BASIC DATABASES

View, Stored Procedure, Index

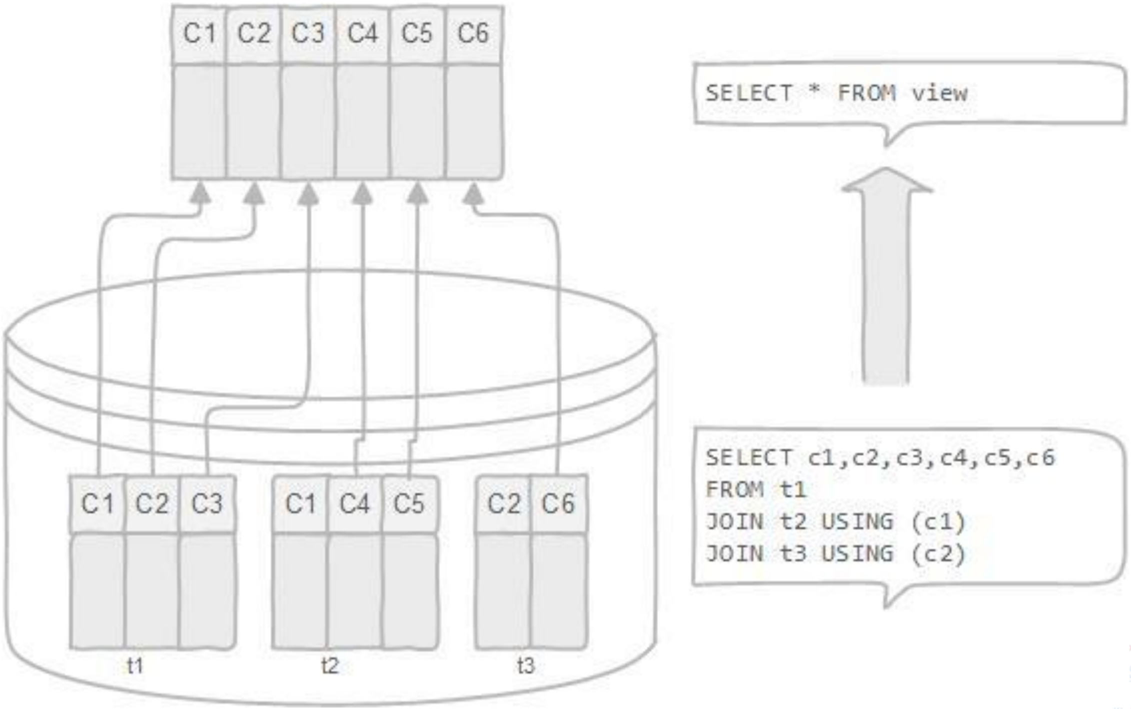
NGUYEN Hoang Ha

Email: nguyen-hoang.ha@usth.edu.vn

VIEW

View concept

- A view is a “virtual” or logical table that is derived from other tables



```
SELECT * FROM view
```

```
SELECT c1,c2,c3,c4,c5,c6
FROM t1
JOIN t2 USING (c1)
JOIN t3 USING (c2)
```

Pros vs. Cons

- Pros:
  - Simplify complex queries
  - Enable computed columns
  - Provide a security layer: hide sensitive data
  - Enable backward capability
- Cons
  - Performance
  - Table dependency: table changes → need to change views

Syntax

```
CREATE [ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view_name [(column_list)]
AS
select-statement [WITH CHECK OPTION];
```

- ALGORITHM
  - MERGE: MySQL combines input query with the select-statement. MERGE is not allowed if the SELECT statement contains aggregate functions or `DISTINCT, GROUP BY, HAVING, LIMIT, UNION, UNION ALL`, subquery, SELECT statement refers to no table.
  - TEMPTABLE: MySQL creates a temporary table based on the SELECT statement that defines the view, then performs query against this temporary table.
  - UNDEFINED: MySQL makes choice of MERGE or TEMTABLE.

