CSEN 604: Databases II

Lecture 1

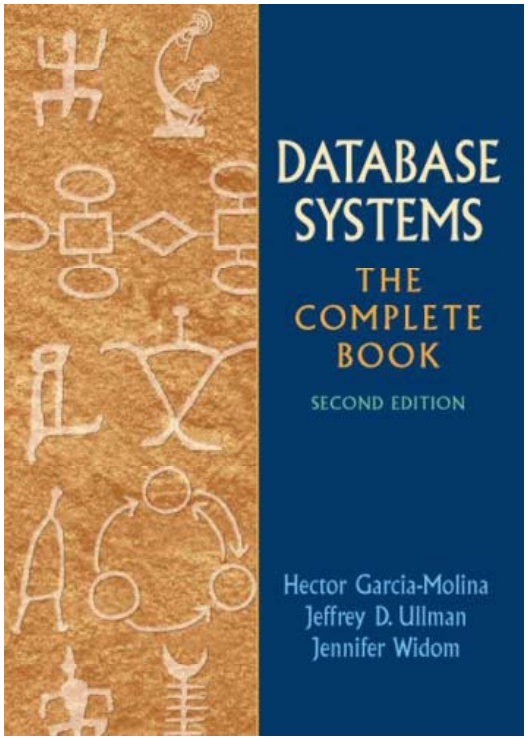
Dr. Wael Abouelsaadat
wael.abouelsaadat@guc.edu.eg
Office: C7.208

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the book: *Database Systems; the Complete Book*

CSEN604

- Architecture, Data Structures & Algorithms in an RDBMS
- Teaching Assistant:
  - Nader Alexan
  - Mohamed Karam Gabr
- Textbook:

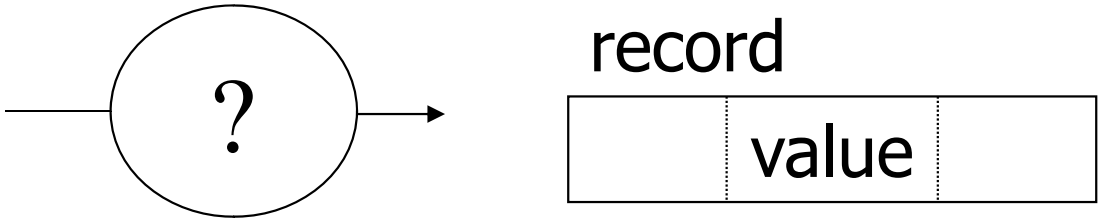
Database Systems: The Complete Book
(2nd Edition)
ISBN-13: 978-0131873254



