THE UNIVERSITY OF MELBOURNE

INFO20003 Database Systems

Xiuge Chen

Tutorial 6

2021.04.12

Agenda

1. **Storage and Indexing review (key concepts with examples) - 30 min**
2. **Exercises - 30 min**

Storage and Indexing

Why?

- Database management systems store information on disks (normally hard disks)
- involves many READ and WRITE operations when data is accessed: high cost
- **READ**: transfer of data from the disk to main memory (RAM)
- **WRITE**: transfer data from RAM to the disk

Storage and Indexing

alternative terms used with respect to disk storage

Conceptual modelling	Entity	Attribute	Instance of an entity
Logical modelling	Relation	Attribute	Tuple
Physical modelling/SQL	Table	Column/Field	Row
Disk storage	File	Field	Record

Storage and Indexing

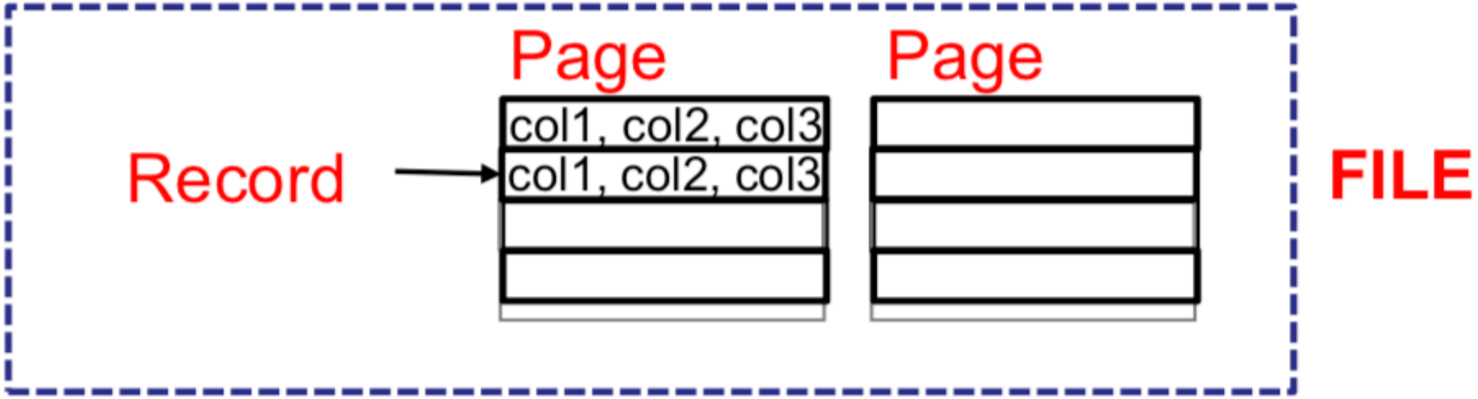
Files, pages and records

- **record**: an individual **row** of a table and has a unique *rid* (disk address of the page containing the record).

  ex. *rid* (3, 7) refers to the seventh record from third page
- **page**: an allocation of space on disk or in memory containing a collection of **records**. (every page is the same size)
- **file**: a collection of **pages** containing records (In simple database scenarios: single table)