View examples

- Computed columns

```
CREATE VIEW sale_per_order AS
SELECT order_id, SUM(quantity * unit_price * (1-discount)) total
FROM order_details
GROUP BY order_id
ORDER BY total DESC;
```

- Based on a sub query

```
CREATE VIEW above_avg_products AS
SELECT product_code, product_name, list_price
FROM products
WHERE list_price > (SELECT AVG(list_price)
FROM products)
ORDER BY list_price DESC;
```

- Based on another view

```
CREATE VIEW big_sale_orders AS
SELECT order_id, ROUND(total,2) AS total
FROM sale_per_order
WHERE total > 1000;
```

Updatable views

- SELECT statement defining the view must not contain following elements:
  - Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
  - DISTINCT
  - GROUP BY clause.
  - HAVING clause.
  - UNION or UNION ALL clause.
  - Left join or outer join.
  - Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
  - Reference to non-updatable view in the FROM clause.
  - Reference only to literal values.
  - Multiple references to any column of the base table

WITH CHECK OPTION Clause

- Role: to prevent updating or inserting rows that are not visible through the view
- Example

```
CREATE OR REPLACE VIEW northwind_products
AS
SELECT id, product_code, product_name
FROM products
WHERE product_name LIKE 'Northwind%'
WITH CHECK OPTION;

INSERT INTO northwind_products (product_code, product_name)
VALUES('HNB', 'Hanoi Beer');-- This is invalid

INSERT INTO northwind_products (product_code, product_name)
VALUES('NWnew', 'Northwind Beer');

UPDATE northwind_products
SET product_name = 'Nwd beer'
WHERE product_code = 'NWnew'; --- WITH CHECK OPTION will prevent this statement from running
```

View management

- Show view definition

  -
    ```
    SHOW CREATE VIEW [database_name].[view_ name];
    ```
- Delete view:

  -
    ```
    DROP VIEW [IF EXISTS] view_name
    ```
- Change view

  ```
  ALTER[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
  VIEW view_name [(column_list)]
  AS
  select-statement [WITH CHECK OPTION];
  ```

  - OR: CREATE OR REPLACE VIEW
- List all views with updateable information

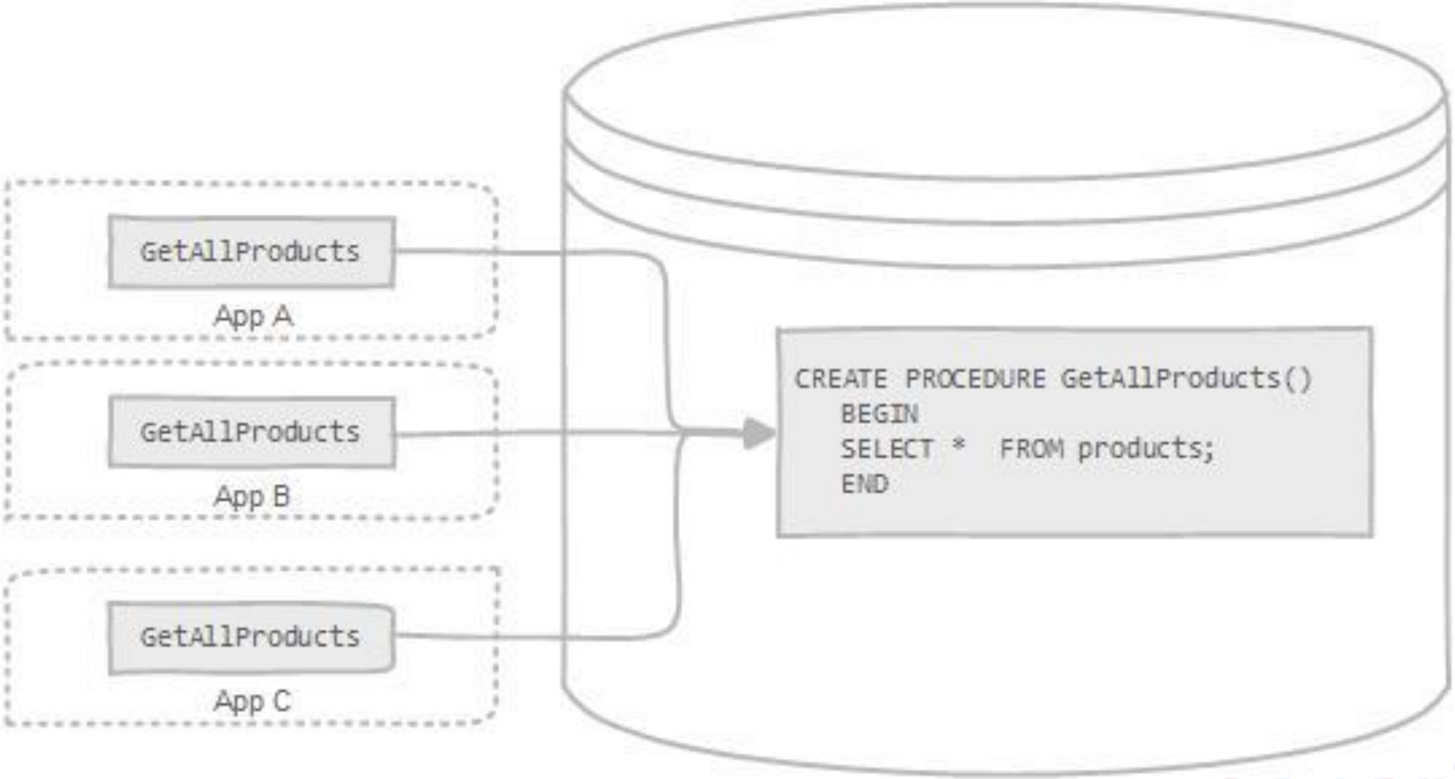
  ```
  SELECT table_name, is_updatable

  FROM information_schema.views
  ```