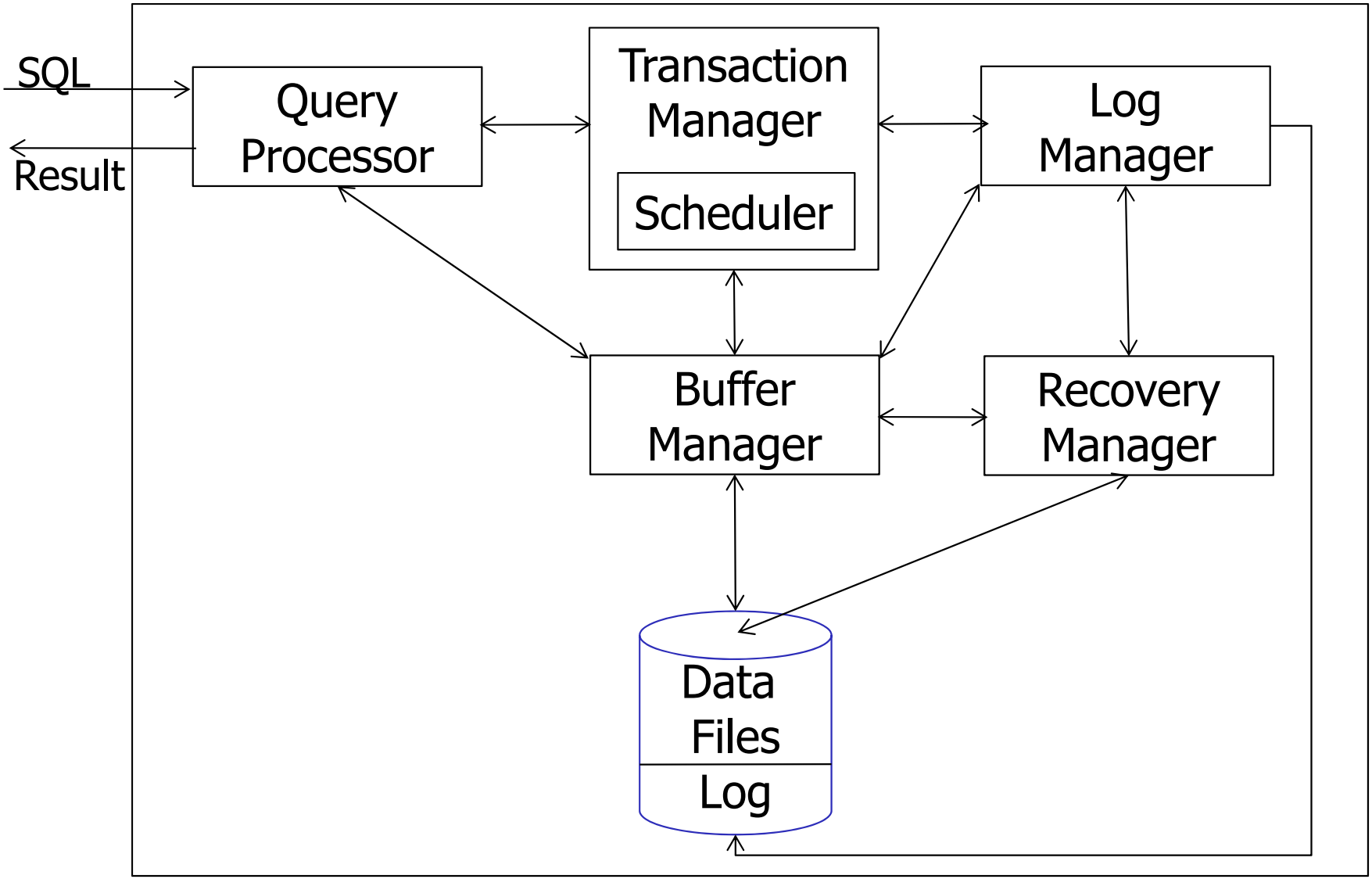
CSEN604 Marking

- 3 Assignments
- 2 Quizzes
- Midterm
- Exam

```
select *
from table
where
value=x;
```



DBMS Architecture



Topics

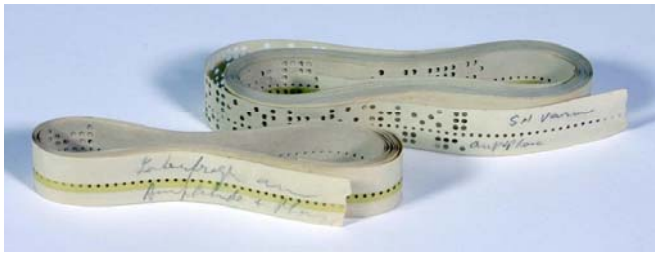
- File Organization
- Conventional Indices
- B-trees

File Organization

- Goals:
  - Allow insertion/deletions of tuples/records
  - Fetch a particular record (specified by record id)
  - Find all tuples that match a condition (say SSN = 123) ?
- Simplest case
  - A relation is mapped to a file
  - A file contains a sequence of records
  - Each record corresponds to a logical tuple
- More Sophisticated approach:
  - A relation is mapped to several files (blocks).
  - A block contains a sequence of records.
  - How are tuples/records stored within a block ?

