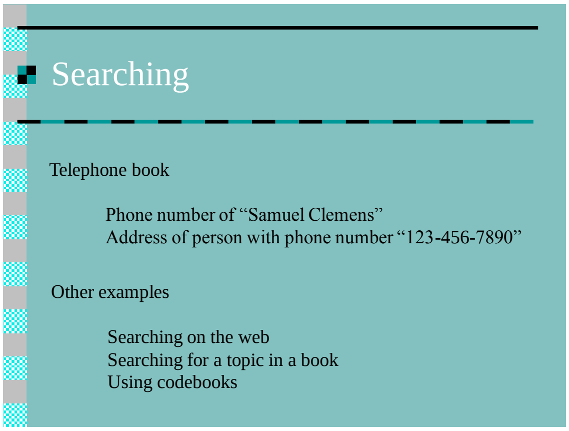
Indexes and Indexing



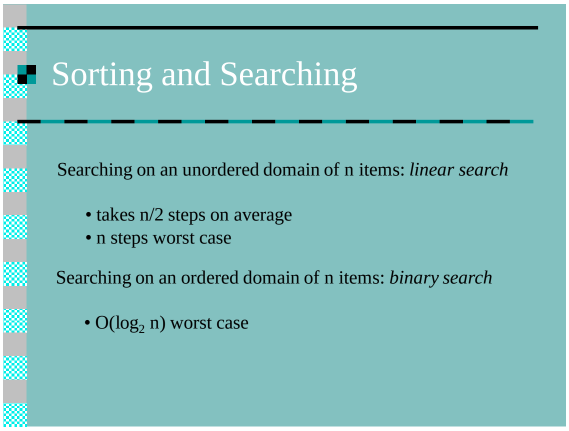
Searching

Telephone book

Phone number of “Samuel Clemens”
Address of person with phone number “123-456-7890”

Other examples

Searching on the web
Searching for a topic in a book
Using codebooks



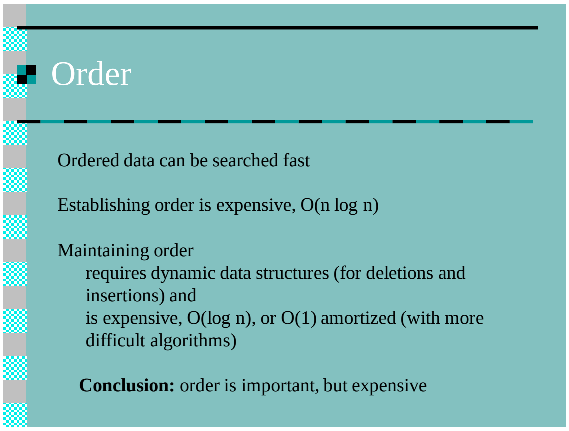
Sorting and Searching

Searching on an unordered domain of n items: *linear search*

- takes n/2 steps on average
- n steps worst case

Searching on an ordered domain of n items: *binary search*

- O($\log_2$ n) worst case



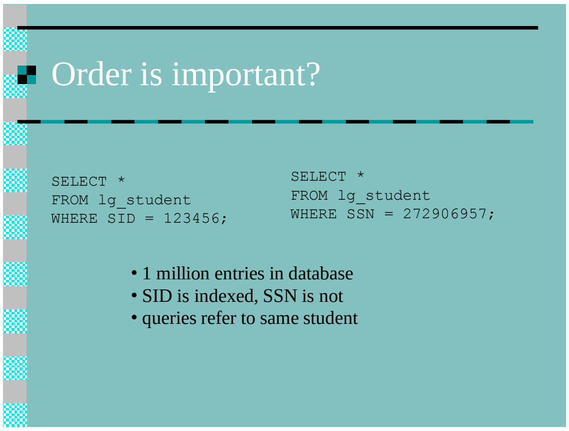
Order

Ordered data can be searched fast

Establishing order is expensive, O(n log n)

Maintaining order
requires dynamic data structures (for deletions and insertions) and
is expensive, O(log n), or O(1) amortized (with more difficult algorithms)

