Database Tuning

Measure of Performance

- **Response time** – average time to wait for a response to a particular query

- **Throughput** – volume of work completed in a fixed amount of time (often measured as transactions per second)

Normalization

- Can improve performance
  - Less redundancy

    => more rows/page => less I/O
  - More tables

    => smaller and more clustered indexes

Normalization (cont.)

- Can decrease performance
- E.g.:
  - Transcript(*StudId, CrsCode, Semester, Grade*)
  - Students(*StuId*, *name*)
  - The relations are in BCNF, *but* ...
  - a join required to list names of students with A in 291

```
SELECT S.Name
FROM Student S, Transcript T
WHERE S.Id = T.StudId AND T.CrsCode = '291'
                      AND T.Grade = 'A'
```

  - … and join is expensive

Denormalize

- Add attribute Name to Transcript

```
SELECT T.Name
FROM Transcript T
WHERE T.CrsCode = 'CS305' AND T.Grade = 'A'
```

- Join is avoided, but added redundancy.

Performance Bottlenecks

- A table can be a performance bottleneck if
  - it is heavily used, causing locking contention
  - it's index is deep (has many rows), increasing I/O
  - rows are wide, increasing I/O
- Solution: table partitioning

Horizontal Partitioning

- Split rows
  - Geographically (e.g., by state), organizationally (e.g., by department), active/inactive (e.g., current students vs. grads)
- Useful when accesses are confined to disjoint subsets of rows
- Advantages:
  - Spreads users out and reduces contention (particularly if tables can be spread over different devices)
  - Indexes have fewer levels (but only marginally)
  - Rows in a typical result set are concentrated in fewer pages
- Disadvantages:
  - Added complexity
  - Difficult to handle queries over all tables

Vertical Partitioning

- Split columns and replicate the key
- Useful when table has many columns and
  - it is possible to distinguish between frequently and infrequently accessed columns
  - different queries use different subsets of columns
- **E.g.**: Split e*mployee* table into
  - Columns related to *compensation* (sin, tax, benefits, salary) and columns related to *job* (sin, department, projects, skills).
  - Decomposition is lossless join (since *sin* is included in both tables), although a join is required.

Horizontal Partitioning in Oracle

- Methods: range, hash, list, etc.
- e.g.

```
Create table transcripts (
        StudId          char(6),
        CrsCode         char(8),
        Semester        char(3),
        Grade           int,
        Primary Key (StudId, CrsCode, Semester))
Partition BY LIST (Semester) (
        Partition p10 values ('F10', 'W10'),
        Partition p11 values ('F11', 'W11'),
        Partition rest values (default));
```

Indexes

- Advantages: fast table access (for search, insert, delete, join, constraint checking)
- Created
  - automatically on primary key
  - Manually on other columns

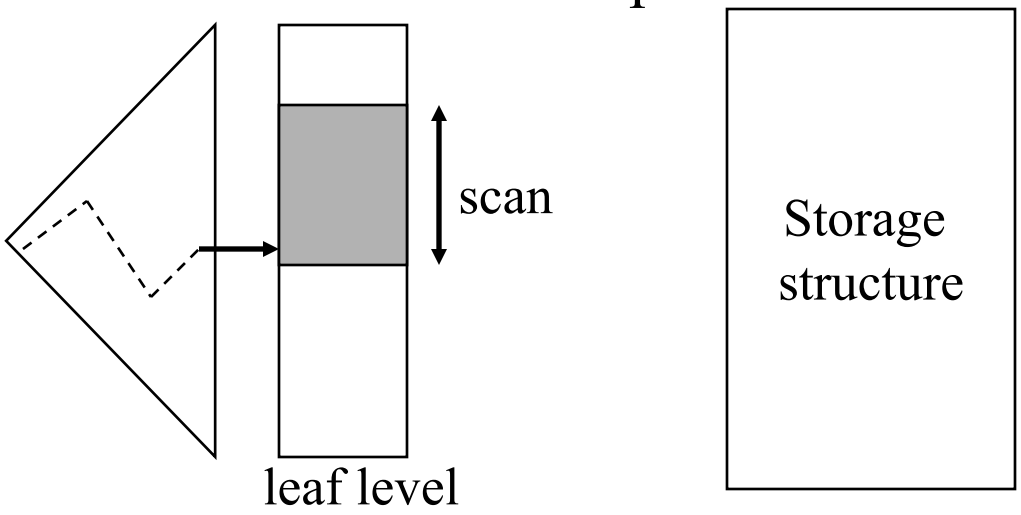
Index Covering

- An index covers a query if
  - the query can be computed from the index alone
- All attributes of the query must be included in a covering index