Sequential File Organization

- Early databases
- Influenced by tape storage



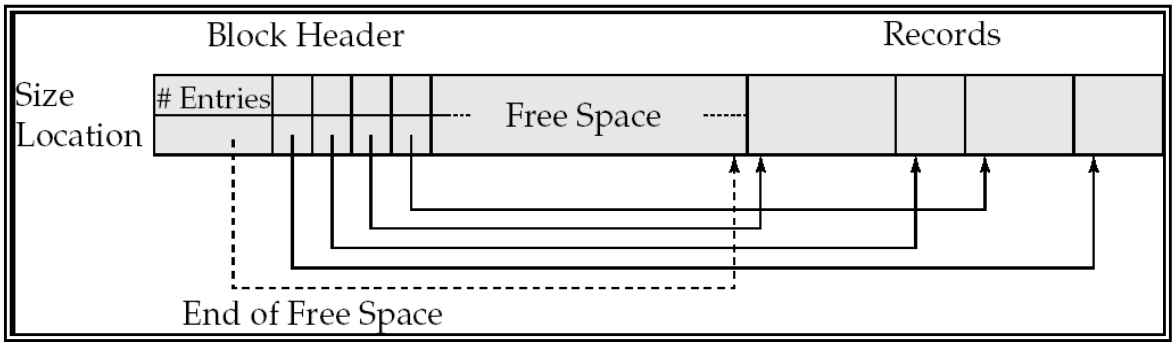
Fixed Length Records

- n = number of bytes per record
- Store record *i* at position:
  - n * (i – 1)
- Cons:
  - Wasted space
- Inserting a tuple ?
  - Simply append at the end of the record
- Deleting a tuple ?
  - Rearrange (defrag)

$n = l_1 + l_2 + l_3$

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Variable-length Records



- *How?*
  - Divide storage into units/pages of equal size (first slotted page structure)
  - Store a record inside a page
  - Headers are used as a indirection mechanism

    E.g. Record ID 1000 is in page number X starting from location Y
  - Pages grow from right to left
- *Advantage:*
  - The records may move inside the page, but the outside world is oblivious to it

Big Idea: Sorted Sequential FO

- Ok, we are going to store a file in several pages from now on.
- But, let us Keep the content sorted by some search key

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Page 1

350
400
500

Page 2

600
700
700

Page 3

700
750
900

Big Idea: Sorted Sequential FO

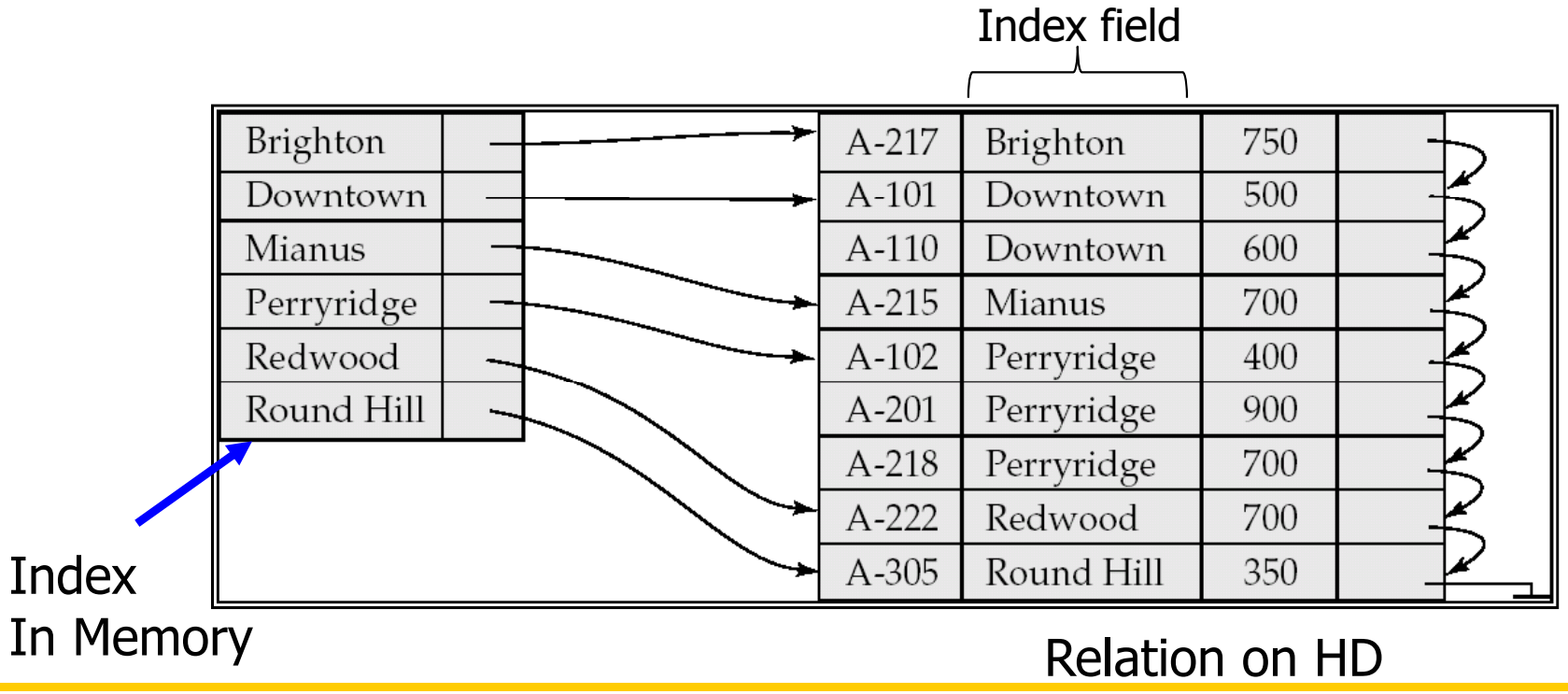
- Insertion ?
  - Find the page in which the tuple should be
  - If there is free space, insert it
  - Otherwise, must create overflow pages
- Deletions ?
  - Delete and keep the free space
  - Databases tend to be insert heavy, so free space gets used fast
- Can become *fragmented*
  - Must reorganize once in a while
- What if we want to find a particular record by value ?

Even Better: Indexed FO

- A data structure for efficient search through large databases
- Two key ideas:
  1) The pages are mapped to the disk blocks in a specific way
  2) Auxiliary data structures are maintained that allow quick search
- Think library index/catalogue
- Search key:
  - Attribute or set of attributes used to look up records

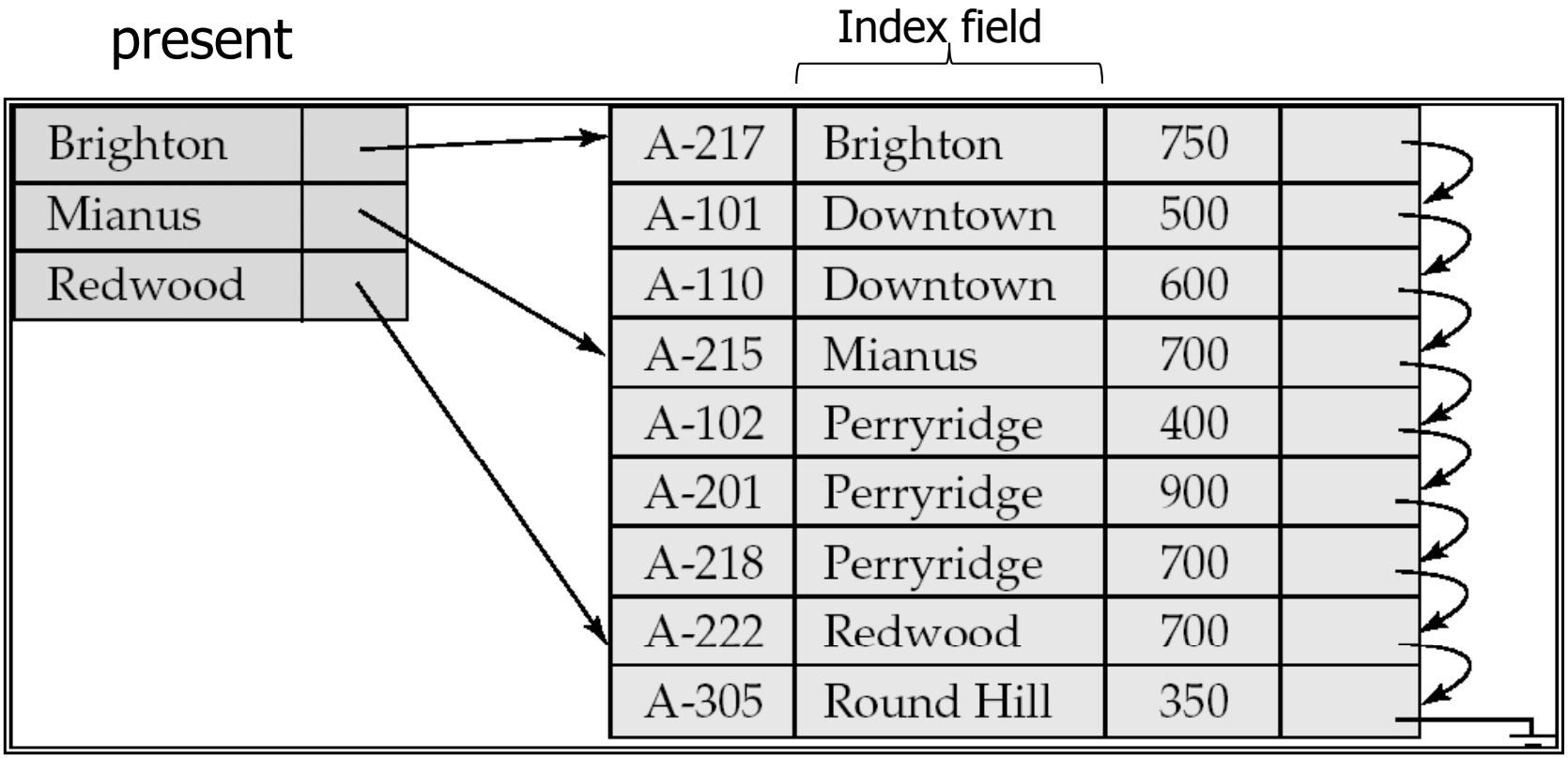
Primary Index

- Primary (dense) index
  - The relation is sorted on the key of the index
  - Can have only one primary index on a relation