STORED PROCEDURE

Concept

- A stored procedure is a segment of SQL statements stored inside the database catalog



```
CREATE PROCEDURE GetAllProducts()
   BEGIN
   SELECT *  FROM products;
   END
```

Pros vs. Cons

- Pros:
  - Better performance
  - Reduce traffic
  - Be reusable and transparent
  - Provide a secure way to access data
- Cons
  - CPU usage can increase if logical operators are overused
  - Hard to debug, maintain

Example

```
DROP PROCEDURE IF EXISTS count_products;
delimiter //
CREATE PROCEDURE count_products (OUT param1 INT)
BEGIN
   SELECT COUNT(*) INTO param1
   FROM products;
END//
delimiter ;

CALL count_products(@a); SELECT @a;
```

Input parameter

```
DELIMITER //
CREATE PROCEDURE get_customers_by_city(IN search_city nvarchar(255))
AS
BEGIN
        SELECT * FROM customers
        WHERE city = search_city;
END //
DELIMITER ;
```

```
CALL get_customers_by_city('Seattle');
```

INDEX

Why indexing?

- Indexing are one of the most important and useful tools for achieving high performance in a relational database
- Many database administrators consider indexes to be the single most critical tool for improving database performance
- An index is a data structure that contains a copy of some of the data from one or more existing database tables
- A database index provides an organizational framework that the DBMS can use to quickly locate the information that it needs
- This can vastly improve the speed with which SQL queries can be answered

