

Choice of Indexes



- ❖ What indexes should we create?
 - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- ❖ For each index, what kind of index should it be?
 - Clustered? Hash/tree?

Choice of Indexes (Contd.)



- ❖ **One approach:** Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
 - Obviously, this implies that we must understand how a DBMS evaluates queries and creates **query evaluation plans!**
 - For now, we discuss simple 1-table queries.
- ❖ Before creating an index, must also consider the impact on updates in the workload!
 - **Trade-off:** Indexes can make queries go faster, updates slower. Require disk space, too.

Index Selection Guidelines



- ❖ Attributes in WHERE clause are candidates for index keys.
 - Exact match condition suggests hash index.
 - Range query suggests tree index.
 - Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.
- ❖ Multi-attribute search keys should be considered when a WHERE clause contains *several conditions*.
 - Order of attributes is important for range queries.
 - Such indexes can sometimes enable **index-only** strategies for important queries.
 - For index-only strategies, clustering is not important!
- ❖ Try to choose indexes that benefit as many queries as possible. 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



- ❖ Clustered B+ tree index on *E.age* can be used to get qualifying tuples.

- How selective is the condition?

- ❖ Consider the GROUP BY query.

- If many tuples have *E.age* > 10, using *E.age* index and sorting the retrieved tuples may be costly.
- Clustered *E.dno* index may be better!

- ❖ Equality queries and duplicates:

- Clustering on *E.hobby* helps!

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

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

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

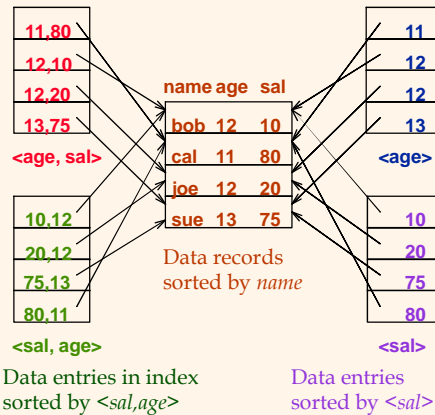
Indexes with Composite Search Keys

❖ **Composite Search Keys:** Search on a combination of fields.

- **Equality query:** Every field value is equal to a constant value. E.g. wrt $\langle \text{sal}, \text{age} \rangle$ index:
 - age=20 and sal =75
- **Range query:** Some field value is not a constant. E.g.:
 - age =20; or age=20 and sal > 10

❖ Data entries in index sorted by search key to support range queries.

Examples of composite key indexes using lexicographic order.



Composite Search Keys

- ❖ To retrieve Emp records with $\text{age}=30$ AND $\text{sal}=4000$, an index on $\langle \text{age}, \text{sal} \rangle$ would be better than an index on age or an index on sal .
 - Choice of index key orthogonal to clustering.
- ❖ If condition is: $20 < \text{age} < 30$ AND $3000 < \text{sal} < 5000$:
 - Clustered tree index on $\langle \text{age}, \text{sal} \rangle$ or $\langle \text{sal}, \text{age} \rangle$ is best.
- ❖ If condition is: $\text{age}=30$ AND $3000 < \text{sal} < 5000$:
 - Clustered $\langle \text{age}, \text{sal} \rangle$ index much better than $\langle \text{sal}, \text{age} \rangle$ index!
- ❖ Composite indexes are larger, updated more often.

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.

$\langle E.dno \rangle$

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

$\langle E.dno, E.sal \rangle$
Tree index!

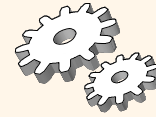
```
SELECT E.dno, MIN(E.sal)
FROM Emp E
GROUP BY E.dno
```

$\langle E.age, E.sal \rangle$
or

$\langle E.sal, E.age \rangle$
Tree index!

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

Index-Only Plans (Contd.)



- ❖ Index-only plans are possible if the key is $\langle dno, age \rangle$ or we have a tree index with key $\langle age, dno \rangle$
 - Which is better?
 - What if we consider the second query?

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age=30
GROUP BY E.dno
```

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>30
GROUP BY E.dno
```

Index-Only Plans (Contd.)



- ❖ Index-only plans can also be found for queries involving more than one table; more on this later (Ch 10).

<E.dno>

```
SELECT D.mgr
FROM Dept D, Emp E
WHERE D.dno=E.dno
```

<E.dno,E.eid>

```
SELECT D.mgr, E.eid
FROM Dept D, Emp E
WHERE D.dno=E.dno
```

Summary



- ❖ Many alternative file organizations exist, each appropriate in some situation.
- ❖ If selection queries are frequent, sorting the file or building an *index* is important.
 - Hash-based indexes only good for equality search.
 - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- ❖ Index is a collection of data entries plus a way to quickly find entries with given key values.

Summary (Contd.)



- ❖ Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
 - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- ❖ Can have several indexes on a given file of data records, each with a different search key.
- ❖ Indexes can be classified as clustered vs. unclustered, primary vs. secondary, etc. Differences have important consequences for utility/performance.

Summary (Contd.)



- ❖ Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
 - What are the important queries and updates? What attributes/relations are involved?
- ❖ Indexes must be chosen to speed up important queries (and perhaps some updates!).
 - Index maintenance overhead on updates to key fields.
 - Choose indexes that can help many queries, if possible.
 - Build indexes to support index-only strategies.
 - Clustering is an important decision; only one index on a given relation can be clustered!
 - Order of fields in composite index key can be important.