Data Storage and Indexing

Index Classification

Ira A. Fulton Schools of Engineering
Arizona State University

Objectives

Objective

Identify major indexing schemes in Database Systems

Index Classification

Primary Vs. Secondary

If search key contains primary key, then called primary index.

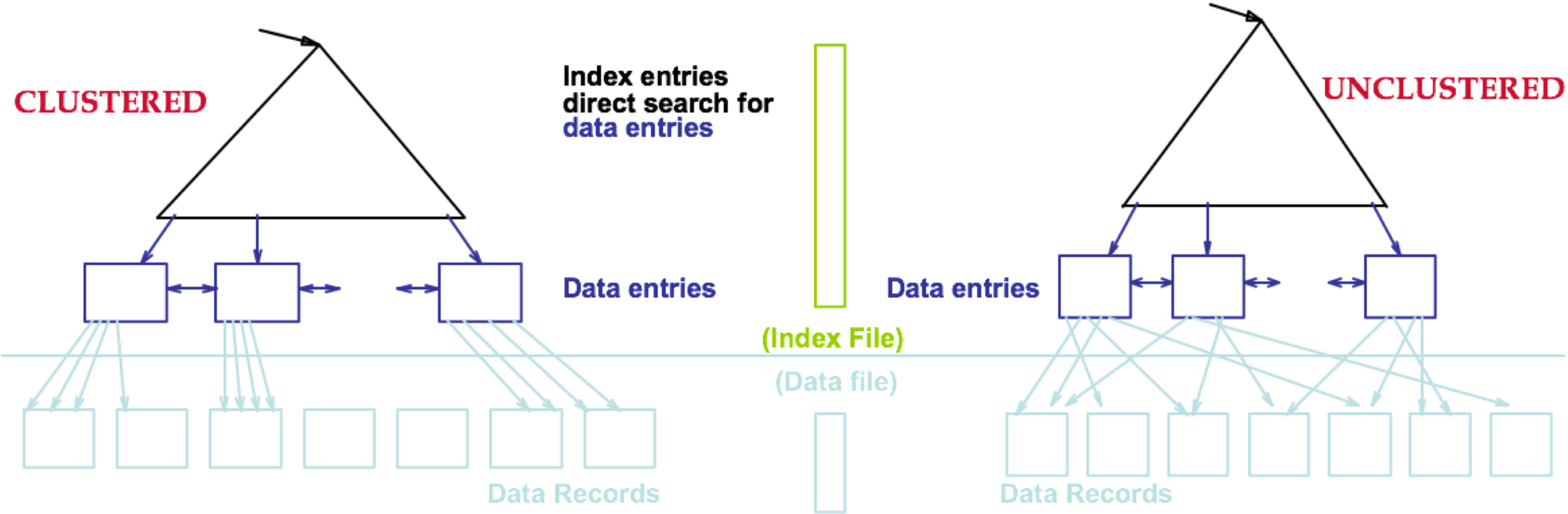
- Unique index: Search key contains a candidate key.

Clustered vs. Unclustered

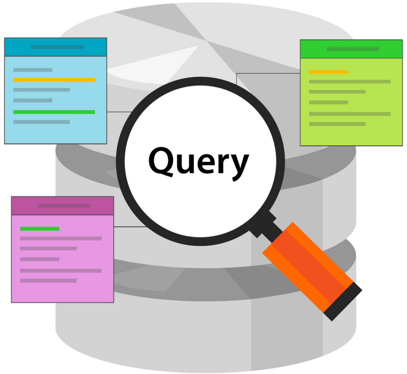
If order of data records is the same as, or `close to’ , order of data entries, then called clustered index.

- A file can be clustered on at most one search key.

Clustered vs. Unclustered Index



Understanding the Workload



For each query in the workload:

- Which relations does it access?
- Which attributes are retrieved?

- Which attributes are involved in selection/join conditions?
- How selective are these conditions likely to be?