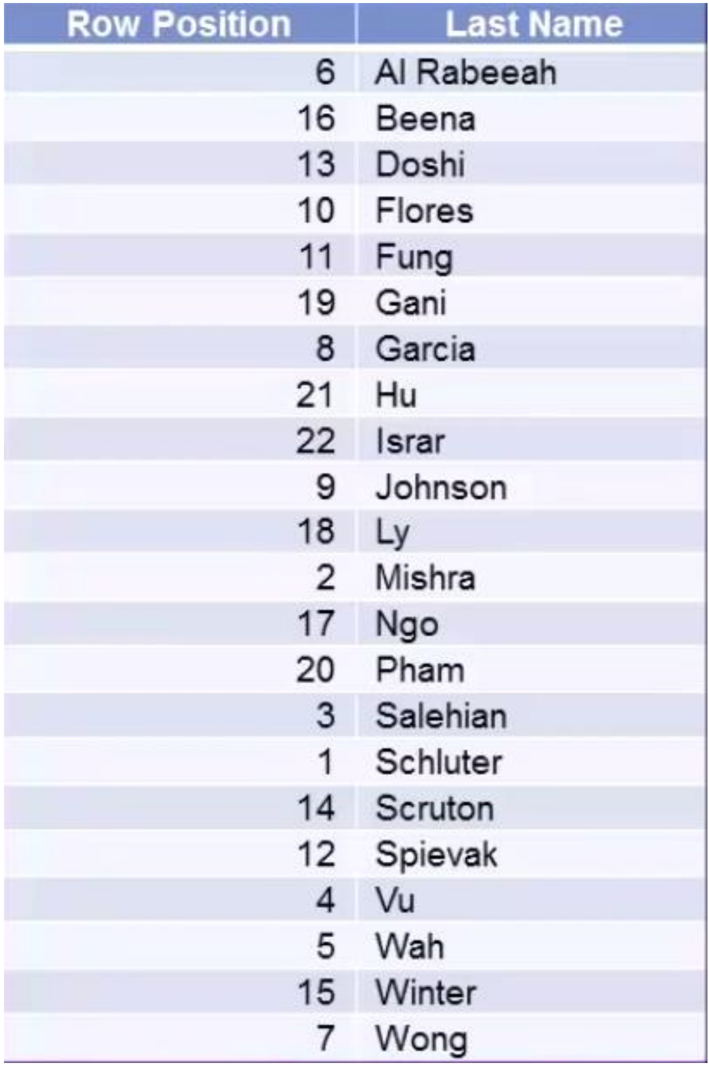
Without index

- Query for a random name within the table
- What is the average search time if the process is repeated many times?

$$average = \frac{n+1}{2}$$

- What is the maximum search time?

$$Maximum = n$$



Row Position	Last Name
6	Al Rabeeah
16	Beena
13	Doshi
10	Flores
11	Fung
19	Gani
8	Garcia
21	Hu
22	Israr
9	Johnson
18	Ly
2	Mishra
17	Ngo
20	Pham
3	Salehian
1	Schluter
14	Scruton
12	Spievak
4	Vu
5	Wah
15	Winter
7	Wong

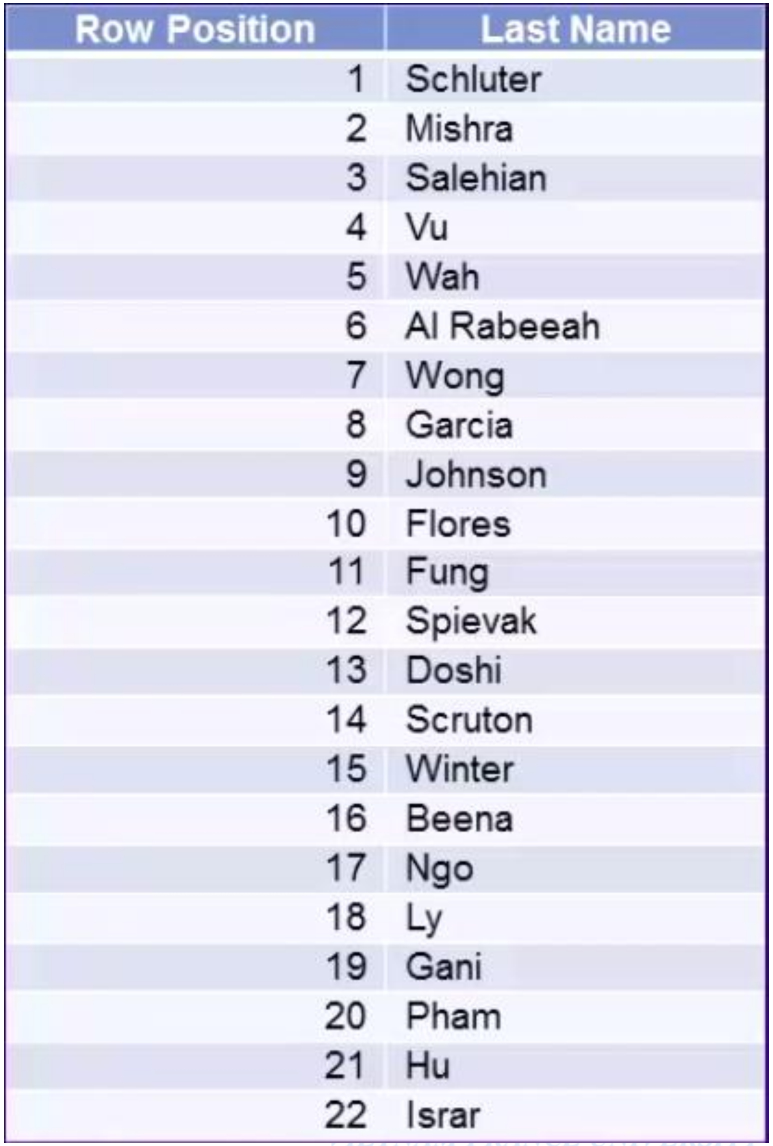
With index

- Query for a random name within the table
- What is the average search time if the process is repeated many times?

