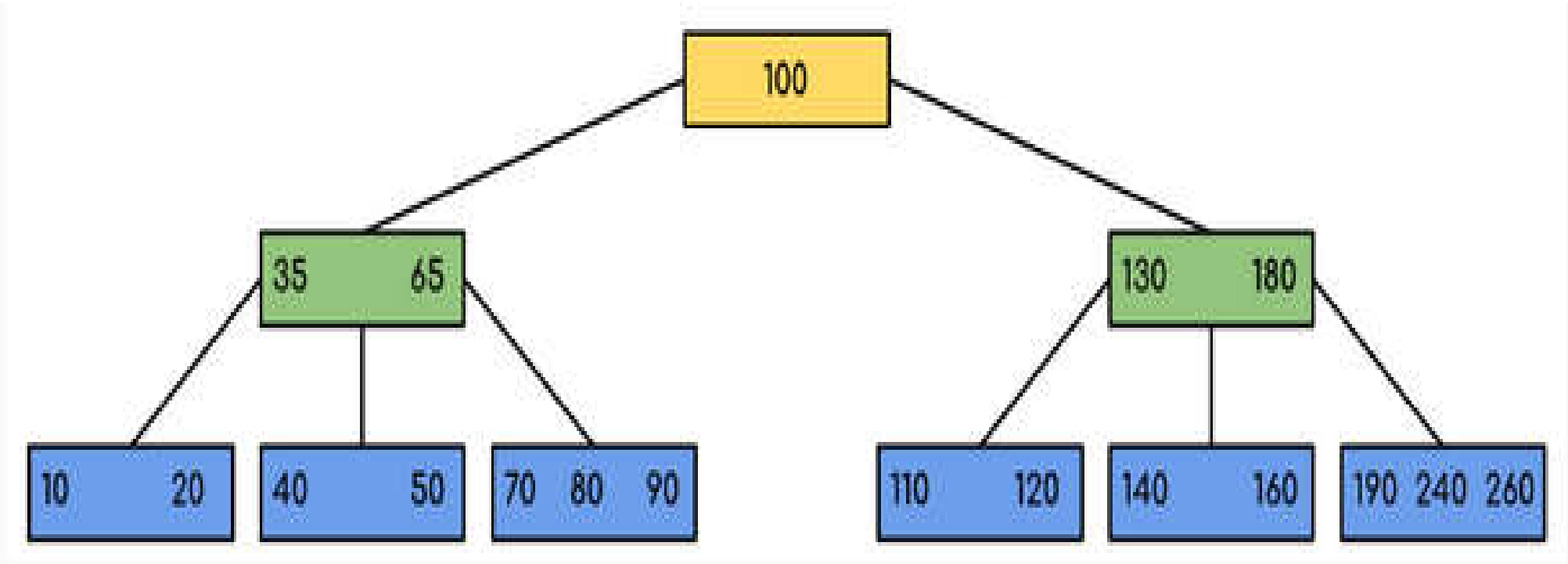
Ans for 10,20,30,40,50,60,70,80 if ORDER m=3

Formulas for reference

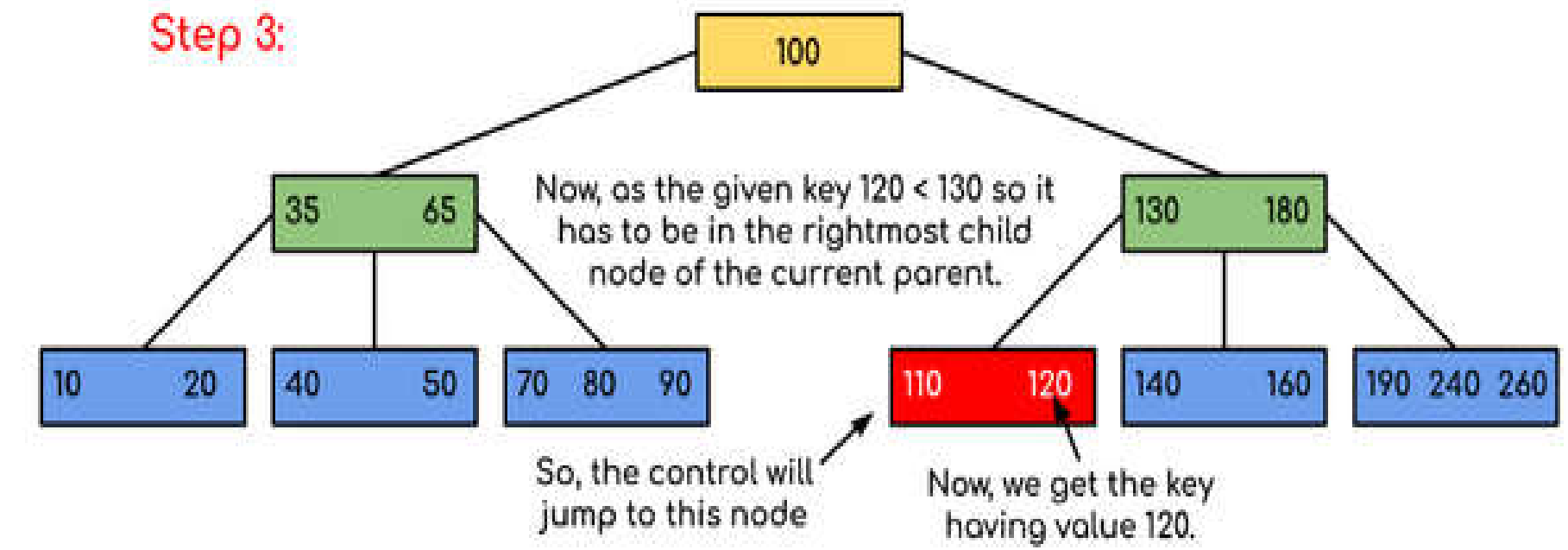
- Order (m) = 4
- Maximum Keys (m – 1) = 3
- Minimum Keys ($\lceil \frac{m}{2} \rceil$) – 1 = 1
- Maximum Children = 4
- Minimum Children ($\lceil \frac{m}{2} \rceil$) = 2



Search 120 in the given B-Tree.



Search 120 in the given B-Tree.



Deletion Operation

Before going through the steps below, one must know these facts about a B tree of degree **m**. **ORDER IS m=3**

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)