Storage and Indexing

Files, pages and records



Storage and Indexing

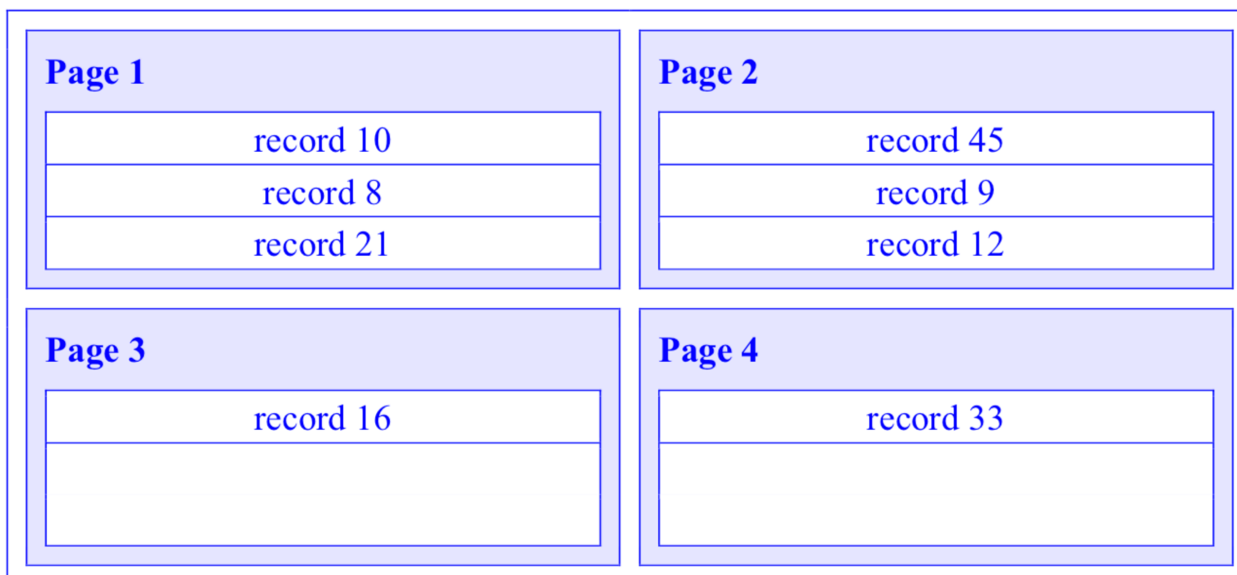
File organisations

- defines how file records are mapped onto pages (stored on disk).
- Heap file organization:
- Sorted file organization
- Index file organization

Storage and Indexing

Heap file organisation:

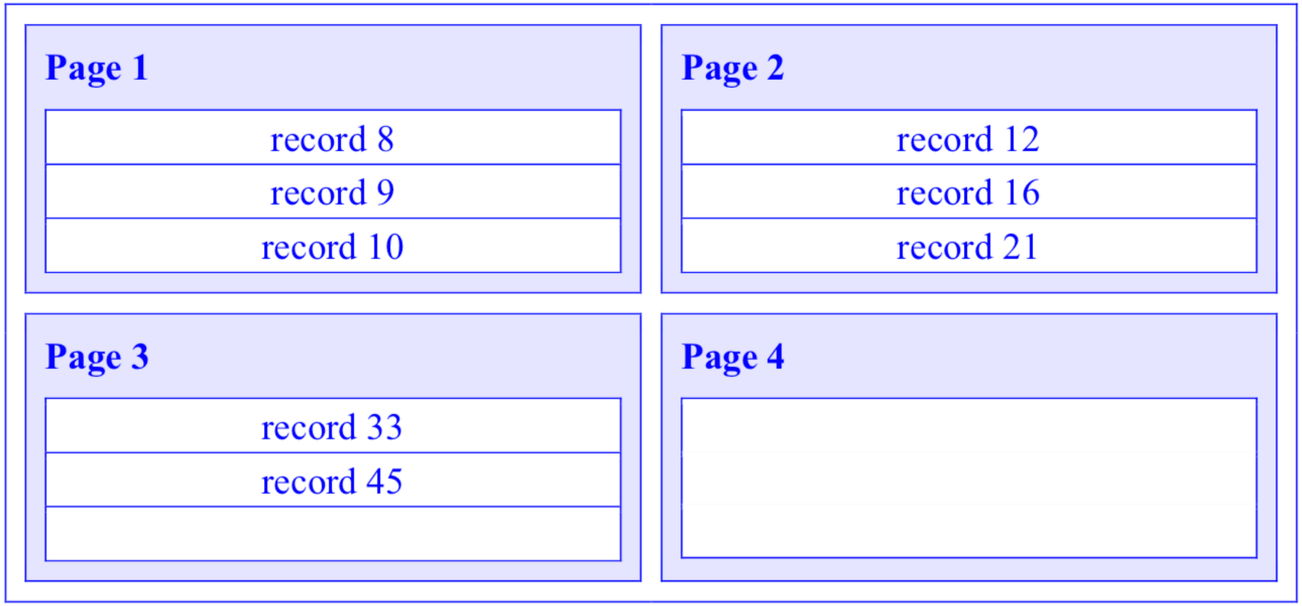
- No ordering, sequencing or indexing
- Suitable when retrieving all records
- Slow search
- Quick insert



