Storage and Indexing

Memory Types

CITY UNIVERSITY LONDON
EST 1894

- DRAM
  - Volume. Up to 256 Gb
  - Cost. ~2-3$ per Gb
  - Access. 1-10 ns
- SSD
  - Volume. Up to 15 Tb
  - Cost. ~0.1$ per Gb
  - Access. 0.1-0.2 ms
- HDD
  - Volume. Up to 20 Tb
  - Cost. ~0.03$ per Gb
  - Access. 5-100 ms

Hard drives specificity

CITY UNIVERSITY LONDON
EST 1894

- Large search time
- Speed of reads
  - Consecutive - reasonable
  - Random - slow
- We need to reduce the number of accesses
  - Hopefully, making them consecutive

Pages

- The memory of hard drives is split into pages
- Read loads into cache
- Processing is faster
- Write is also slow
- It is better to store data in consecutive pages

Record load

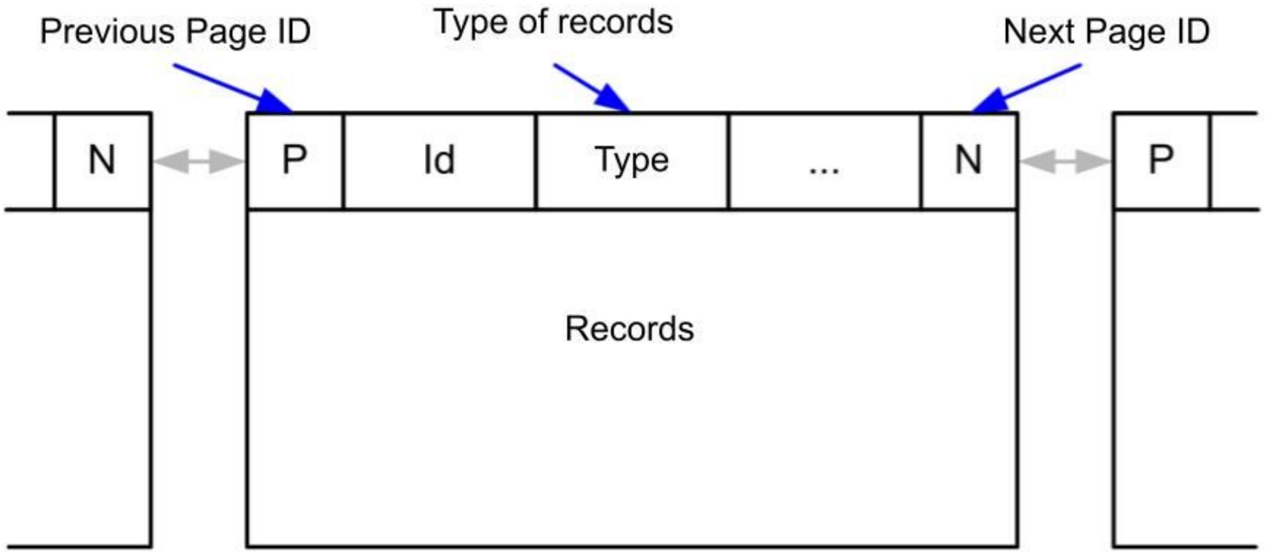
- We ask for a record
- Record Manager gives us the corresponding page
- Disk Manager loads the corresponding page

Data Storage

CITY UNIVERSITY LONDON
EST 1894

- Data is stored in files
- A file can contain several tables
- A table consists of several pages
- Page contains several records