Conclusion: order is important, but expensive

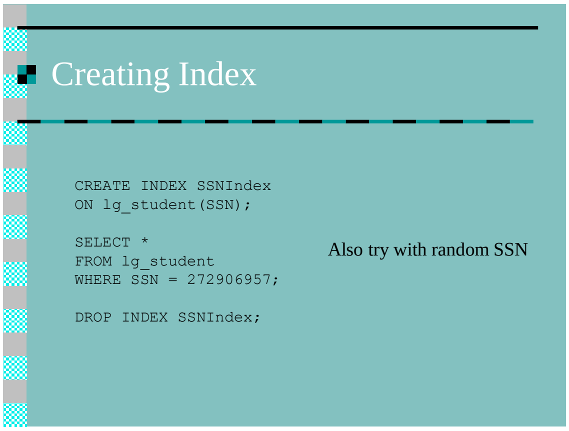


Order is important?

```
SELECT *
FROM lg_student
WHERE SID = 123456;
```

```
SELECT *
FROM lg_student
WHERE SSN = 272906957;
```

- 1 million entries in database
- SID is indexed, SSN is not
- queries refer to same student



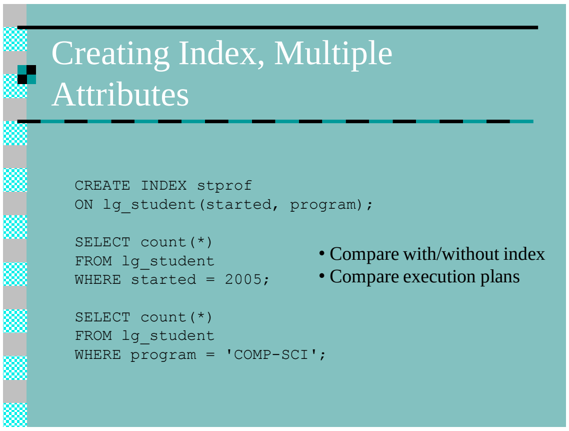
Creating Index

```
CREATE INDEX SSNIndex
ON lg_student(SSN);

SELECT *
FROM lg_student
WHERE SSN = 272906957;

DROP INDEX SSNIndex;
```

Also try with random SSN



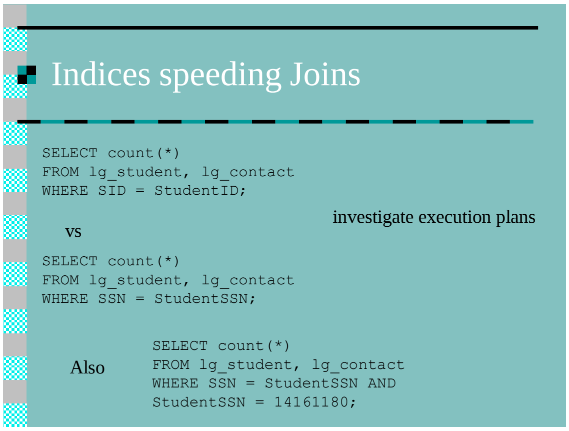
Creating Index, Multiple Attributes

```
CREATE INDEX stprof
ON lg_student(started, program);

SELECT count(*)
FROM lg_student
WHERE started = 2005;

SELECT count(*)
FROM lg_student
WHERE program = 'COMP-SCI';
```

- Compare with/without index
- Compare execution plans



Indices speeding Joins

```
SELECT count(*)
FROM lg_student, lg_contact
WHERE SID = StudentID;
```