Storage and Indexing

Sorted file organization:

- sequential order based on the ***search key*** (not PK or FK)
- Quick search for search key(especially on range)
- Slow insert



Storage and Indexing

Index file organization:

- Any **subset** of the fields of a table is indexed based on queries that are frequently run against the database
- Quick search on the subset attributes
- Different index could be built
- Different types index could be chosen
- Insert depend on types

Storage and Indexing

What is index?

- made up of data entries which refer back to the data in the relation. (*k*, *rid*) *k:* search key, *rid:* record ID.
- speeds up selection on the search key fields
- ***search key***: subset of the attributes of a relation on which the index is built (not be relation's key!!!)
- stored in an *index file*, in contrast to the *data file* which contains the actual records themselves

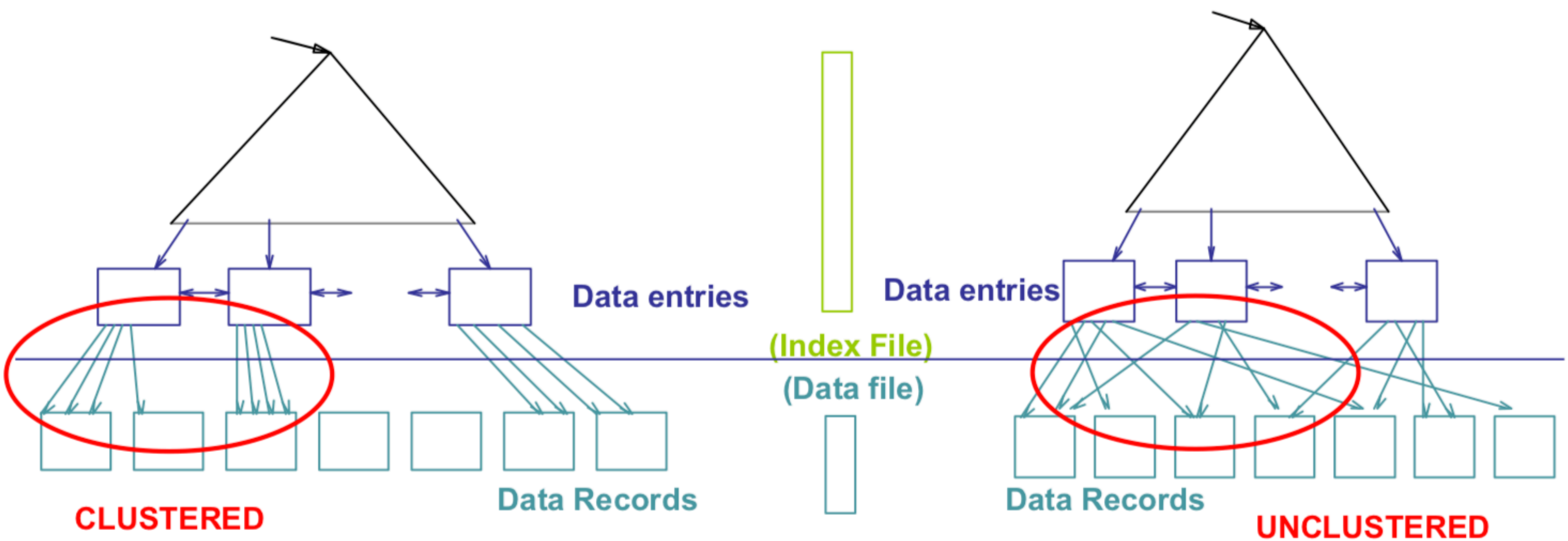
Storage and Indexing

Type of index 1

- **clustered**: data records in the data file have the **same order** as data entries of the index
- **unclustered**: data records in the data file are **not sorted by** search key/data entries
- **primary**: on the primary key of the relation
- **secondary**: on any other set of attributes