Records management

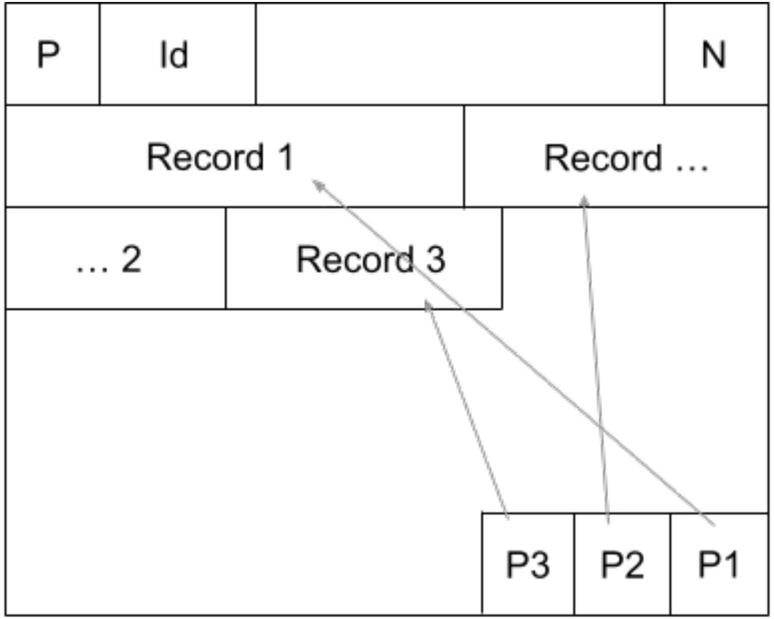


Record ID

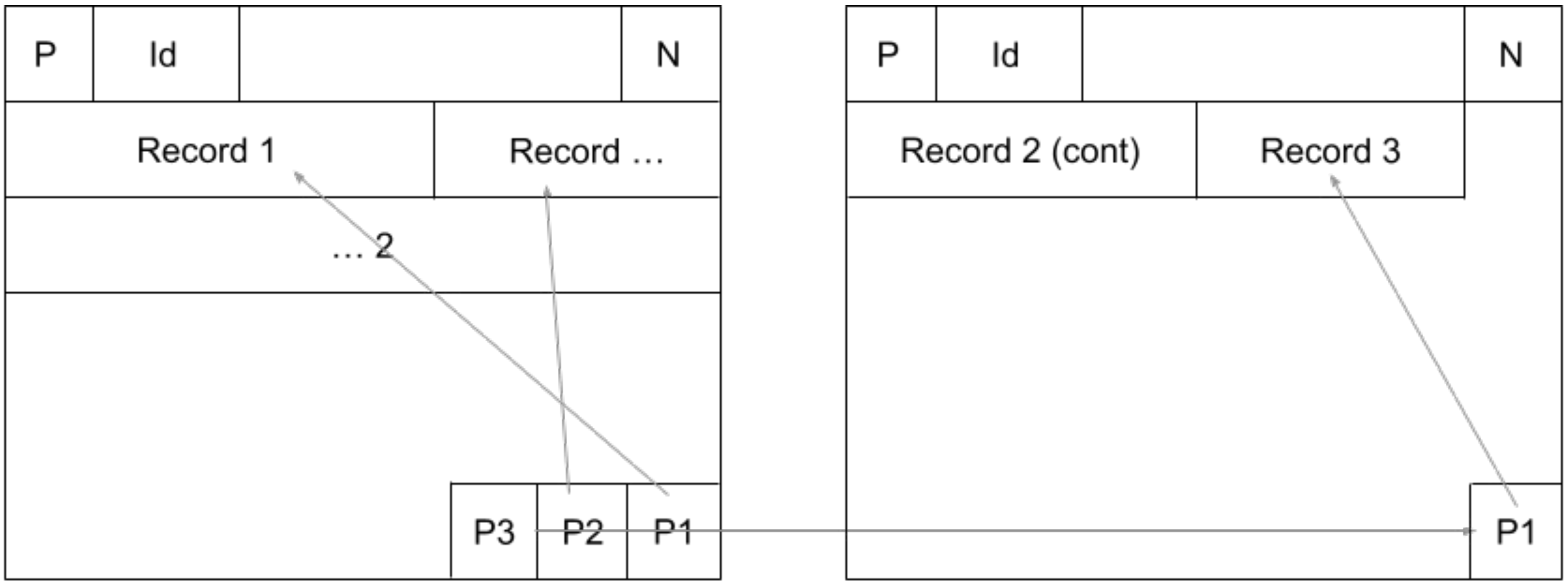
CITY UNIVERSITY LONDON
EST 1894

- Record ID (RID) consists of two parts:
  - Page ID
  - ID Record on the page
- We do not want it to change through time