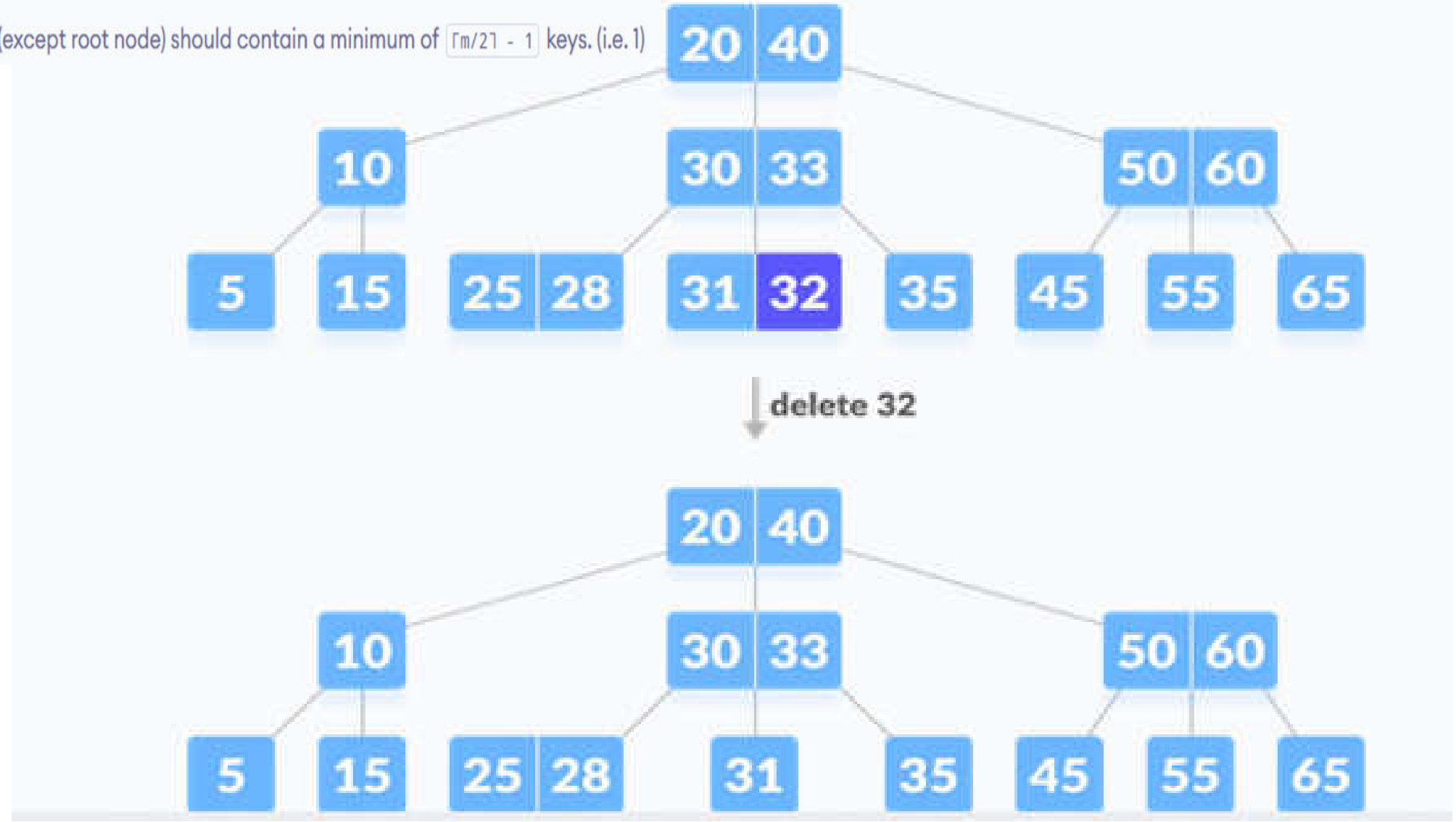
Case I

The key to be deleted lies in the leaf. There are two cases for it.

1. The deletion of the key does not violate the property of the minimum number of keys a node should hold.

A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)



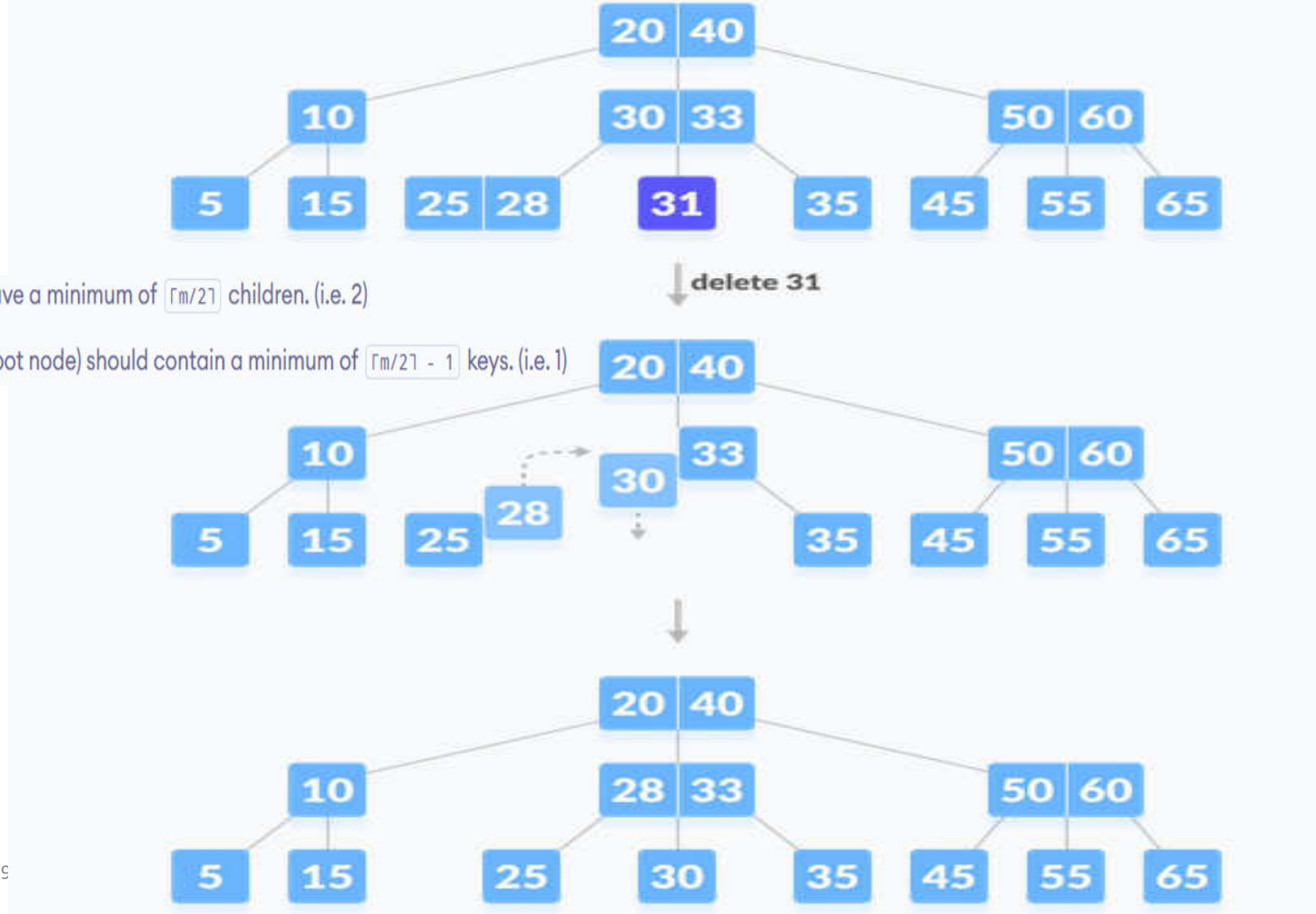
2. The deletion of the key violates the property of the minimum number of keys a node should hold. In this case, we borrow a key from its immediate neighboring sibling node in the order of left to right.

First, visit the immediate left sibling. If the left sibling node has more than a minimum number of keys, then borrow a key from this node.

Else, check to borrow from the immediate right sibling node.