$$average = \log_2(n) - 1 = 3.5$$

- What is the maximum search time?

$$Maximum = \log_2(n) = 4.5$$



Row Position	Last Name
1	Schluter
2	Mishra
3	Salehian
4	Vu
5	Wah
6	Al Rabeeah
7	Wong
8	Garcia
9	Johnson
10	Flores
11	Fung
12	Spievak
13	Doshi
14	Scruton
15	Winter
16	Beena
17	Ngo
18	Ly
19	Gani
20	Pham
21	Hu
22	Israr

Index concepts

- Indexes are created on one or more columns in a table
  - For example:
    - An index is created on a PK column
    - The index will contain the PK value for each row in the table, along with each row's ordinal position (row number) within the table
    - When a query involving the PK is run, the DBMS will find the PK value within the index. The DBMS will then know the position of the row within the table
    - The DBMS can then quickly locate the row in the table that is associated with the PK value
- Without an index, the DBMS has to perform a table scan in order to locate the desired row

Index concepts

- An index can be created on most, but not all, columns. Whether an index can be created on a column depends on the column's datatype
- Columns with large object data types cannot be indexed without employing additional mechanisms These data types include:
  - Text
  - ntext
  - Image
  - varchar (max)
  - Nvarchar(max)
  - varbinary(max)

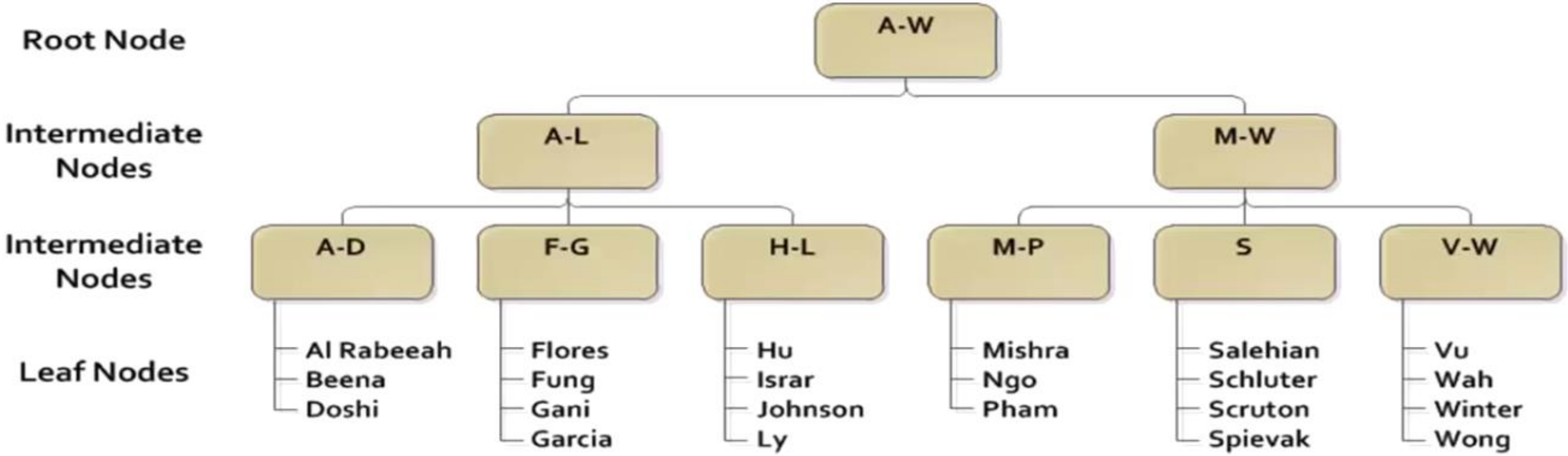
Index concepts

- Creating an index increases the amount of storage space required by the database
  - This occurs because an index contains a copy of some of the data in a table
  - To estimate the storage space requirements of an index, we can use the following formula:
- Number of rows in table x Average number of bytes required per row for the indexed columns

B-Tree Index

- Balance-Tree: the most common type of database indexing
- B-trees use pointers and several layers of nodes in order to quickly locate desired data
- Root node
- Intermediate nodes
- Leaf nodes
- When the DBMS processes a query which includes an indexed column, it starts at the root node of the B-tree and navigates downward until it finds the desired leaf