- Note: index is smaller in size
- Additional link tells where is the next record on the HD.



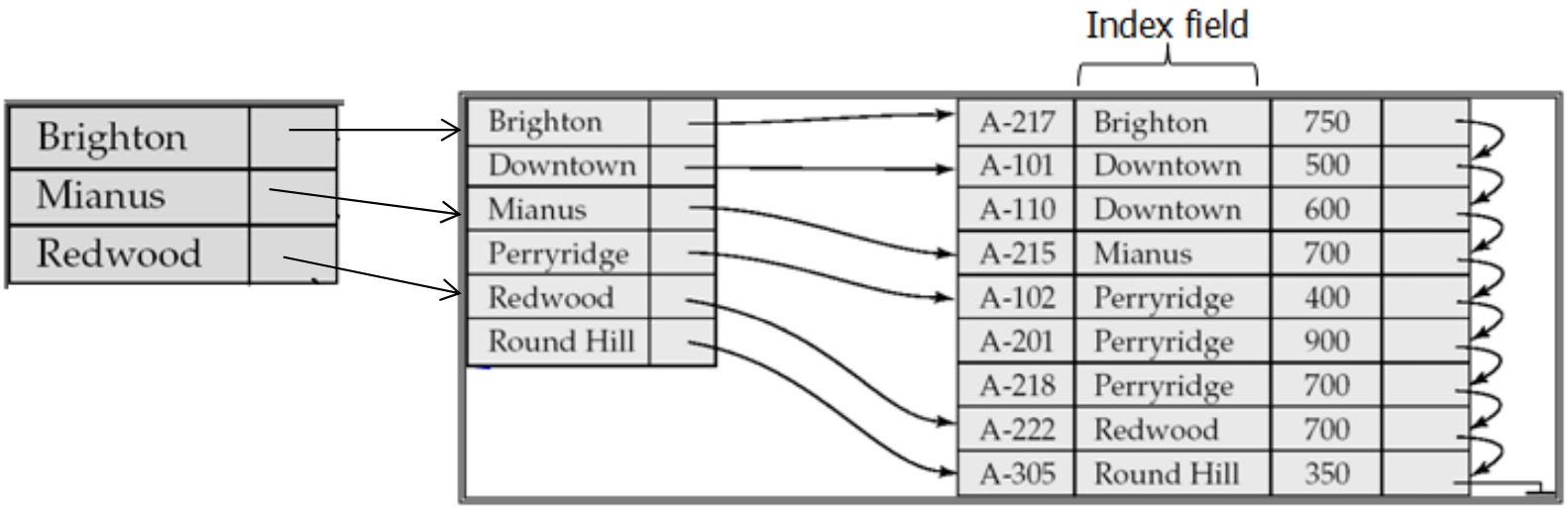
Primary *Sparse* Index

- Every key doesn't have to appear in the index
- Allows for very small indexes
  - Better chance of fitting in memory
  - Tradeoff: Must access the relation file even if the record is not present



Second-Level *Sparse* Index on Primary field

- When you do not have enough memory to load all of the primary index.
- Points to primary dense index

