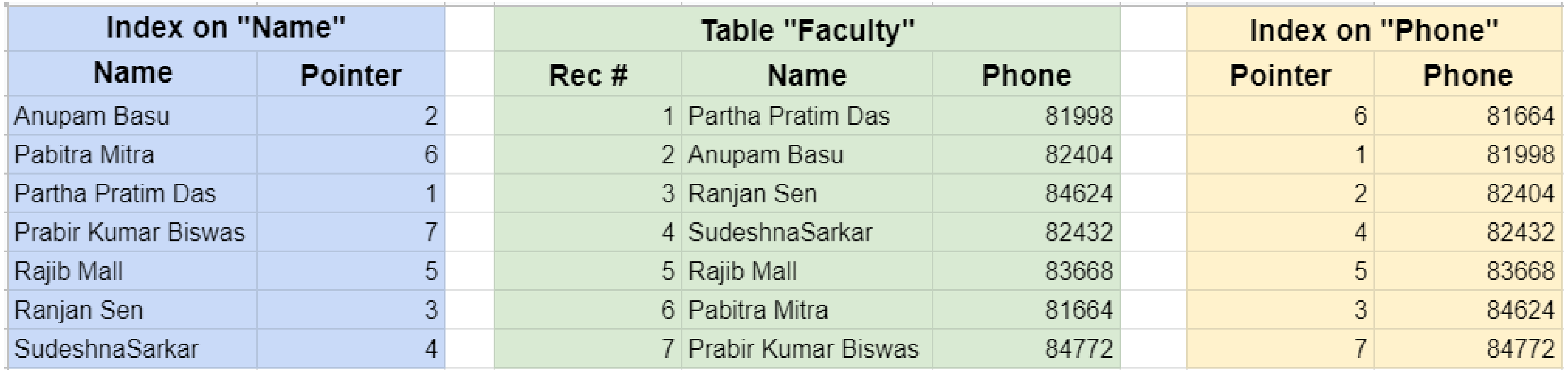
Basic Concepts of Indexing

Search Records

- Consider a table: **Faculty**(*Name, Phone*)



Index on "Name"		Table "Faculty"			Index on "Phone"	
Name	**Pointer**	**Rec #**	**Name**	**Phone**	**Pointer**	**Phone**
Anupam Basu	2	1	Partha Pratim Das	81998	6	81664
Pabitra Mitra	6	2	Anupam Basu	82404	1	81998
Partha Pratim Das	1	3	Ranjan Sen	84624	2	82404
Prabir Kumar Biswas	7	4	SudeshnaSarkar	82432	4	82432
Rajib Mall	5	5	Rajib Mall	83668	5	83668
Ranjan Sen	3	6	Pabitra Mitra	81664	3	84624
SudeshnaSarkar	4	7	Prabir Kumar Biswas	84772	7	84772

- How to search on Name?
  - Get the phone number for ‘Pabitra Mitra’
  - Use “Name” Index – sorted on ‘Name’, search ‘Pabitra Mitra’ and navigate on pointer (rec #)

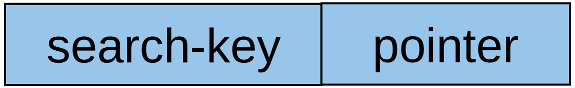
- How to search on Phone?
  - Get the name of the faculty having phone number = 84772
  - Use “Phone” Index – sorted on ‘Phone’, search ‘84772’ and navigate on pointer (rec #)

- We can keep the records sorted on ‘Name’ or on ‘Phone’ (called the primary index), but not on both

Basic Concepts

- Indexing mechanisms used to speed up access to desired data
- For example:
  - Name in a faculty table
  - Author catalog in library

- **Search Key** attribute to set of attributes used to look up records in a file
- An **index file** consists of records (called **index entries**) of the form



- Index files are typically much smaller than the original file

- Two basic kinds of indices:
- **Ordered indices:** Search keys are stored in sorted order
- **Hash indices:** Search keys are distributed uniformly across “buckets” using a “hash function”

Index Evaluation Metrics

- Access types supported efficiently
- For example:
  - Records with a specified value in the attribute
  - Records with an attribute value falling in a specified range of values
- Access time
- Insertion time
- Deletion time
- Space overhead

Ordered Indices

- In an **ordered index,** index entries are stored sorted on the search key value
  - For example, author catalog in library