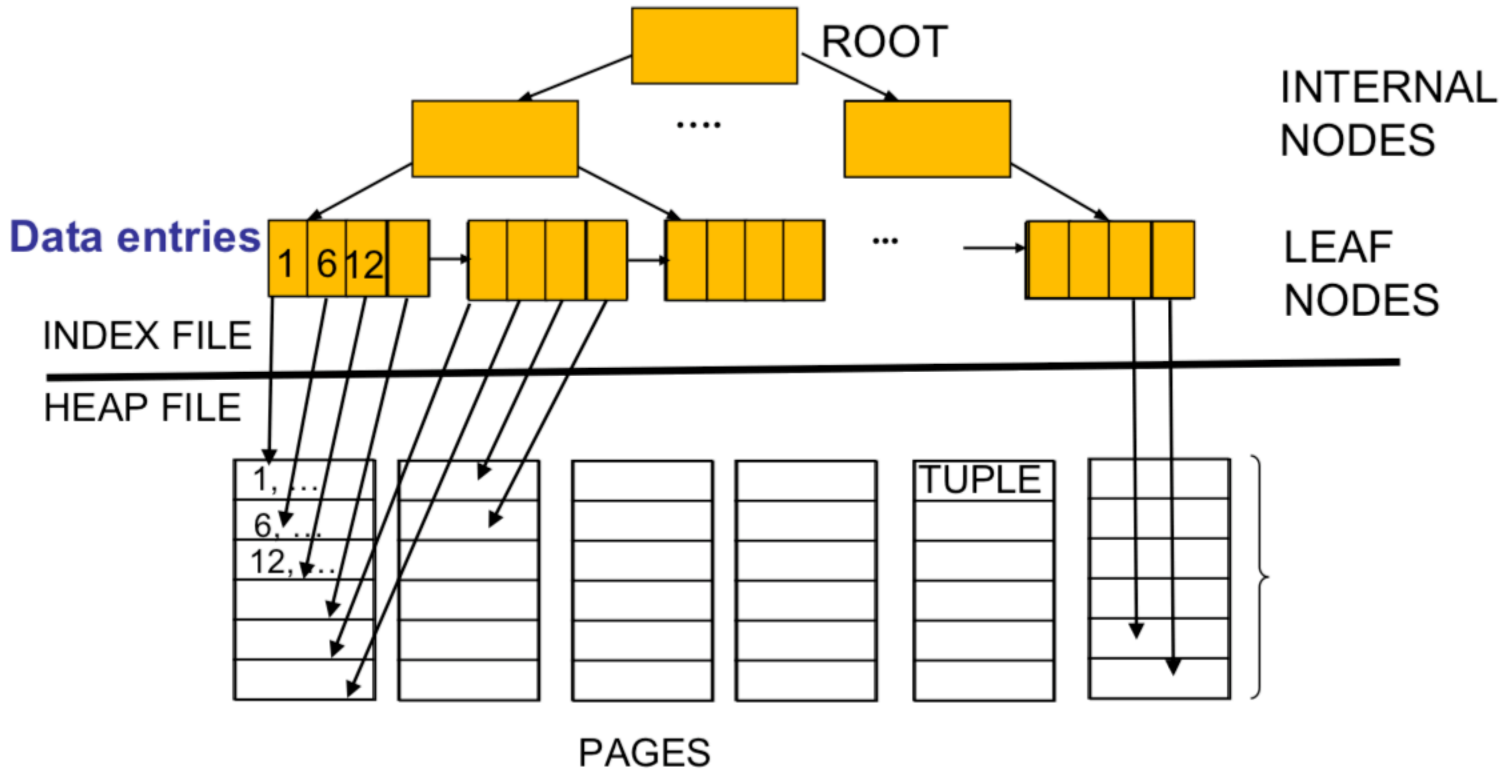
Storage and Indexing

Type of index 1



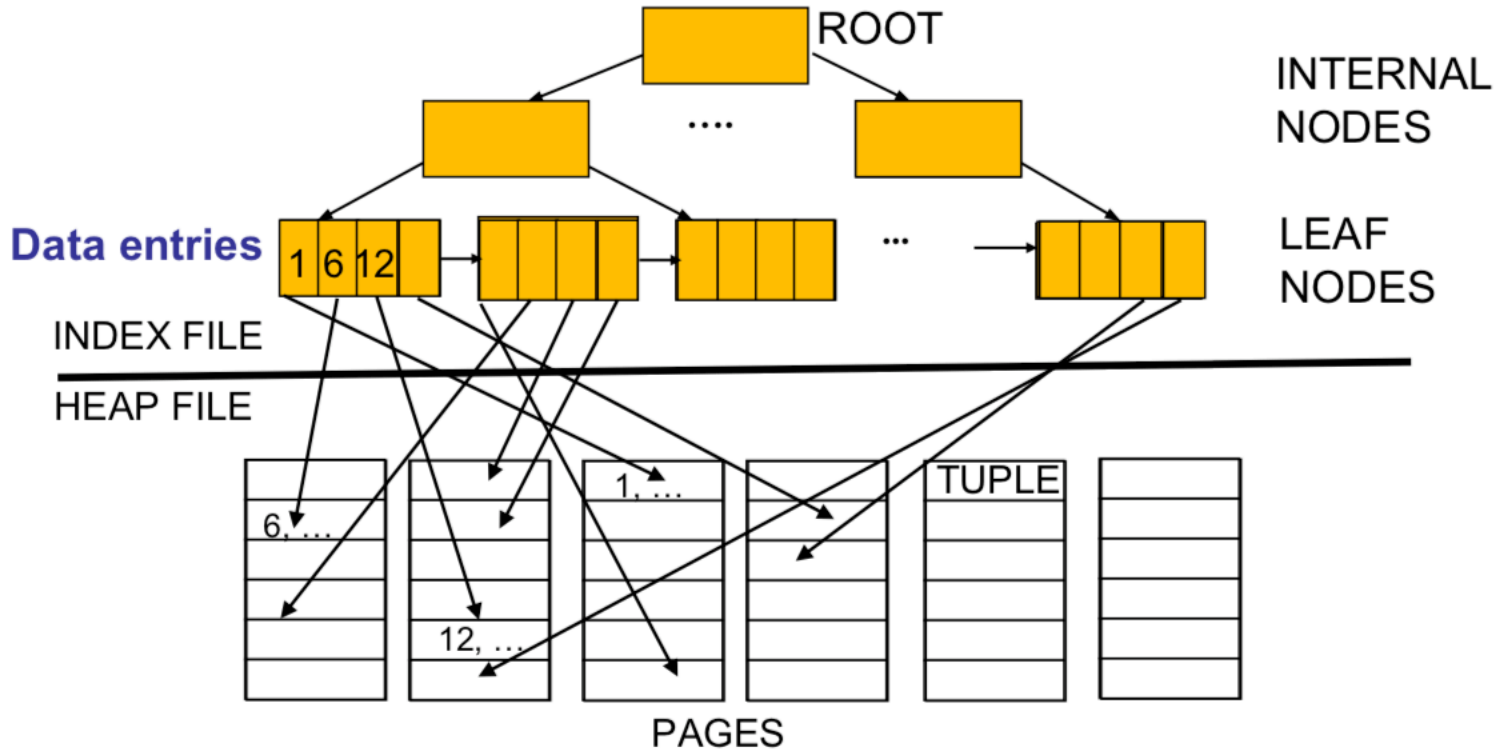
Storage and Indexing

Type of index 1



Storage and Indexing

Type of index 1



Storage and Indexing

How to choose index

- Which relations are accessed frequently?
- Which attributes are retrieved?
- Which attributes are involved in selection, join and other conditions?
- If a query involves updating the relation, what attributes are affected?