A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

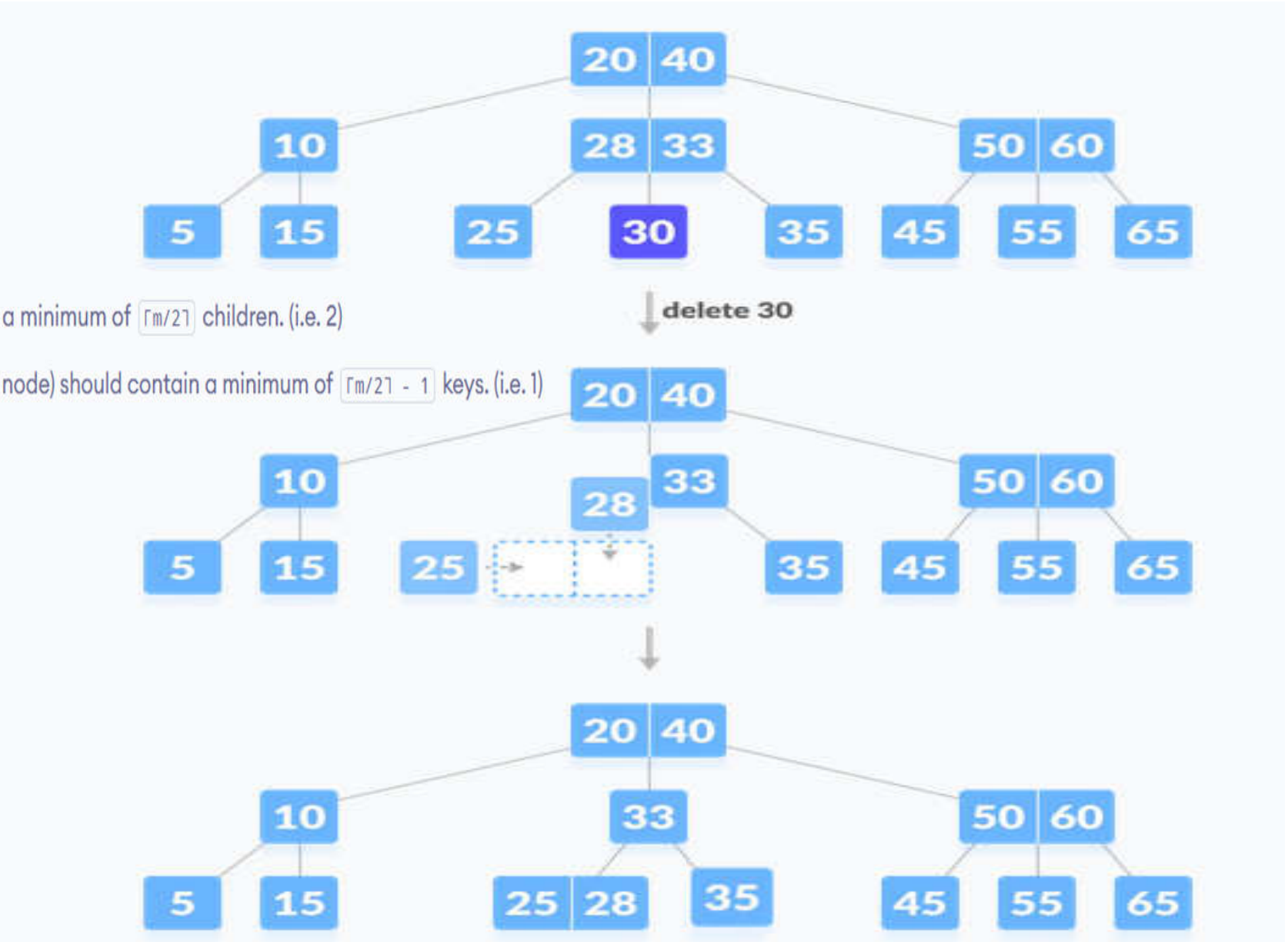
A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)



If both the immediate sibling nodes already have a minimum number of keys, then merge the node with either the left sibling node or the right sibling node. **This merging is done through the parent node.**

A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)



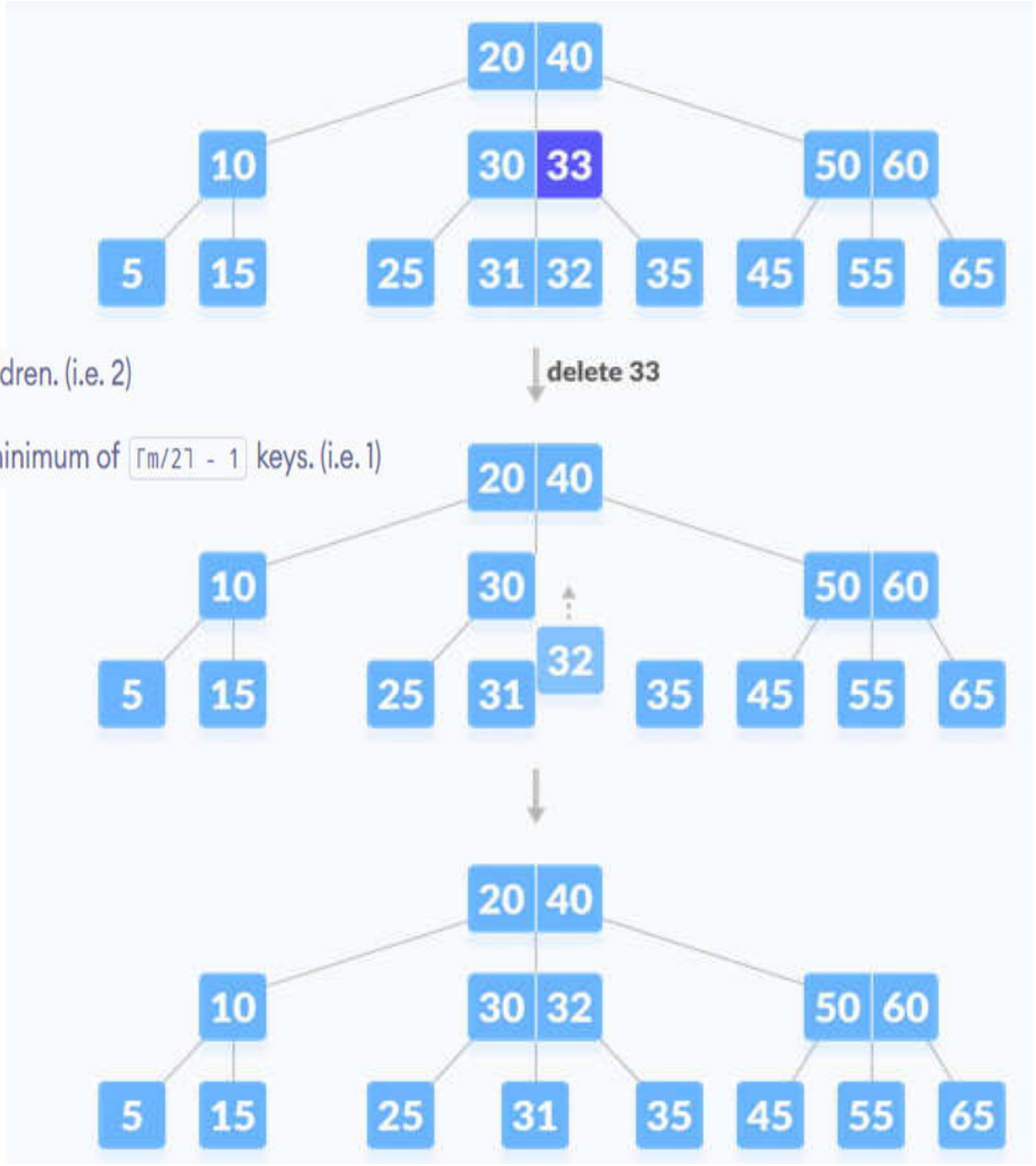
Case II

If the key to be deleted lies in the internal node, the following cases occur.