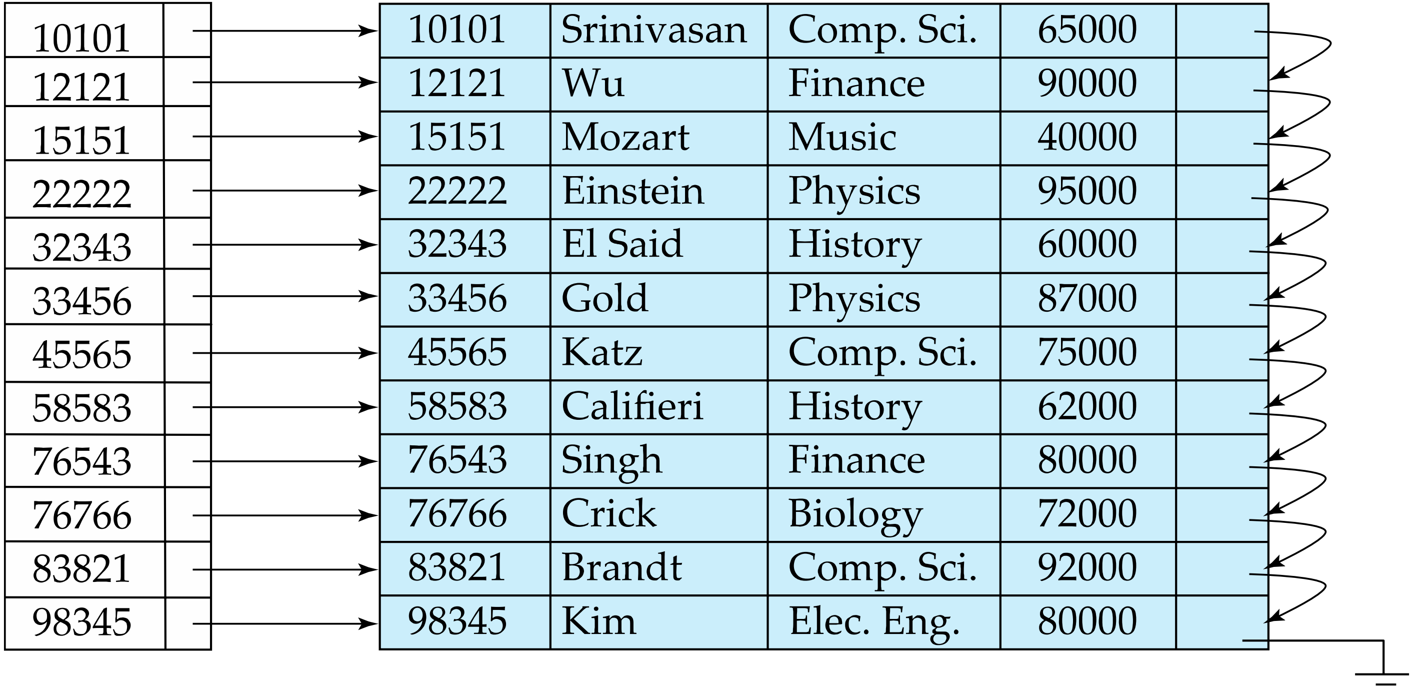
- **Primary index:** In a sequentially ordered file, the index whose search key specifies the sequential order of the file
  - Also called **clustering index**
  - The search key of a primary index is usually but not necessarily the primary key

- **Secondary index**: An index whose search key specifies an order different from the sequential order of the file
  - Also called **nonclustering index**

- **Index-sequential file:** Ordered sequential file with a primary index

Dense Index Files

- **Dense index:** Index record appears for every search-key value in the file
  - E.g. index on *ID* attribute of *instructor* relation

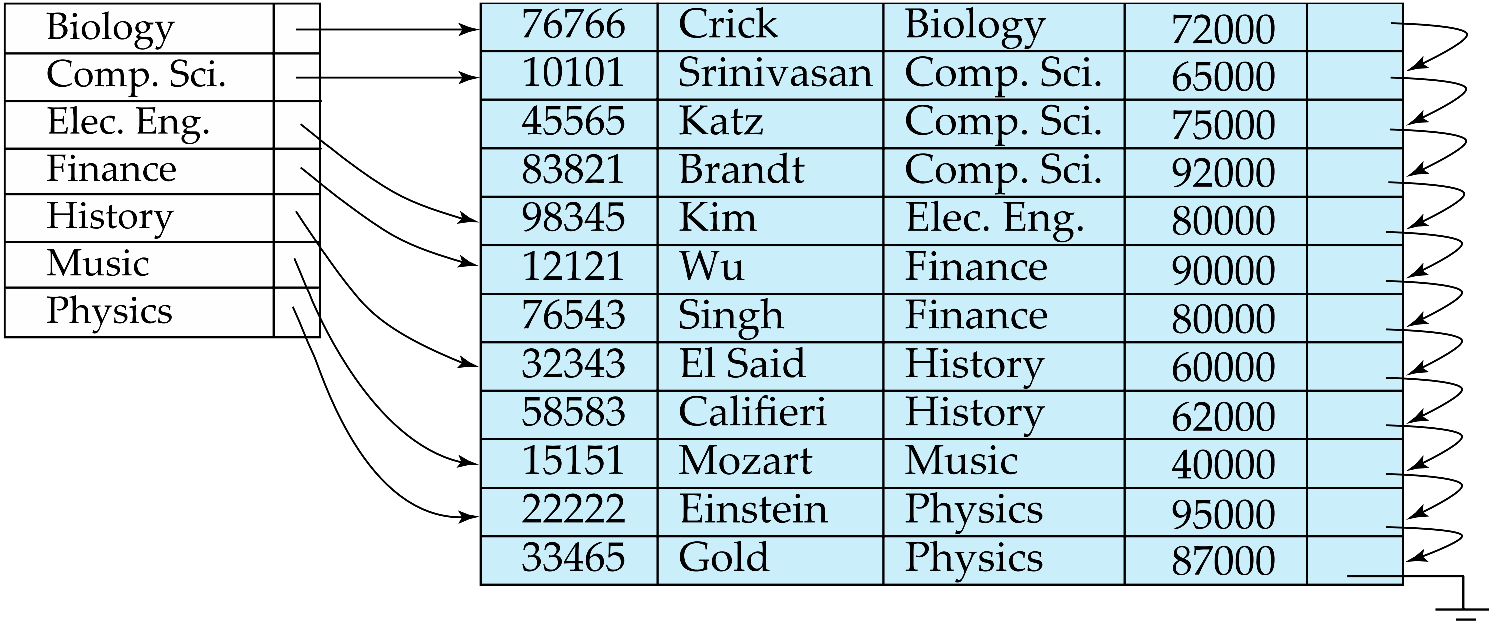


10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Dense Index Files

- Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

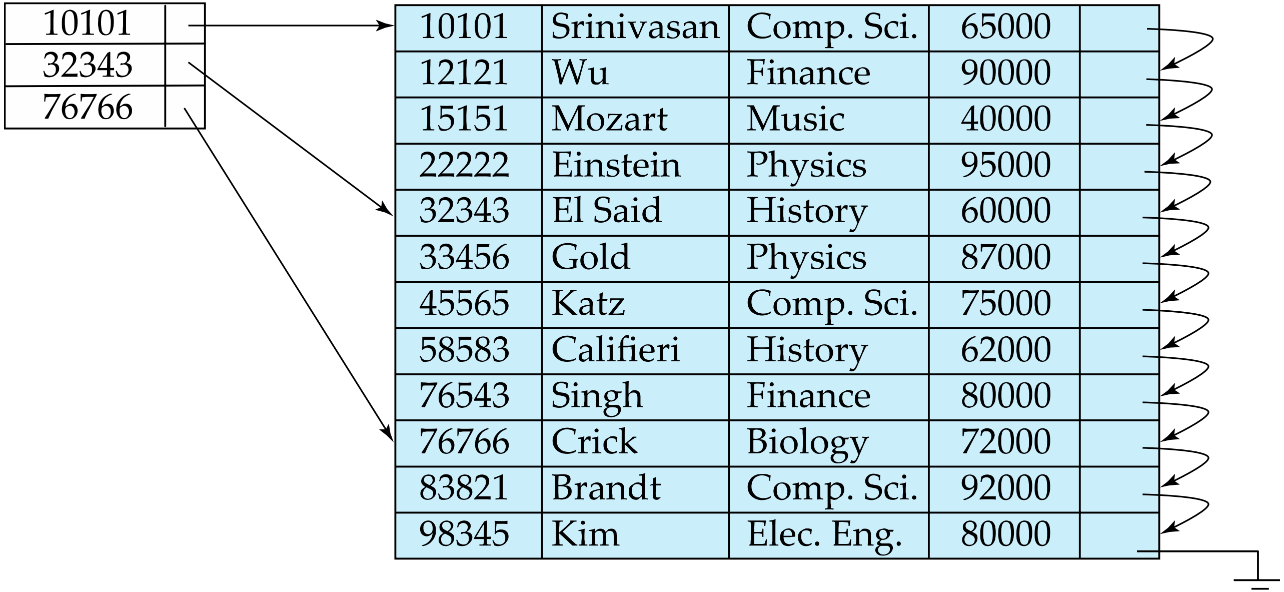


Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

Sparse Index Files

- **Sparse Index**: Contains index records for only some search-key values
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points