Records on page



Records Overflow



Indexing

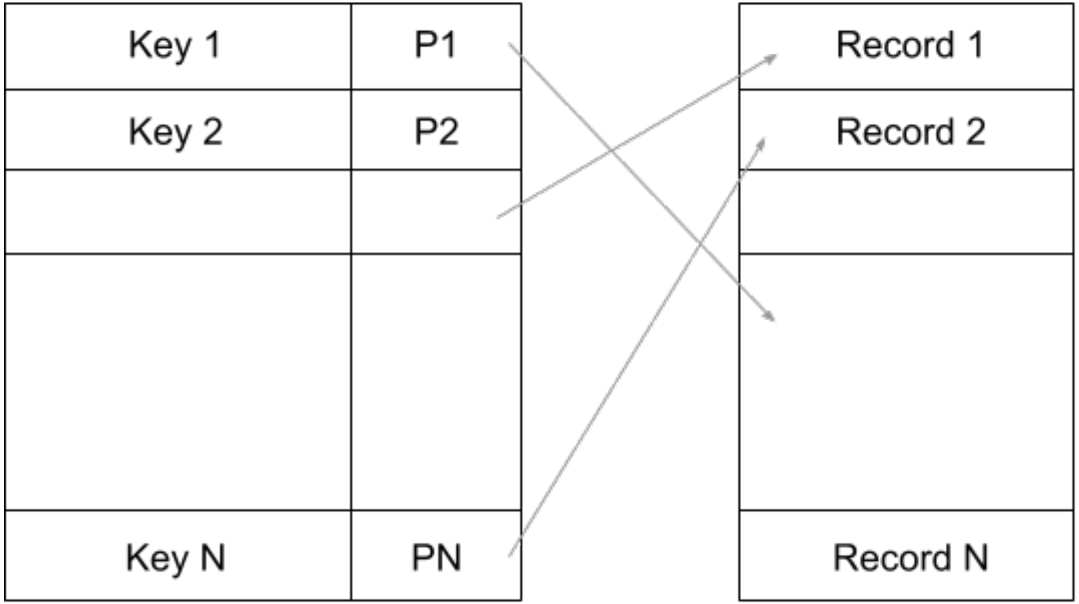
CITY UNIVERSITY LONDON
EST 1894

- There can be millions of records
- How to find the one necessary?
  - Search through the whole table
  - Have an additional data structure Index

Index

- Some subset of columns
- Not necessarily primary key
- Build at first and then updated
- Fast search gives the pointer to the record

Simple index

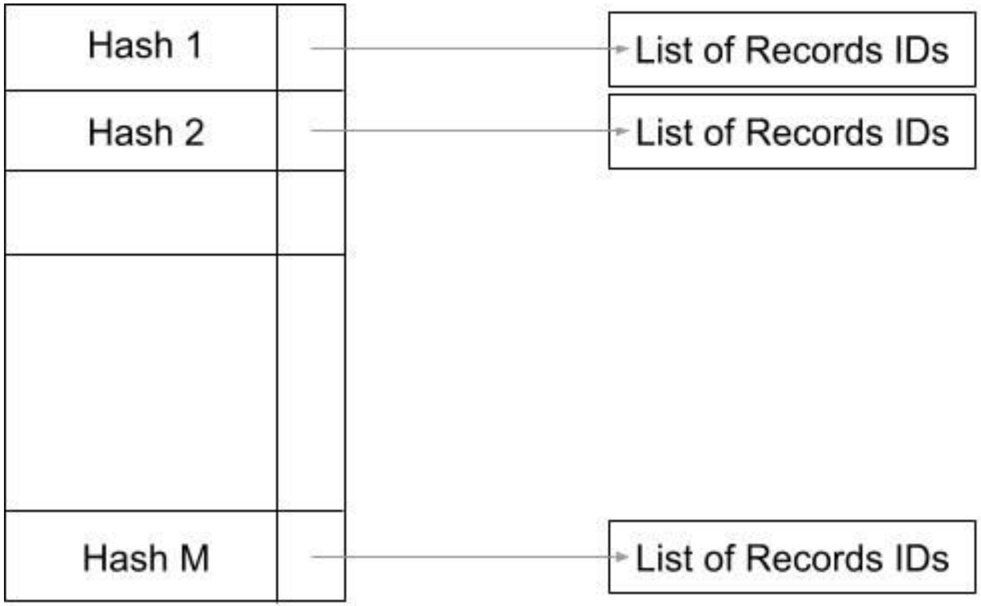


Why we do not want to store already in the sorted way: (Key, Record)?

Hash Index

- Each defined index tuple is transformed into some integer
- There is a table that given some integer returns a list of elements
- HashMap translated a tuple into integer and look into the corresponding element in the table
- Expected number of elements with the same hash and different values is O(1)