1. The internal node, which is deleted, is replaced by an inorder predecessor if the left child has more than the minimum number of keys.

A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)

A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)

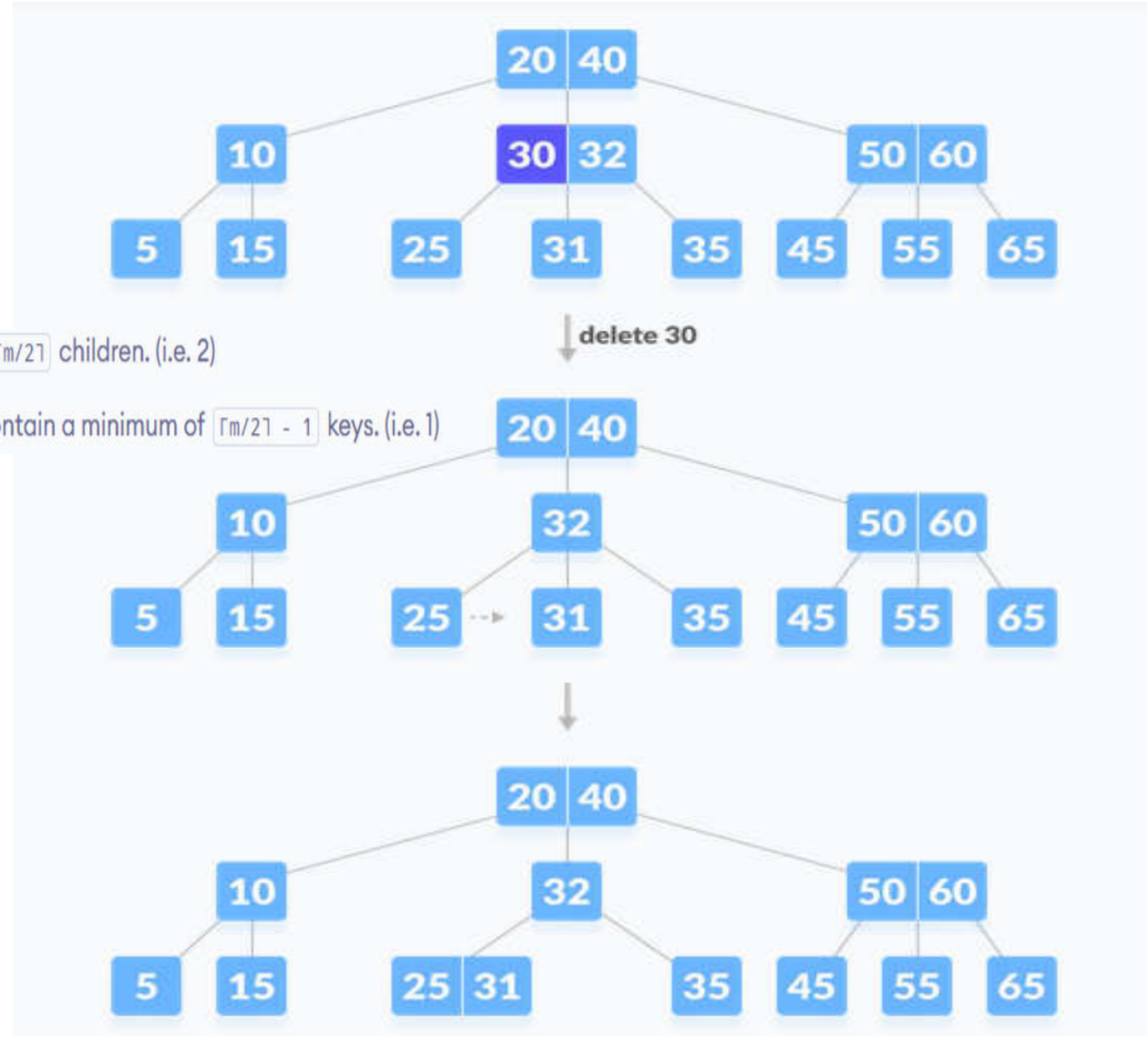


2. The internal node, which is deleted, is replaced by an inorder successor if the right child has more than the minimum number of keys.

3. If either child has exactly a minimum number of keys then, merge the left and the right children.

A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)

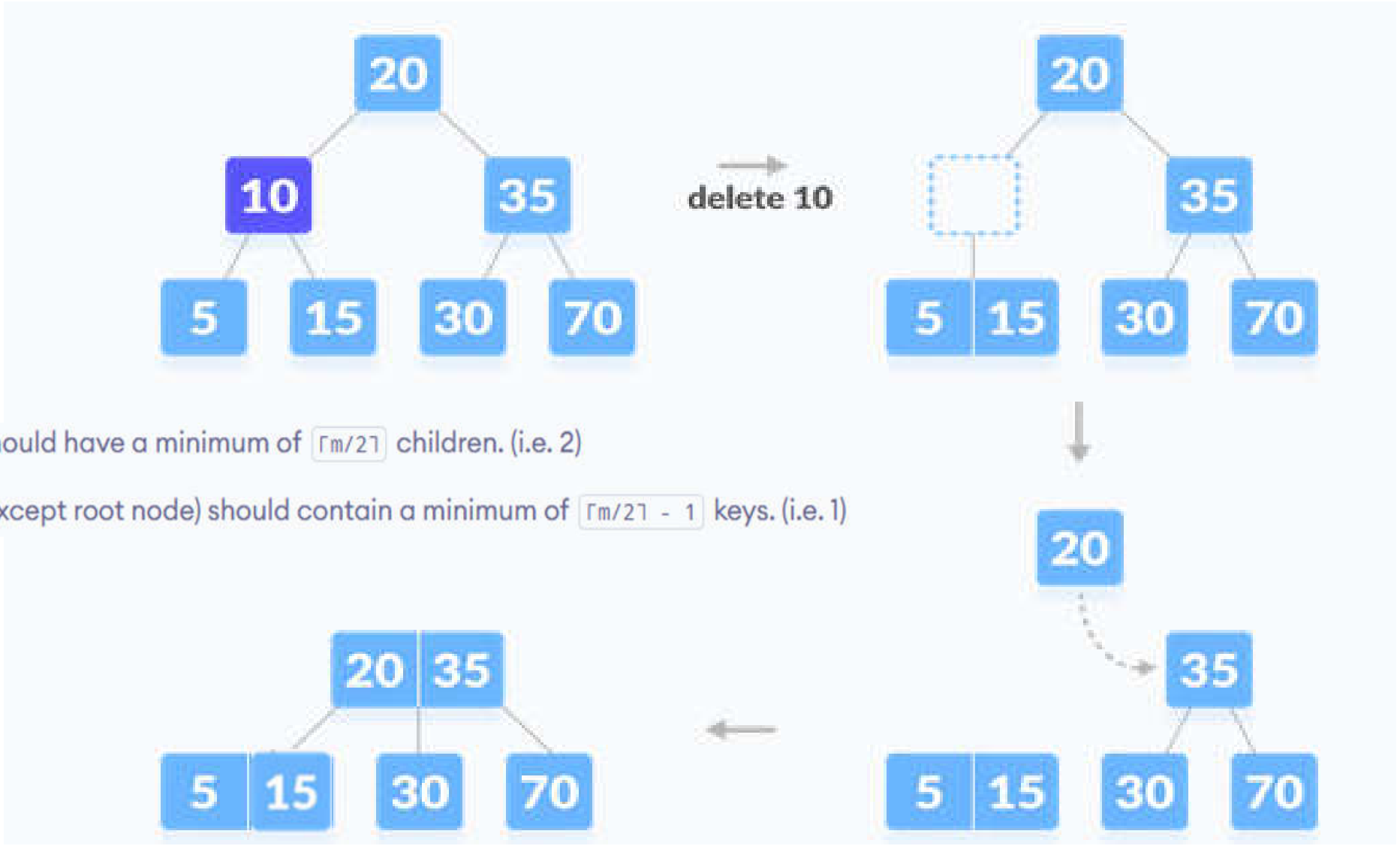
A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)



Case III

In this case, the height of the tree shrinks. If the target key lies in an internal node, and the deletion of the key leads to a fewer number of keys in the node (i.e. less than the minimum required), then look for the inorder predecessor and the inorder successor. If both the children contain a minimum number of keys then, borrowing cannot take place. This leads to Case II(3) i.e. merging the children.

Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).



A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)

For 'B' Tree if $m = 5$

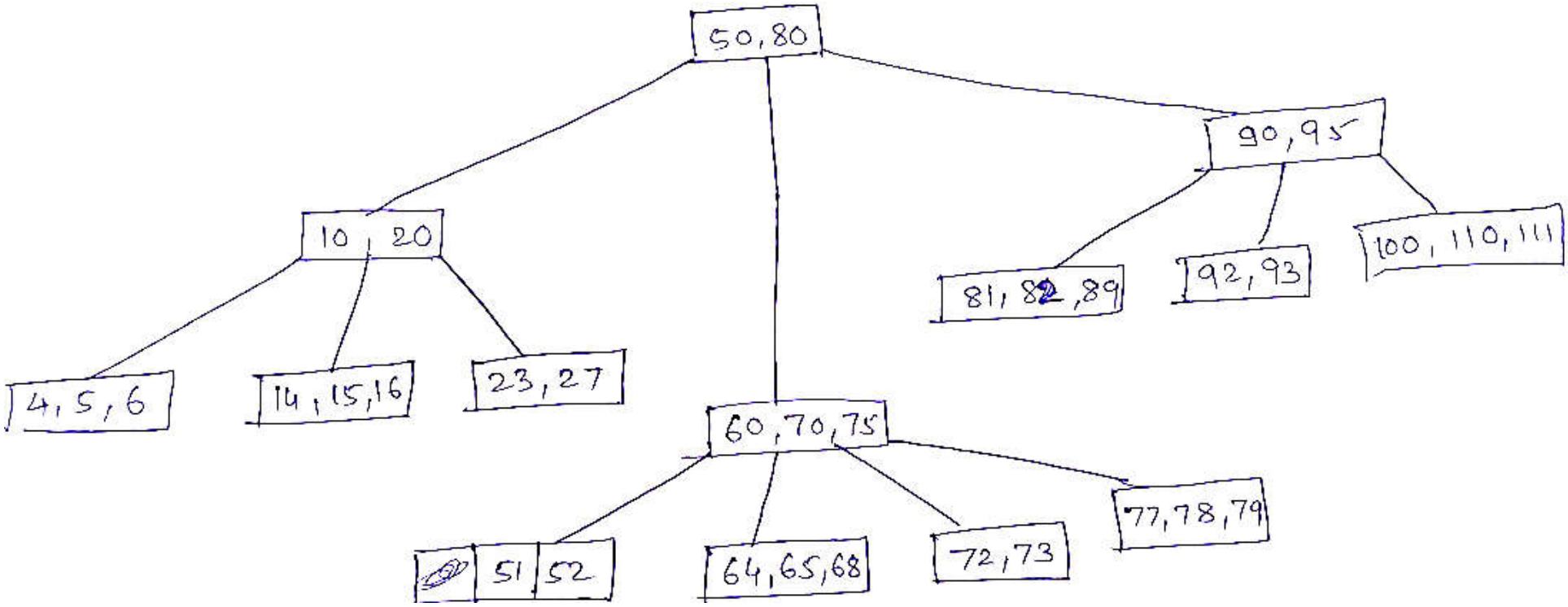
$\min^m$ children $= \lceil \frac{m}{2} \rceil = 3$

$\max^m$ children $= m = 5$

$\min^m$ key $= \lceil \frac{m}{2} \rceil - 1 = 2$

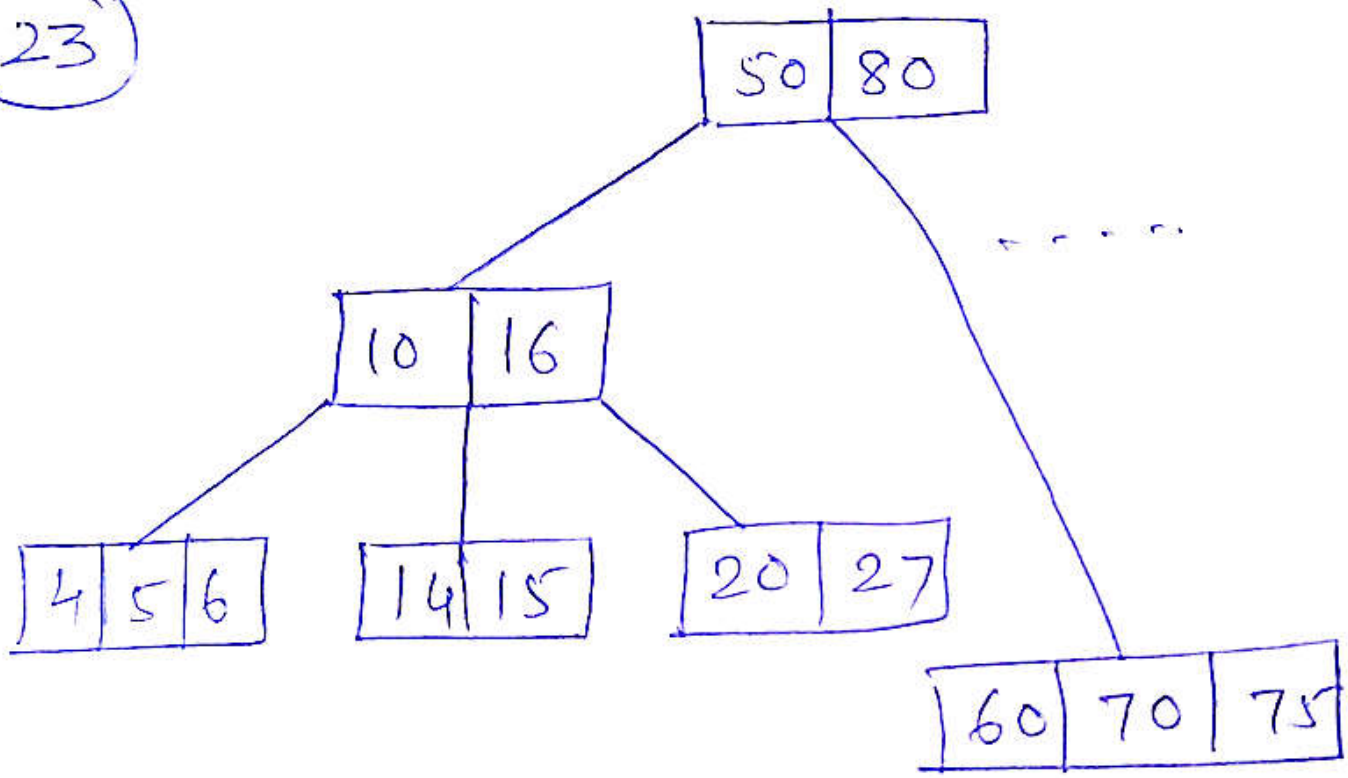
$\max^m$ key $= m - 1 = 4$

* Delete 64, 23, 72, 65, 20 & obtain resultant 'B' Tree



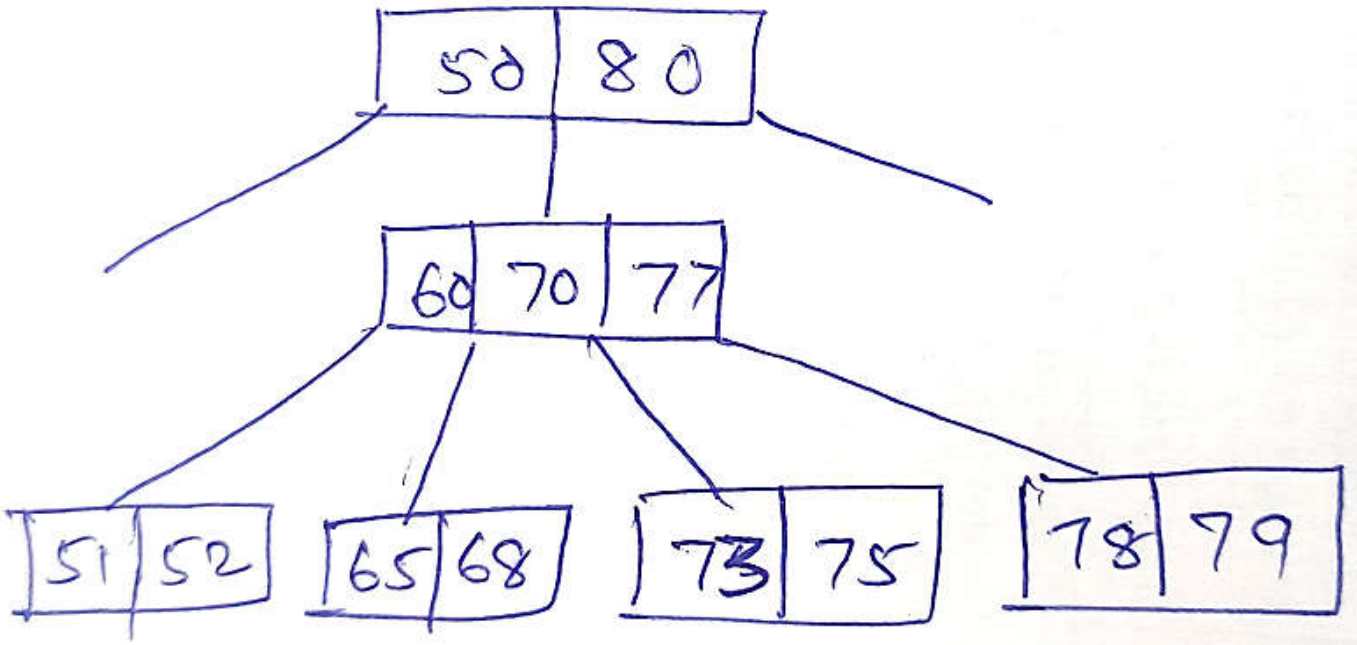
Case - 2

(23)