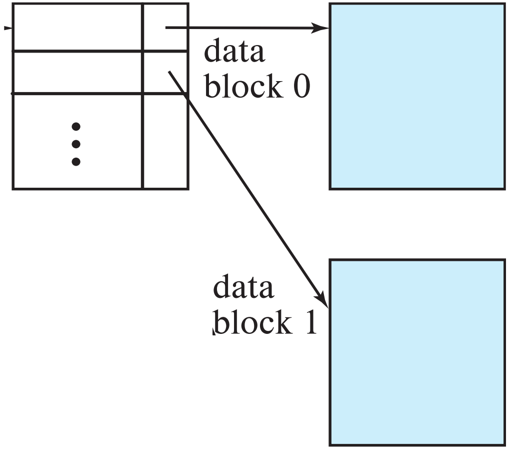


10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

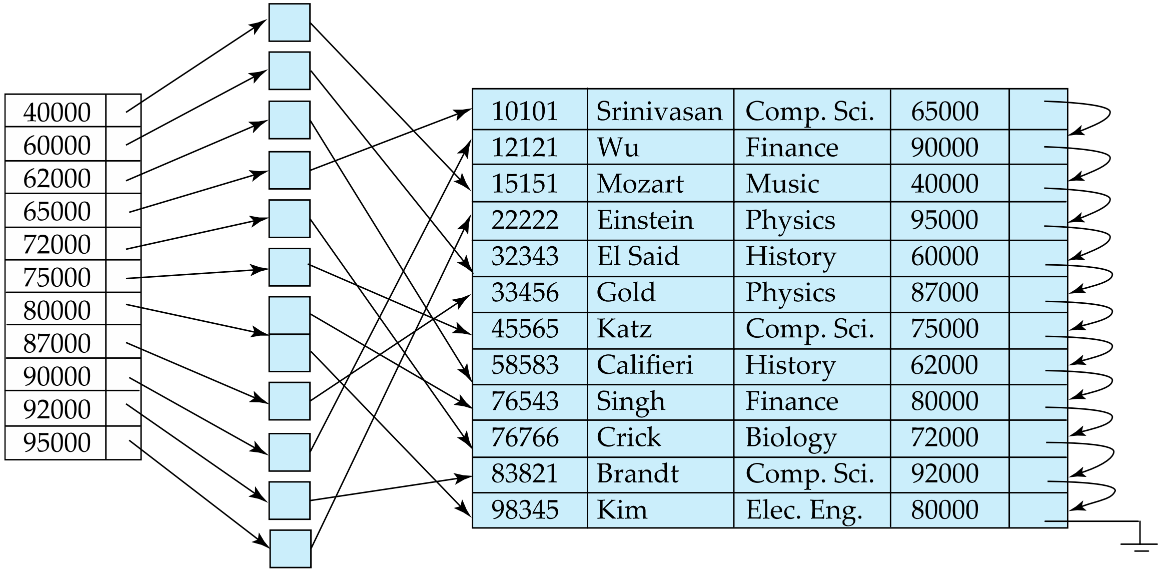
Sparse Index Files

- Compared to dense indices:
  - Less space and less maintenance overhead for insertions and deletions
  - Generally slower than dense index for locating records
- **Good tradeoff**: Sparse index with an index entry for every block in file, corresponding to least search-key value in the block
- For unclustered index: Sparse index on top of dense index (multilevel index)