Understanding the Workload

For each update in the workload:

- Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

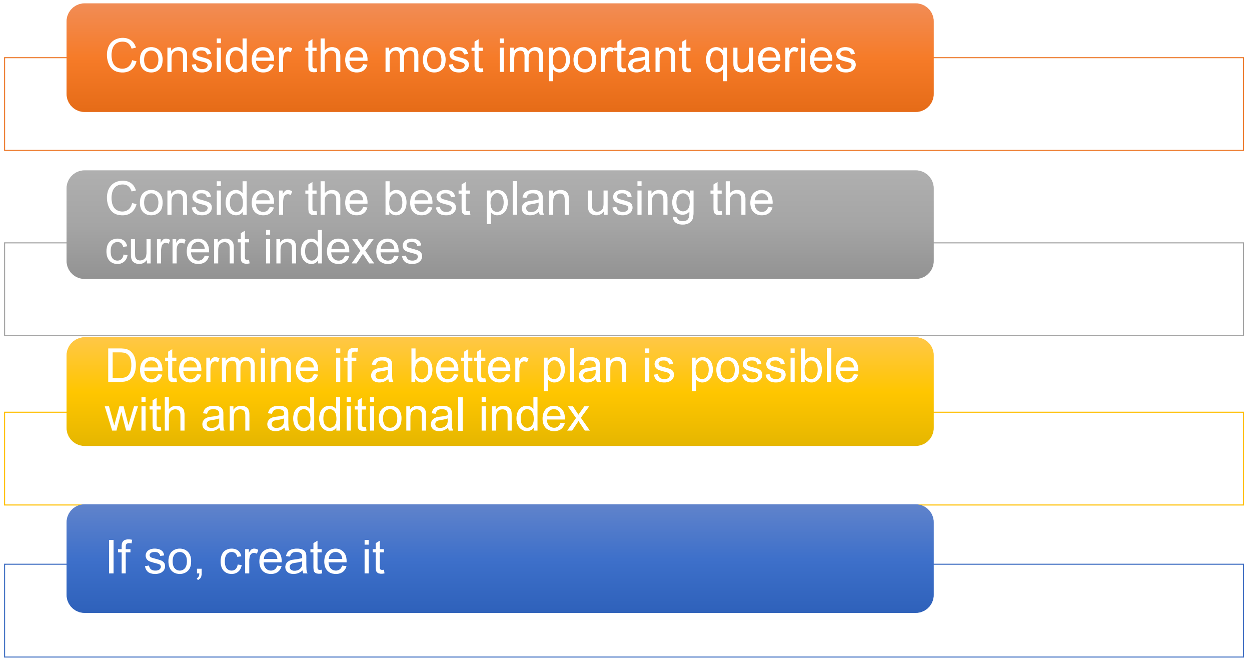
Choices of Indexes

- What indexes should you create?

- Which relations should have indexes?
- What field(s) should be the search key?
- Should you build several indexes?

- For each index, what kind of an index should it be?