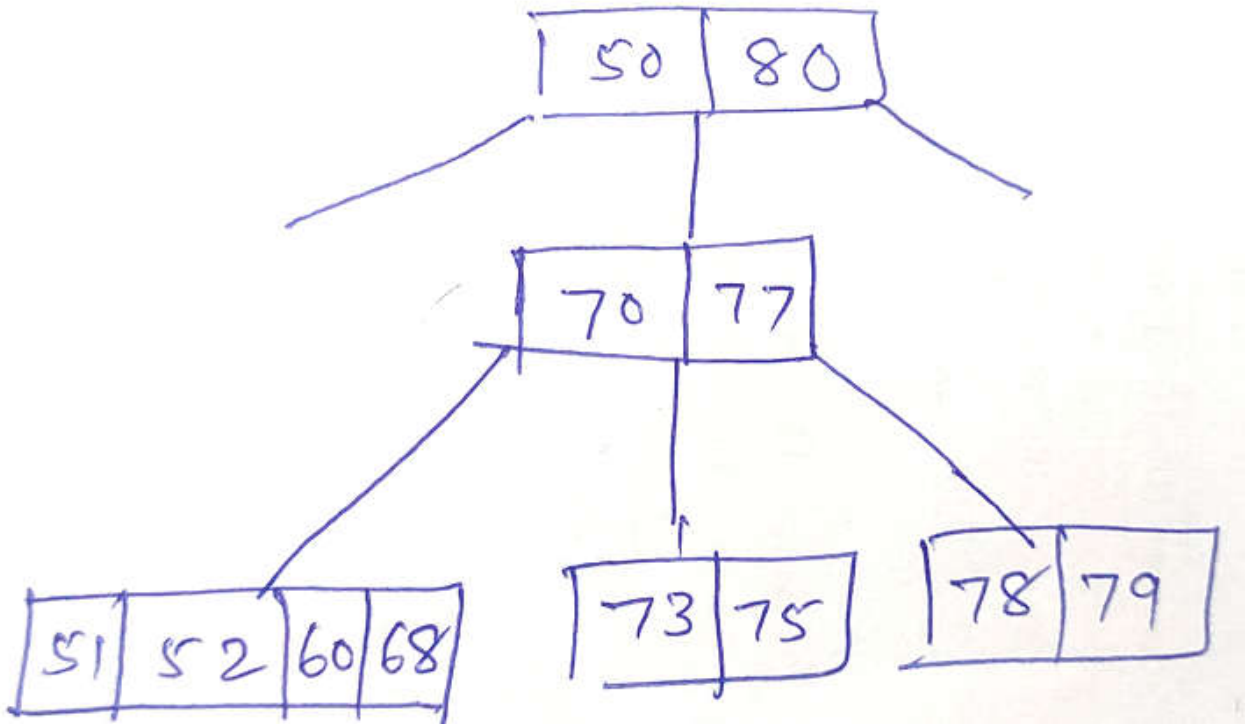
Case - 3

72

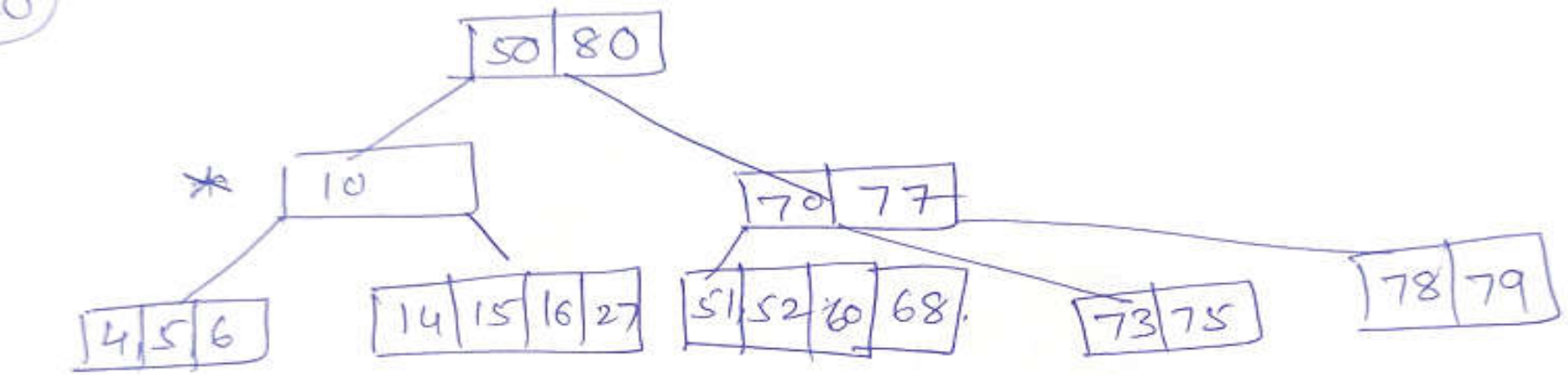


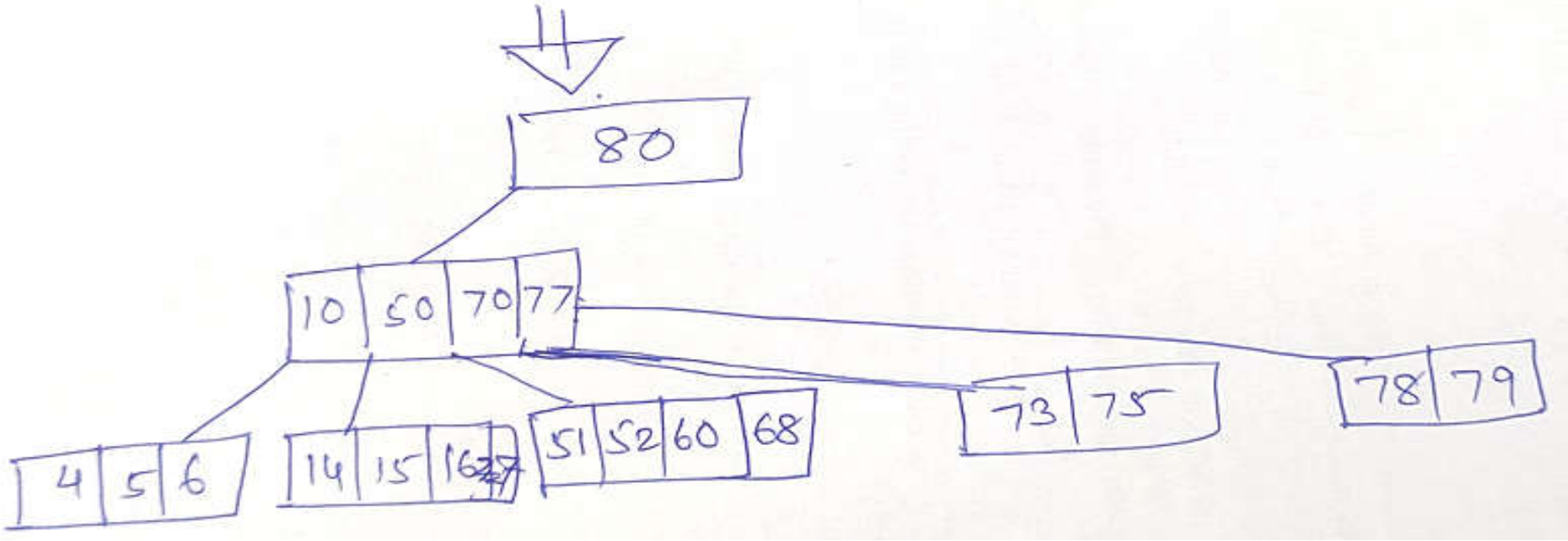
Case - 4

65



(20) *


50 80
*
10
70 77
4 5 6
14 15 16 27
51 52 60 68
73 75
78 79


80
10 50 70 77
4 5 6
14 15 16 27
51 52 60 68
73 75
78 79

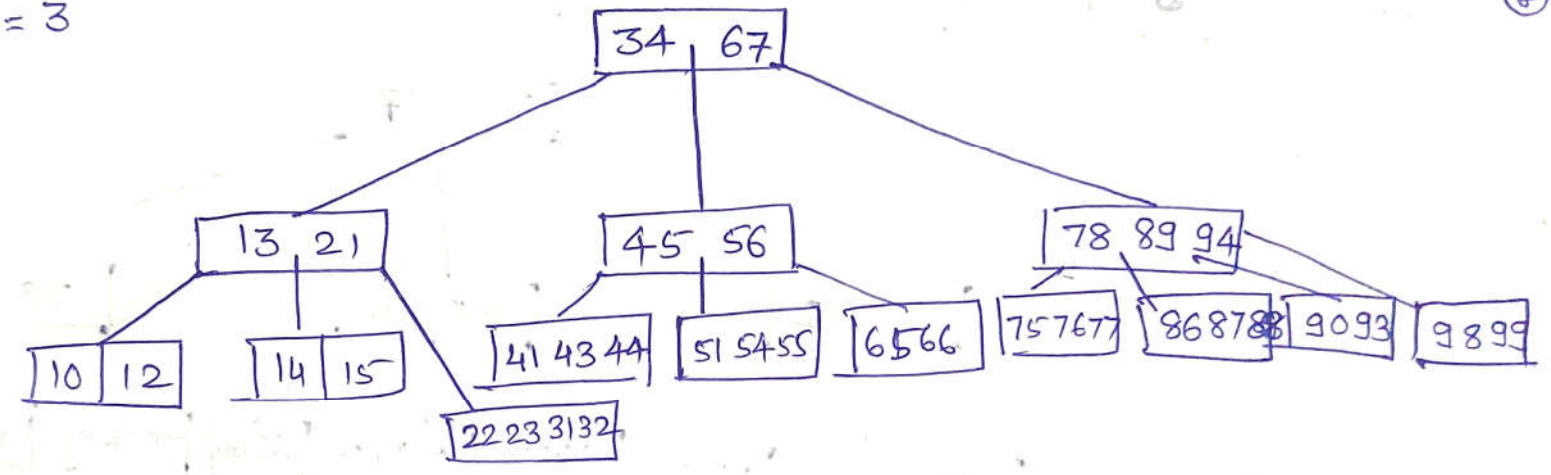
For B Tree with $m = 5$

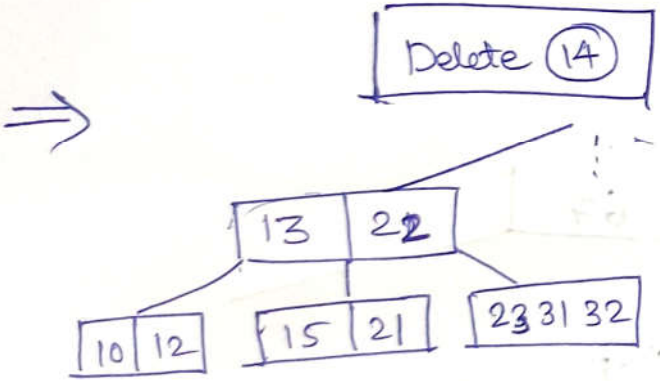
1) Min Children $\lceil \frac{m}{2} \rceil = \lceil \frac{5}{2} \rceil = 3$