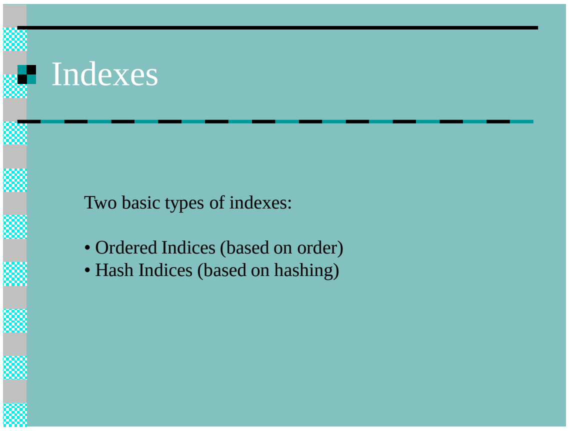
investigate execution plans

vs

```
SELECT count(*)
FROM lg_student, lg_contact
WHERE SSN = StudentSSN;
```

Also

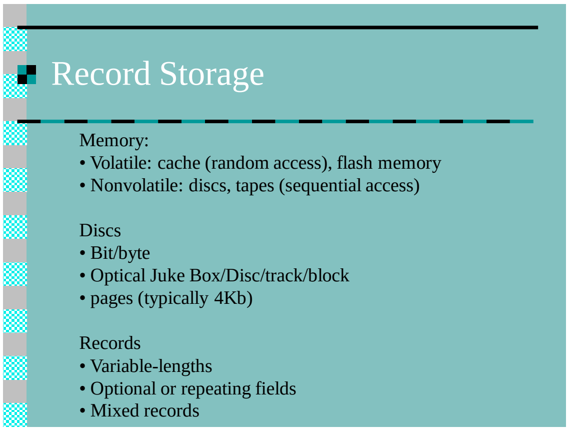
```
SELECT count(*)
FROM lg_student, lg_contact
WHERE SSN = StudentSSN AND
StudentSSN = 14161180;
```



Indexes

Two basic types of indexes:

- Ordered Indices (based on order)
- Hash Indices (based on hashing)

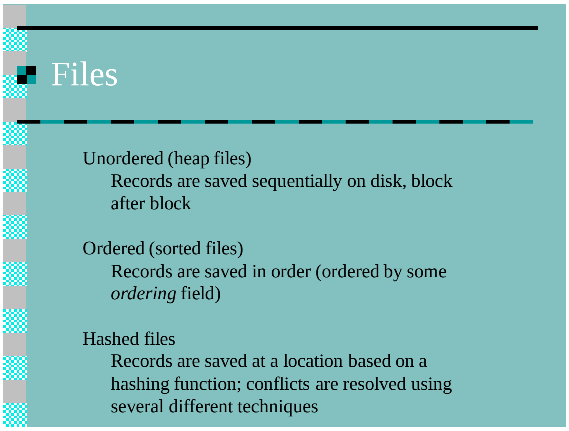


Record Storage

Memory:
- Volatile: cache (random access), flash memory
- Nonvolatile: discs, tapes (sequential access)

Discs
- Bit/byte
- Optical Juke Box/Disc/track/block
- pages (typically 4Kb)

Records
- Variable-lengths
- Optional or repeating fields
- Mixed records

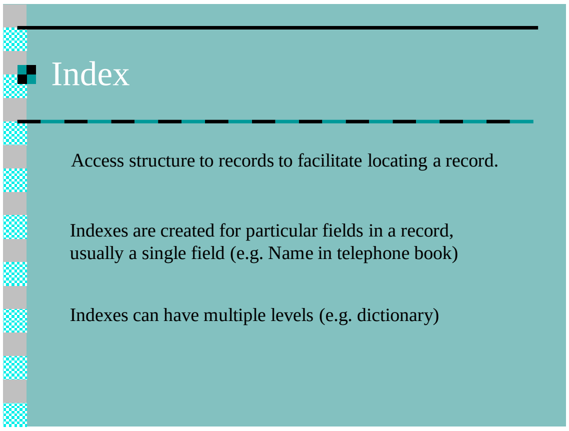


Files

Unordered (heap files)
Records are saved sequentially on disk, block after block

Ordered (sorted files)
Records are saved in order (ordered by some *ordering* field)

Hashed files
Records are saved at a location based on a hashing function; conflicts are resolved using several different techniques

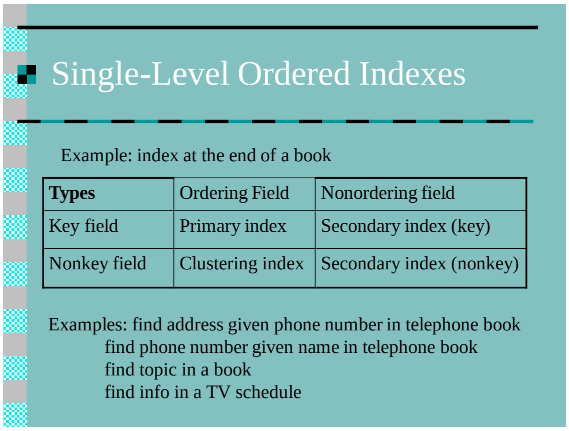


Index

Access structure to records to facilitate locating a record.