Index Covering

- **E.g.** Professor(DeptId, Name, …)
  - Dense B+-tree index on (DeptId, Name) covers the query because …

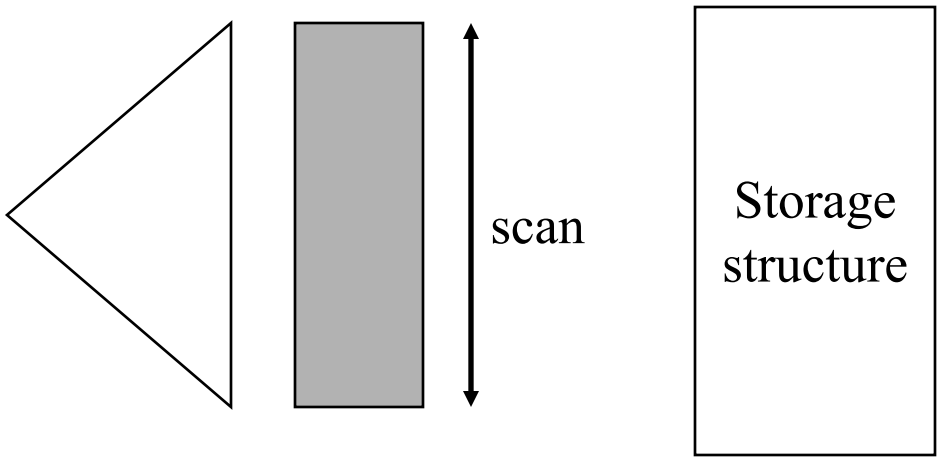
```
SELECT P.Name
FROM Professor P
WHERE P.DeptId = 'CS'
```



Index Covering

- **E.g.** Student(Id, Name, Address, …)
  - Dense B+-tree index on (Id, Name, Address)

```
SELECT S. Id, S.Name
FROM Student S
WHERE S.Address = '1 Lake St'
```



Choosing a Clustered Index

- Choose clustered index to support:
  - Point queries with large result sets
    - A clustered index on Transcript with search key (CrsCode, Semester) supports queries requesting the Id's of all students in a particular class.
  - Range queries
    - A clustered index on Transcript with search key (Grade) supports queries requesting the Id's of all students with grades between B+ and A
  - ORDER BY clauses
    - Sequence of attributes in ORDER BY is a prefix of search key
  - Index-nested (get all rows satisfying a particular search key value) and sort-merge (avoid sort) joins.

Choosing a Clustered Index

- Do not choose a clustered index
  - If it is not needed for one of the above
  - If the search key attribute is frequently updated
    - Since it will be necessary to move rows frequently
      - Hash – from bucket to bucket
      - B^+ tree – from one leaf page to another

Choosing an Unclustered Index

- Choose unclustered index to support
  - Queries with small result sets
  - Index-nested loop joins (when the indexed column is in the inner relation)
  - Index covering
  - Point queries with small result sets

Miscellaneous Hints

- If an attribute is unique, declare it so (query optimizer can use it in query planning)
  - Only one row can match a search or join attribute
- Adjust fill factor
  - To 100% if table is read-only
  - To a smaller value if table is dynamic
- Keep search keys small to flatten index
- Add unclustered indexes only when necessary

Tuning SQL

- Hints:
  - Avoid sorts
    - Use of DISTINCT, UNION, EXCEPT, ORDER BY, GROUP BY cause sorts. Avoid their use if possible.

Tuning SQL

- Hints (con't):
  - Beware of views since they may cause unnecessary joins
  - Consider restructuring a query – different formulations will have different costs depending on state of tables and indexes available.

Influencing the Query Optimizer

- Statistics: Used by opimizers to estimate cost of a query plan (based on size of result sets)
  - Table: number of rows, number of distinct values of an attribute, max and min attribute values
  - Index: depth, number of leaf pages, number of distinct search key values
  - Histograms of attribute values
    - Example: use unclustered index if histogram shows that number of rows with attribute value specified in query is small, else use scan
  - Must be periodically updated if table dynamic

More Levels of Tuning

- Application level:
  - Query: schema redesign, use of indexes, client vs. server processing (stored procedures)
  - Transaction: isolation level, code design
- System level:
  - Cache issues: cache size, binding, I/O size
  - Distribution of data across devices
  - Log management
- Hardware level:
  - Number of cpu's
  - Disk configuration (many small disks vs. a few large ones)
  - Backup (mirrored disks vs logs)
- Distribution:
  - Replication
  - Distributing data and processing