2) Min Keys $\lceil \frac{m}{2} \rceil - 1 = 2$

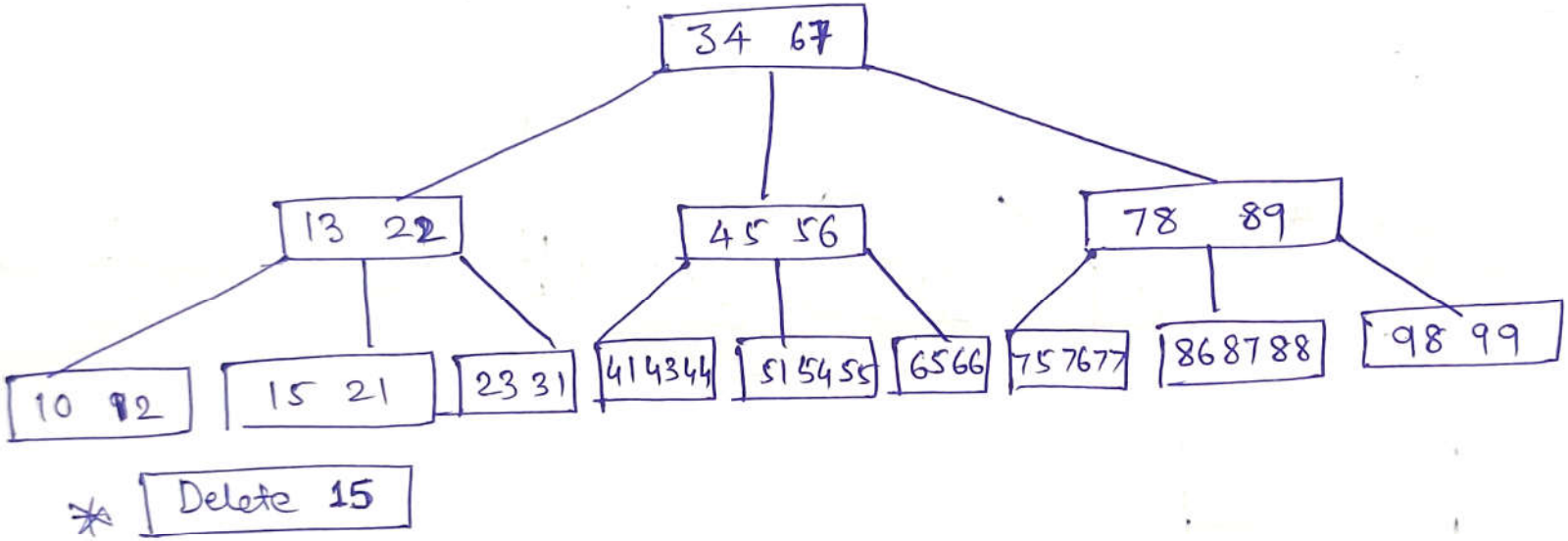
3) Max Children = 5

4) Max Key = 5 - 1 = 4

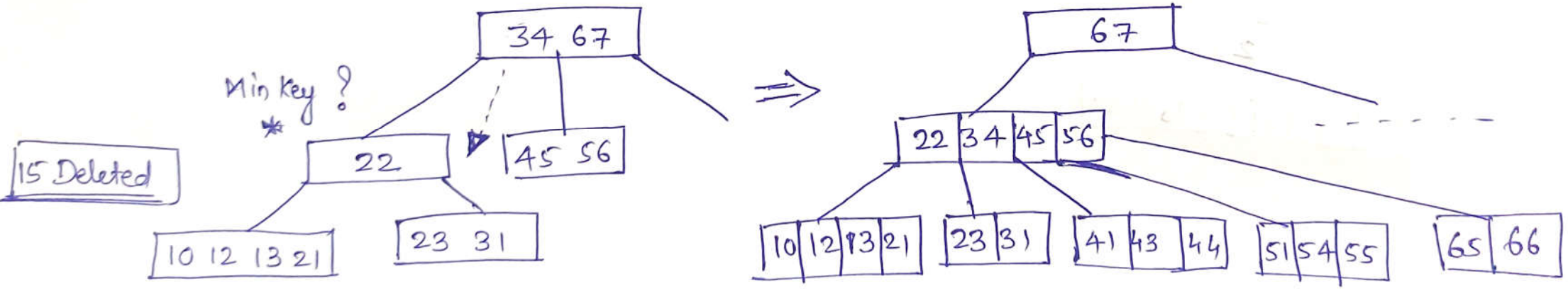




* For B Tree with order m=5



* Delete 15



Advantages of B-Trees:

• B-Trees have a guaranteed time complexity of O(log n) for basic operations like insertion, deletion, and searching, which makes them suitable for large data sets and real-time applications.

• B-Trees are self-balancing.

•High-concurrency and high-throughput.

•Efficient storage utilization.

Disadvantages of B-Trees:

•B-Trees are based on disk-based data structures and can have a high disk usage.

•Not the best for all cases.

•Slow in comparison to other data structures.

Parameters	B+ Tree	B Tree
Structure	Separate leaf nodes for data storage and internal nodes for indexing	Nodes store both keys and data values
Leaf Nodes	Leaf nodes form a linked list for efficient range-based queries	Leaf nodes do not form a linked list
Order	Higher order (more keys)	Lower order (fewer keys)
Key Duplication	Typically allows key duplication in leaf nodes	Usually does not allow key duplication
Disk Access	Better disk access due to sequential reads in a linked list structure	More disk I/O due to non-sequential reads in internal nodes

Applications	Database systems, file systems, where range queries are common	In-memory data structures, databases, general-purpose use
Performance	Better performance for range queries and bulk data retrieval	Balanced performance for search, insert, and delete operations
Memory Usage	Requires more memory for internal nodes	Requires less memory as keys and values are stored in the same node

CREATE INDEX Syntax (Duplicate values are allowed)

CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

CREATE UNIQUE INDEX Syntax (Duplicate values are not allowed**)**

CREATE UNIQUE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

DROP INDEX Statement

DROP INDEX *index_name*; IN ORACLE

ALTER TABLE *table_name*
DROP INDEX *index_name*; IN MYSQL

OLTP systems Online Transaction Processing	OLAP systems Online Analytical processing
Enable the real-time execution of large numbers of database transactions by large numbers of people	Usually involve querying many records (even all records) in a database for analytical purposes
Require lightning-fast response times	Require response times that are orders of magnitude slower than those required by OLTP
Modify small amounts of data frequently and usually involve a balance of reads and writes	Do not modify data at all; workloads are usually read-intensive
Use indexed data to improve response times	Store data in columnar format to allow easy access to large numbers of records
Require frequent or concurrent database backups	Require far less frequent database backup
Require relatively little storage space	Typically have significant storage space requirements, because they store large amounts of historical data
Usually run simple queries involving just one or a few records	Run complex queries involving large numbers of records

References used

Text Books:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", McGraw Hill Publishers
2. Connally T, Begg C., "Database Systems", Pearson Education
3. R. P. Mahapatra and Govind Verma, “Database Management Systems”, Khanna Publishing House

Reference Books:

1. Raghurama Krishan, “Database Management Systems”, McGrawHill
2. S.K.Singh, “Database Systems : Concepts, Design and Application”, Pearson, Education
3. Pramod J. Sadalage and Martin Fowler, “NoSQL Distilled”, Addison Wesley
4. Kristina Chodorow, Michael Dirolf, “MangoDB: The Definitive Guide”, O’Reilly Publications

e-Resources: https://nptel.ac.in/courses/106/105/106105175/ ,notes and video sessions of the experts of the respective fields.

Thank you !!!