Creating a New Index



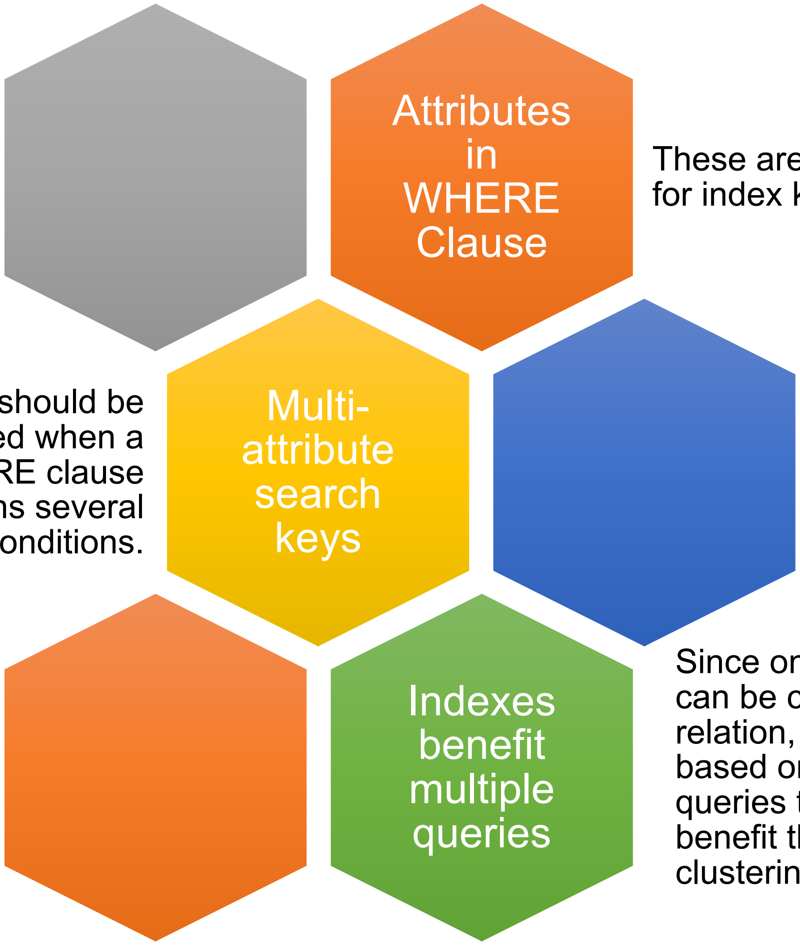
Consider the most important queries

Consider the best plan using the current indexes

Determine if a better plan is possible with an additional index

If so, create it

Index Selection Guidelines



Attributes in WHERE Clause

These are candidates for index keys

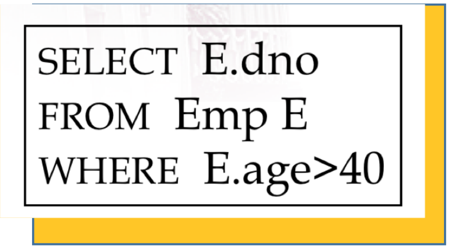
Multi-attribute search keys

Keys should be considered when a WHERE clause contains several conditions.

Indexes benefit multiple queries

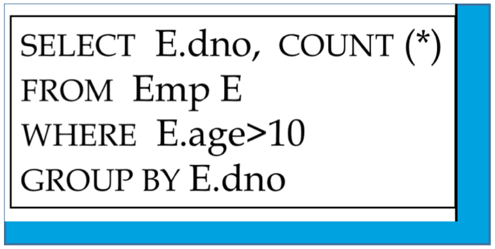
Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Examples of Clustered Indexes



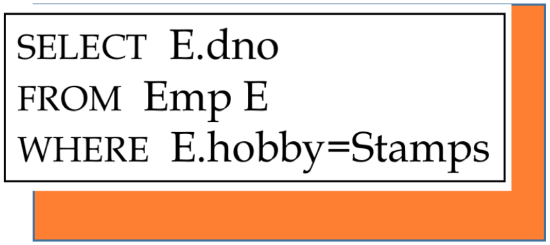
```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

B+ tree index on E.age can be used to get qualifying tuples.