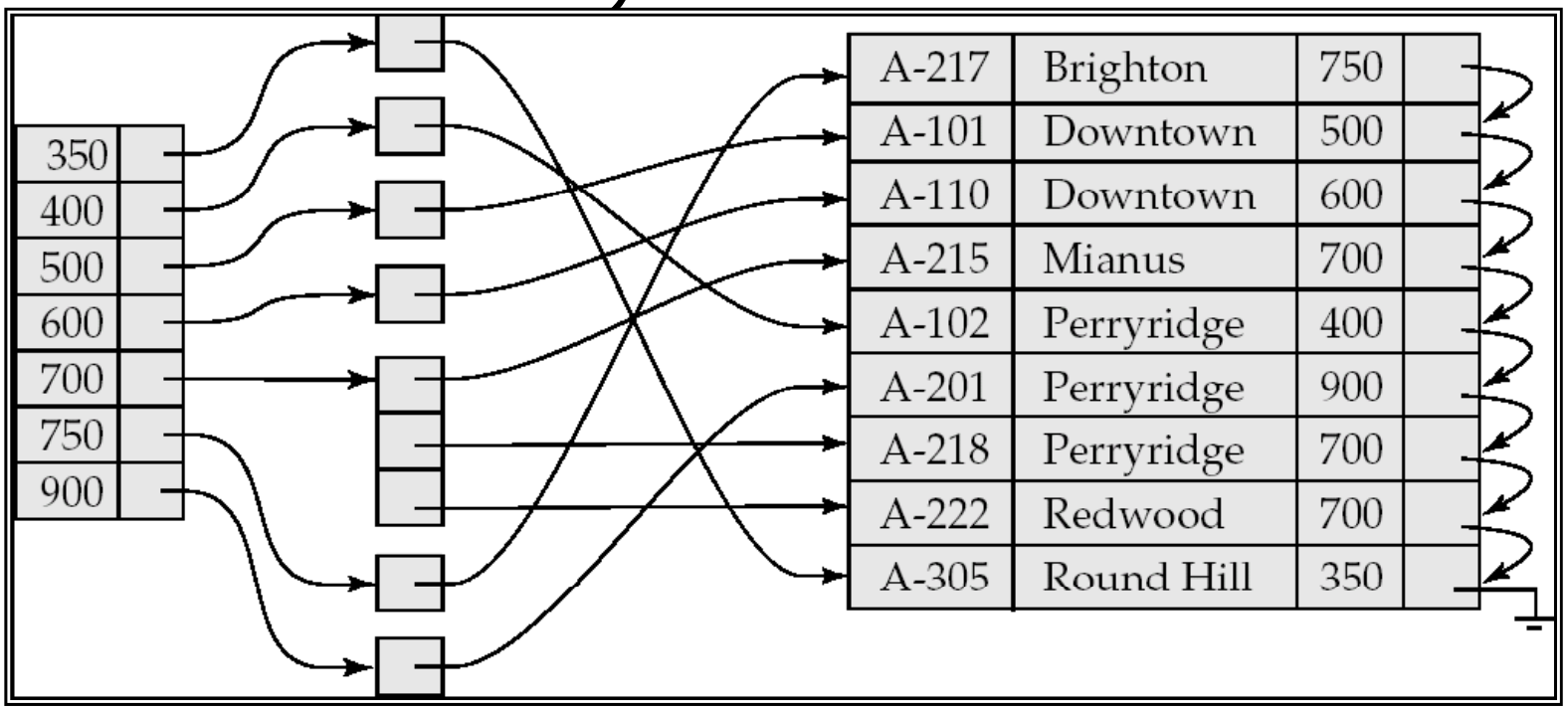


Secondary Index (always on non-Primary field)

- E.g. Relation sorted on *branch* but want an index on *balance*
- Secondary index must be dense. Link end of tuple won't help to go to next value in that secondary index.

Secondary Index (always on non-Primary field)

- If column used in secondary index is not unique, point to a list which points to the relevant tuples (else no need for list).



Second-level Secondary Index (always on non-Primary field)

- A second-level sparse secondary index must point to a dense secondary index of the same column

How to create an index in SQL ?

- Syntax

```
CREATE INDEX  Index-Name on Table-Name(Columns…);
```

- Example:

```
CREATE TABLE Customer  (First_Name char(50),
                        Last_Name char(50),
                        Address char(50),
                        City char(50),
                        Country char(25),
                        Birth_Date date);

CREATE INDEX  IDX_CUSTOMER_LAST_NAME on CUSTOMER (Last_Name);

CREATE INDEX IDX_CUSTOMER_LOCATION on CUSTOMER (City, Country);
```

How to delete an index in SQL ?

- Syntax

```
DROP INDEX  Index-Name;
```

- Example:

```
DROP INDEX  IDX_CUSTOMER_LAST_NAME;

DROP INDEX IDX_CUSTOMER_LOCATION;
```

- PostgreSQL tables

```
select * from pg_index;
```