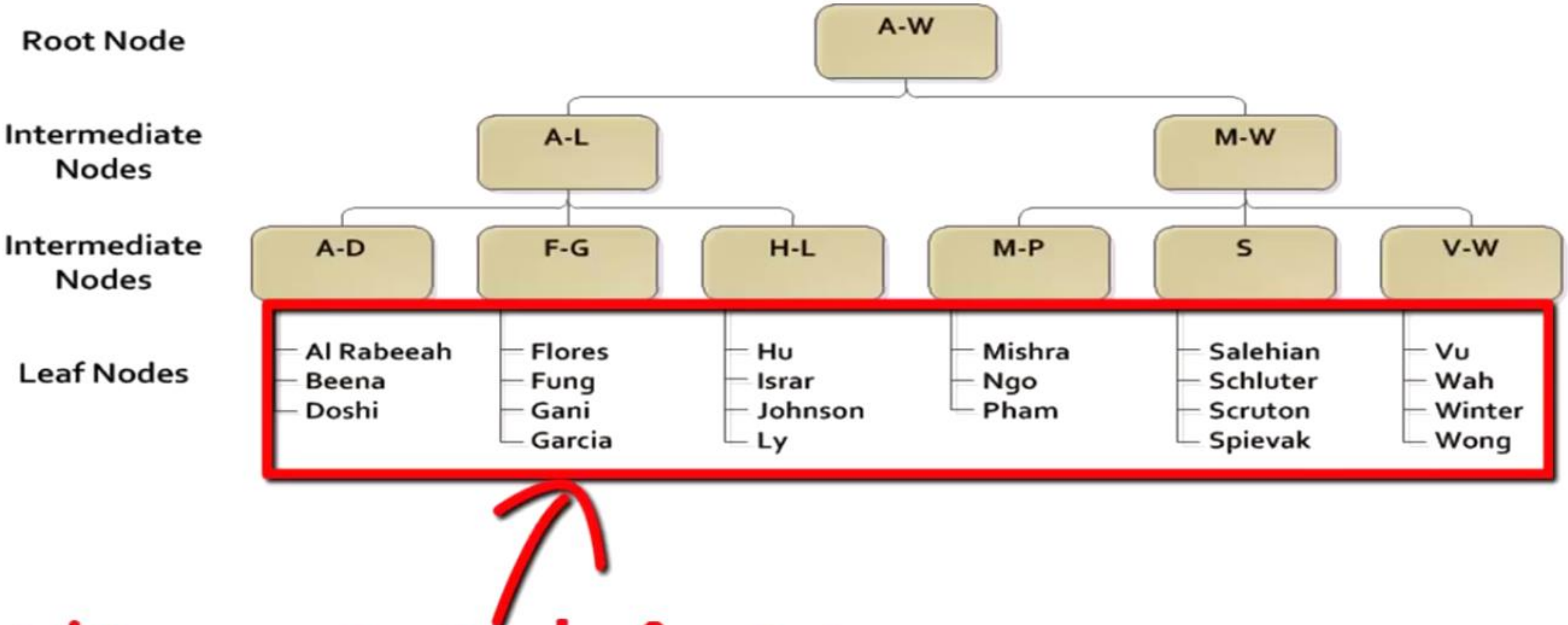
B-tree example



Clustered Indexes

- In a clustered index, the actual data rows that comprise the table are stored at the leaf level of the index
- The indexed values are stored in a sorted order
  - This means that there can be only one clustered index per table
  - PK columns are good candidates for clustered indexes

Clustered B-Tree Example



Root Node

Intermediate Nodes

Intermediate Nodes

Leaf Nodes

A-W

A-L

M-W

A-D

F-G

H-L

M-P

S

V-W

Al Rabeeah
Beena
Doshi

Flores
Fung
Gani
Garcia

Hu
Israr
Johnson
Ly

Mishra
Ngo
Pham

Salehian
Schluter
Scruton
Spievak

Vu
Wah
Winter
Wong

table rows instead of pointers

Indexing Guidelines

- If a table is heavily updated, index as few columns as possible
- If a table is updated rarely, use as many indexed columns as necessary to achieve maximum query performance
- Clustered indexes are best used on columns that do not allow null values and whose value are unique
- The performance benefits on an index are related to the uniqueness of the values in the indexed column
  - Index performance is poor when an indexed column contains a large proportion of duplicate values
  - Index performance is best when an indexed column contains unique values