```
SELECT  E.dno,  COUNT (*)
FROM  Emp E
WHERE  E.age>10
GROUP BY E.dno
```

Consider the GROUP BY query.

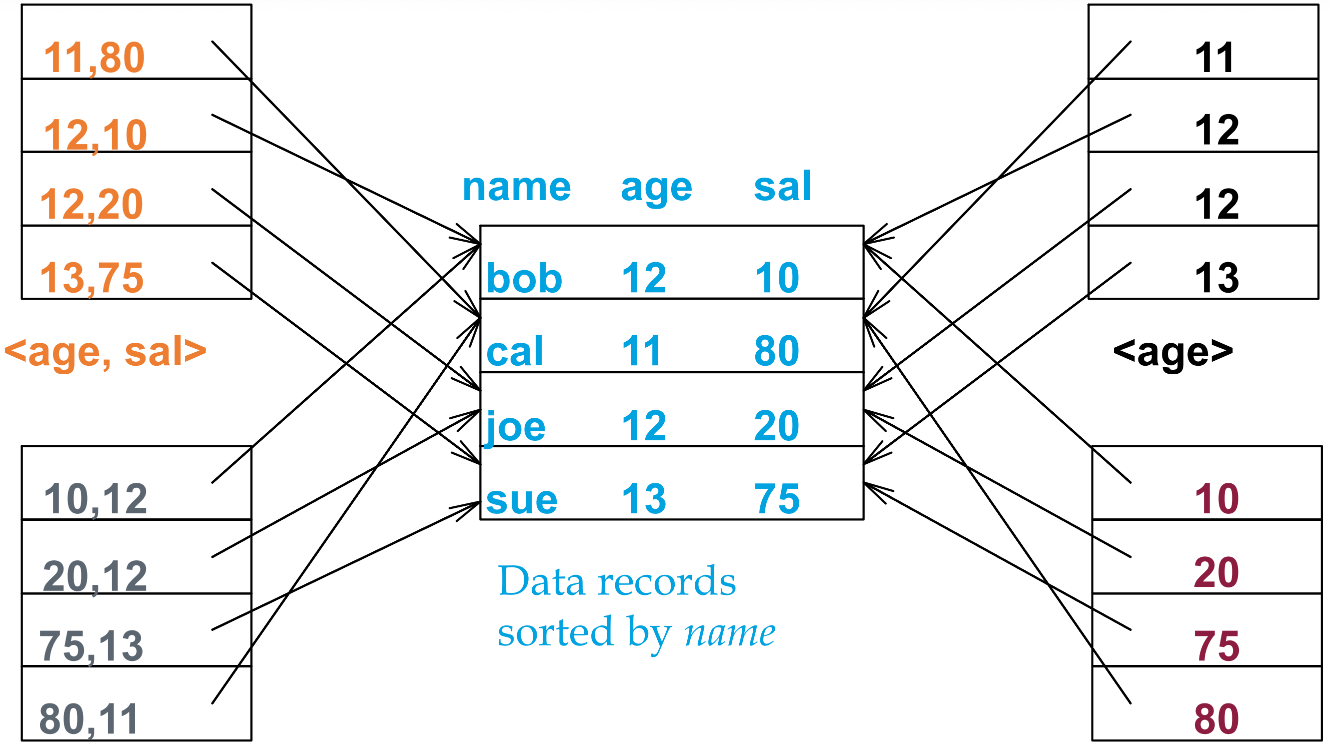


```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby=Stamps
```

Equality queries and duplicates:

- Clustering helps!

Indexes with Composite Search Keys



name	age	sal
bob	12	10
cal	11	80
joe	12	20
sue	13	75

Data records sorted by *name*

<sal, age>

Data entries in index sorted by <*sal,age*>

<sal>

Data entries sorted by <*sal*>

Composite Search Keys

Orthogonal to Clustering

- To retrieve Emp records with:
    - age=30 AND sal=4000,
    - an index on <*age,sal*>

Clustered

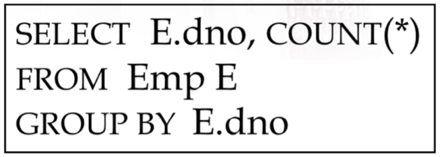
- If condition is: *age*=30 AND 3000<*sal*<5000
- Clustered <*age,sal*> index much better than <*sal,age*> index

Clustered Tree

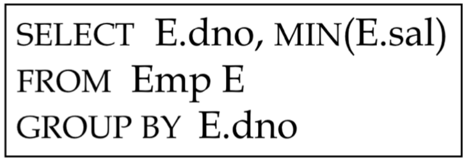
- If condition is: 20<*age*<30 AND 3000<*sal*<5000
    - Index on <*age,sal*> or <*sal,age*>

Index Only Plans

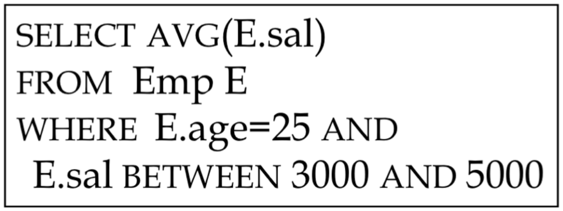
A number of queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available.



```
SELECT  E.dno, COUNT(*)
FROM  Emp E
GROUP BY  E.dno
```



```
SELECT  E.dno, MIN(E.sal)
FROM  Emp E
GROUP BY  E.dno
```



```
SELECT AVG(E.sal)
FROM  Emp E
WHERE  E.age=25 AND
 E.sal BETWEEN 3000 AND 5000
```

<E.dno>

<E.dno,E.sal>
Tree Index

<E. age,E.sal>
or
<E.sal, E.age>
Tree Index