Hash Table


Hash 1
Hash 2
Hash M
List of Records IDs
List of Records IDs
List of Records IDs

Improved Queries

- Key uniqueness
- `IN`
- `EXISTS`
- `COUNT`
- `JOIN`

Ordered Index

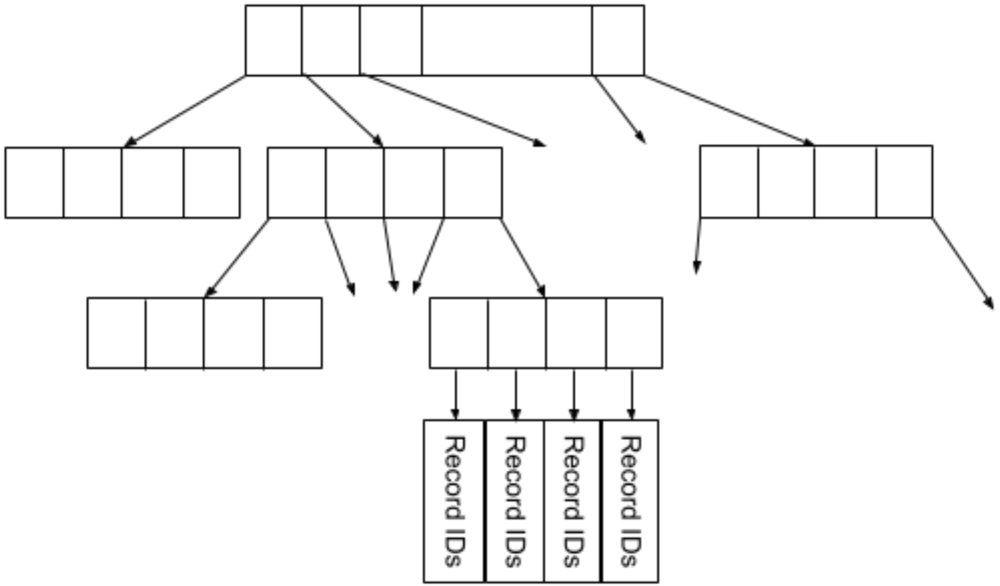
CITY UNIVERSITY LONDON
EST 1894

- The keys are sorted
- Search is like searching in the sorted sequence
- Can support range queries

For that we need a search tree.

In databases, typically, B-tree is used

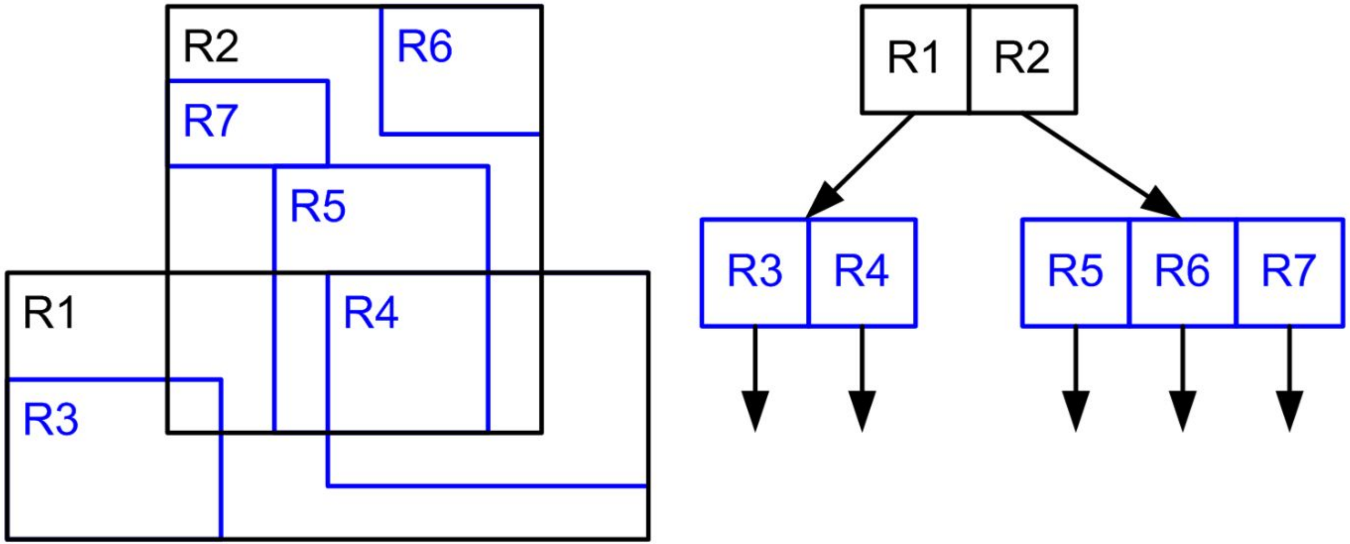
BTree



Multidimensional Index

CITY UNIVERSITY LONDON
EST 1894

- Suppose that the key of the index consists of several attributes
- One can use R-index
- It splits the dimension into cuboids



Index

- Usually, already exists for primary keys (how to check uniqueness?)
- Need to choose the representative ones with good selection
  - Surname is fine
  - Age is maybe ok
  - Gender is not really helpful
- The number of elements per each “bucket” should be not large
- Database can use some statistics to not use the asked index

Suggestions

- On keys
- On foreign keys
- For range queries
- For join operations

MySQL syntax

```
CREATE INDEX index_name {BTREE | HASH}
ON tbl_name (key_part,...);
```

Sources

C.J. Date, An Introduction to Database Systems

D. Knuth, The Art of Computer Programming, Volume 3, Sorting and Searching

A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts