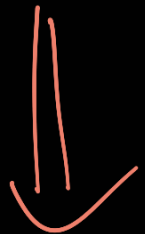
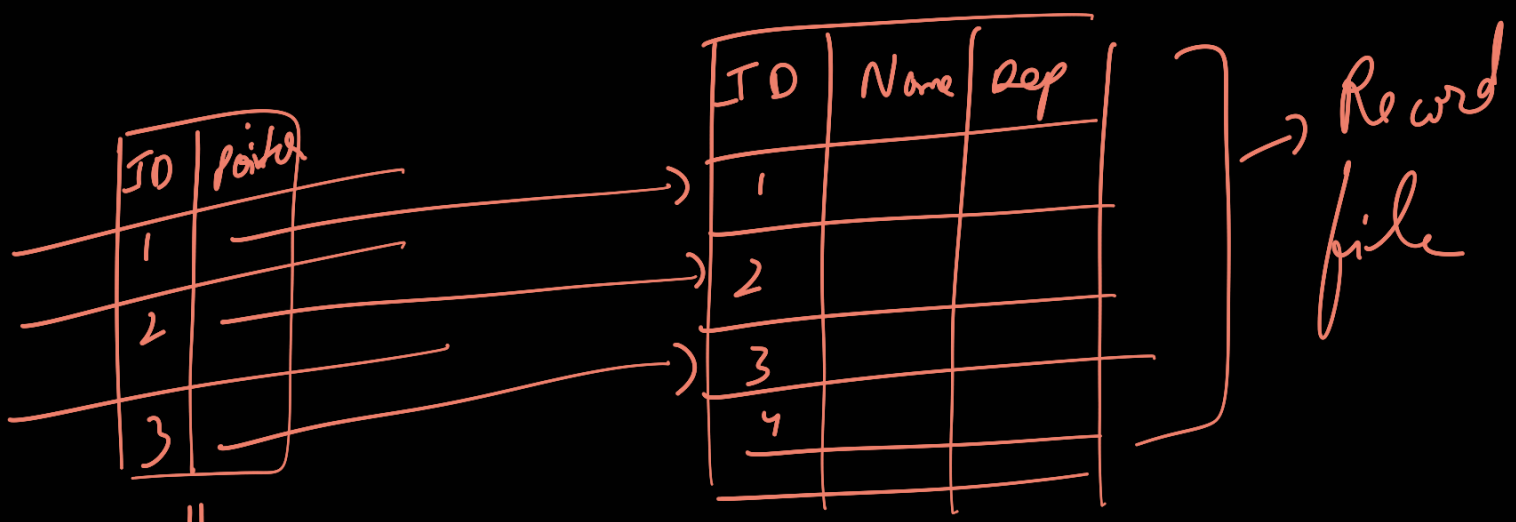
Indexing

- Need (why)
- Implementation (How)
- How file system indexing is different from DBMS indexing

Many queries reference only a small proportion of the records in a file. For example, a query like "Find all instructors in the Physics department" or "Find the total number of credits earned by the student with *ID* 22201" references only a fraction of the student records. It is inefficient for the system to read every tuple in the *instructor* relation to check if the *dept_name* value is "Physics". Likewise, it is inefficient to read the entire *student* relation just to find the one tuple for the *ID* "32556,". Ideally, the system should be able to locate these records directly. To allow these forms of access, we design additional structures that we associate with files.

* Index is also a file but it doesn't contain actual data but pointer to data.





Index file having ID as index

* Both actual file of records and index file associated with it are present in HDD.
* When DBMS process is created, it loads index file into memory for faster look-ups.
* Important part is what data structure can be used to

make search on this index file faster

↳ Tree → log(n) complexity

* Size of index file <<< Size of actual records

||

This will take less HDD blocks than this

* So, accessing time/searching time is significantly decreased.

⟹ | Key | HDD blocks pointer |

↳ Each row in index file

* One table could have multiple indexes/index file depending on type of SQL query

1. **Purpose of Indexes**:
   - Speed up *SELECT* queries and *WHERE* clauses.
   - Improve the performance of sorting and grouping operations, as well as JOINs.
2. **How Indexes Work**:
   - Indexes are built on columns of a table. When a query searches for values in these columns, the database engine uses the index instead of scanning row by row.
   - Indexes often use structures like B-Trees or Hashes. B-Tree indexes are most common because they allow ordered traversal, which is useful for range-based queries.

4. **Index Costs**:
   - **Storage Overhead**: Indexes consume additional disk space.
   - **Slower Writes**: Inserts, updates, and deletes can slow down since indexes also need to be updated.
   - **Maintenance**: Regular updates or high data modification rates can require index maintenance.
5. **Index Best Practices**:
   - Index columns frequently used in WHERE clauses, JOINs, and ORDER BY clauses.
   - Avoid over-indexing; too many indexes can slow down write operations and increase storage costs.
   - Consider composite indexes for columns often queried together.
   - Periodically monitor and adjust indexes based on query performance analysis.