Secondary Indices Example

- Secondary index on *salary* field of instructor



40000
60000
62000
65000
72000
75000
80000
87000
90000
92000
95000

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value

- Secondary indices have to be dense

Primary and Secondary Indices

- Indices offer substantial benefits when searching for records

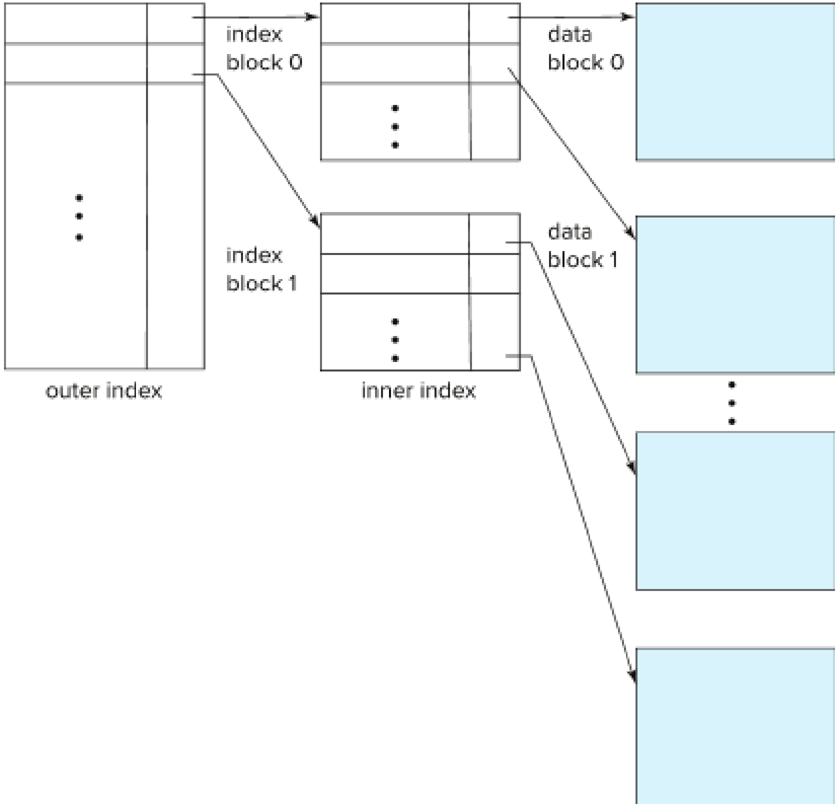
- BUT: Updating indices imposes overhead on database modification, when a file is modified, every index on the file must be updated

- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
  - Each record access may fetch a new block from disk
  - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access

Multilevel Index

- If primary index does not fit in memory, access becomes expensive
- Solution: Treat primary index kept on disk as a sequential file and construct a sparse index on it
  - **Outer index:** A sparse index of primary index
  - **Inner index:** The primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on
- Indices at all levels must be updated on insertion or deletion from the file

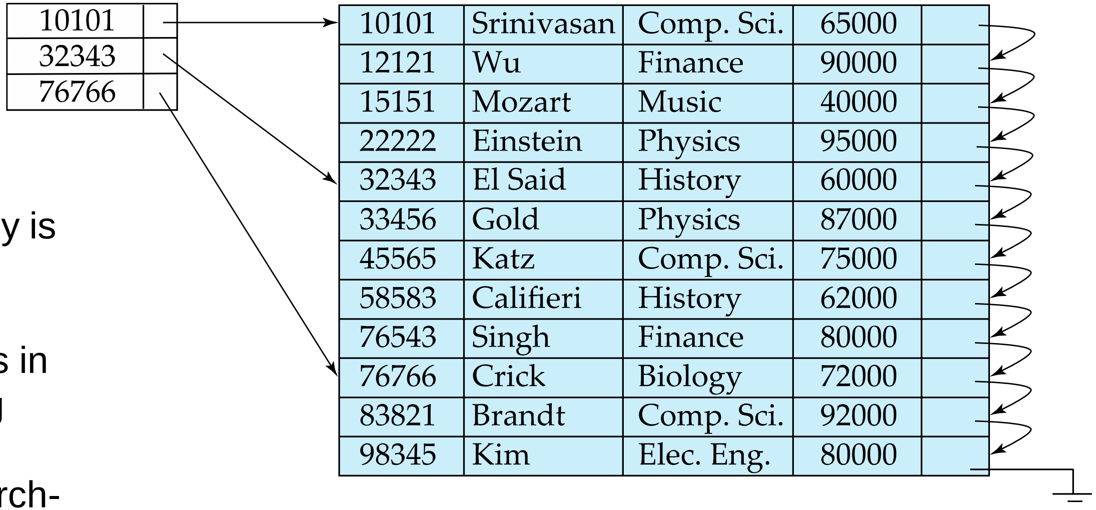
Multilevel Index



Index Update: Deletion

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also

- **Single-level index entry deletion:**
  - **Dense indices:** Deletion of search-key is similar to file record deletion
  - **Sparse indices**
    - If an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order)
    - If the next search-key value already has an index entry, the entry is deleted instead of being replaced



10101
32343
76766

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Index Update: Insertion

- **Single-level index insertion:**
  - Perform a lookup using the search-key value of the record to be inserted
  - **Dense indices:** If the search-key value does not appear in the index, insert it
    - Indices are maintained as sequential files
    - Need to create space for new entry, overflow blocks may be required
  - **Sparse indices:** If index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created
    - If a new block is created, the first search-key value appearing in the new block is inserted into the index

- **Multilevel insertion and deletion:** Algorithms are simple extensions of the single-level algorithms

Secondary Indices

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition
  - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
  - Example 2: As discussed before, but where we want to find all instructors with a specified salary or with salary in a specified range of values

- We can have a secondary index with an index record for each search-key value

Next Lecture

2-3-4 Trees

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in