Indexes are created for particular fields in a record, usually a single field (e.g. Name in telephone book)

Indexes can have multiple levels (e.g. dictionary)

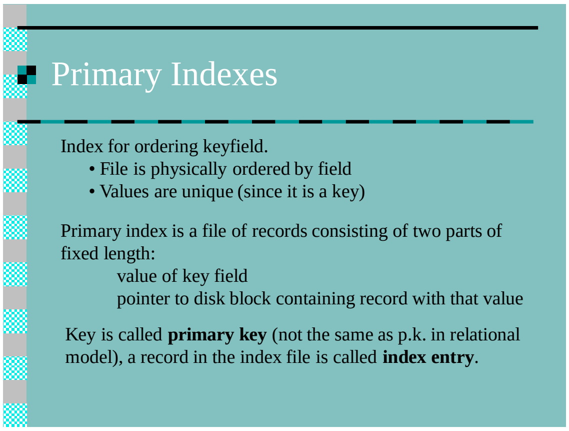


Single-Level Ordered Indexes

Example: index at the end of a book

Types	Ordering Field	Nonordering field
Key field	Primary index	Secondary index (key)
Nonkey field	Clustering index	Secondary index (nonkey)

Examples: find address given phone number in telephone book
find phone number given name in telephone book
find topic in a book
find info in a TV schedule



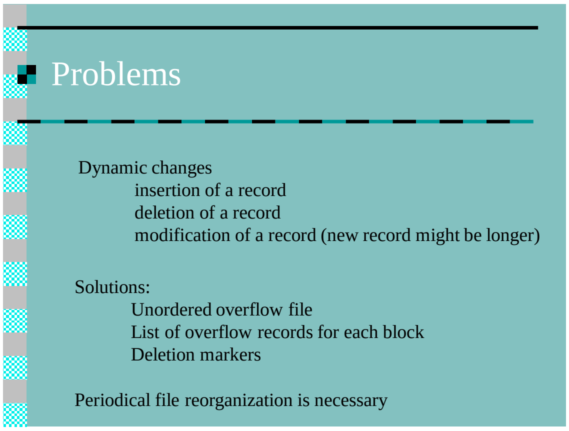
Primary Indexes

Index for ordering keyfield.

- File is physically ordered by field
- Values are unique (since it is a key)

Primary index is a file of records consisting of two parts of fixed length:
value of key field
pointer to disk block containing record with that value

Key is called **primary key** (not the same as p.k. in relational model), a record in the index file is called **index entry**.



Problems

Dynamic changes
insertion of a record
deletion of a record
modification of a record (new record might be longer)

Solutions:
Unordered overflow file
List of overflow records for each block
Deletion markers

Periodical file reorganization is necessary

Clustering Index

Index for ordering field which is not a key

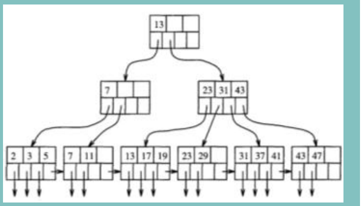
- File is physically ordered by field
- Values are not unique
- Only distinct values are indexed

Same issues as with primary index

Primary/Clustering Implementation?

Need dynamic data structures for maintaining indexes based on search trees:

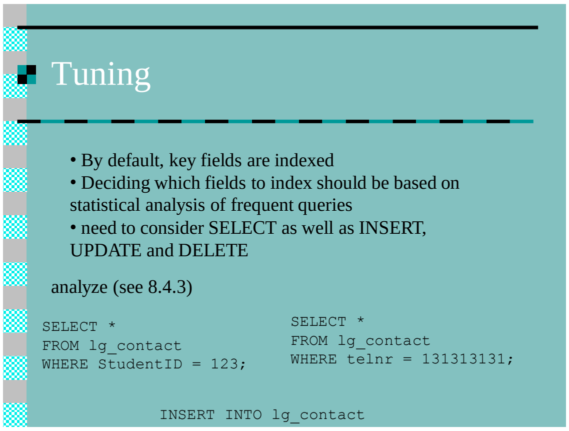
- B-Tree
- B^+-Tree



Hash Indexes

A hash function maps a large set (the set of potential records) to a small set (the storage locations) without causing too many conflicts.

Use hash function to find a location to store the index information of a record.



Tuning

- By default, key fields are indexed
- Deciding which fields to index should be based on statistical analysis of frequent queries
- need to consider SELECT as well as INSERT, UPDATE and DELETE

analyze (see 8.4.3)

```
SELECT *
FROM lg_contact
WHERE StudentID = 123;
```

```
SELECT *
FROM lg_contact
WHERE telnr = 131313131;
```

```
INSERT INTO lg_contact
```