Storage and Indexing

Index

- Coming: how to analyze a given query plan and see if a better query plan exists with an additional index
- **In general**: make SELECT queries faster but slow down the updates
- Indexes also require **additional disk space**.
- carefully analyzed before constructing an index!!!

Storage and Indexing

Type of index 2

- **Hash-based indexing**: hash function $h(r)$ is applied, where r is the field value.
- **Output**: point to a bucket which refers to the primary page and other overflow pages if there is any. These buckets contain a representation (k, rid) for data entries.
- best suited to support *equality* selections
- How to build: not in this subject

Storage and Indexing

Hash index

- Suppose you are given 5 buckets and $h(k) = k \% 5$ where % is the modulus (remainder) operator. Insert 200, 22, 119, 8, and 33 into a hash table.

Bucket	Key
0	200
1	
2	22
3	8, 33
4	119

Storage and Indexing

Hash index

- https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

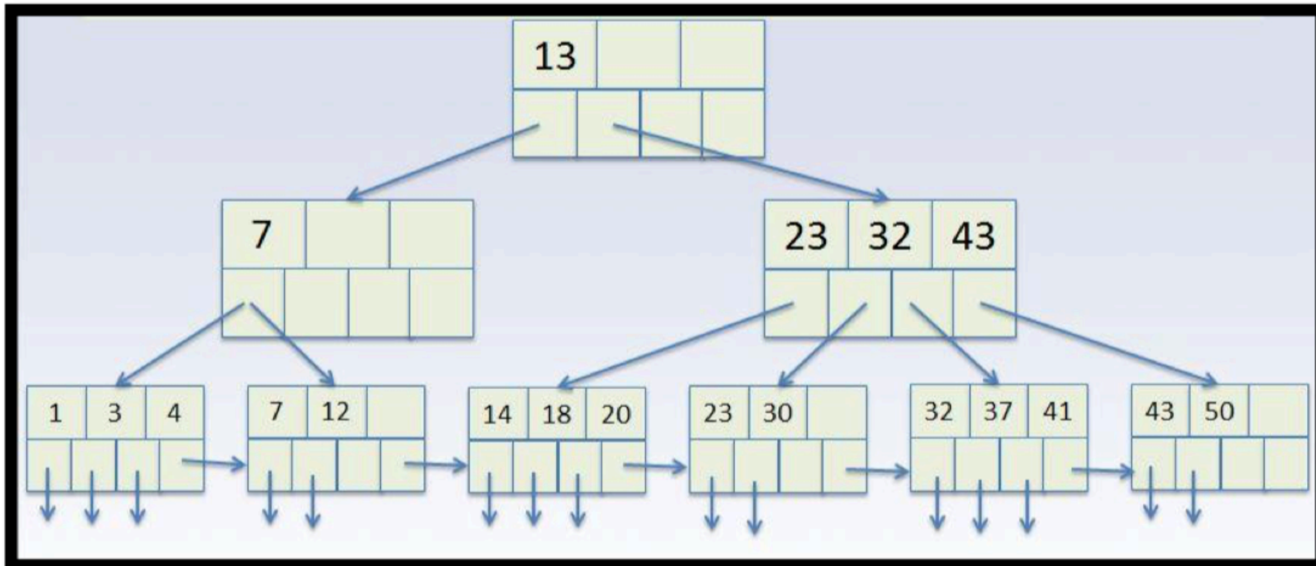
Storage and Indexing

Type of index 3

- **B-tree index**: sorting data on search key and maintaining a hierarchical search data structure (B+ tree) that will direct the search to the respective page of the data entry
- **Insertion** in such a structure is **costly** as the tree is updated with every insertion or deletion.
- Good for: equal or range selections
- How to build: not in this subject

Storage and Indexing

B-tree index



- Start at the root.
- Internal nodes, search the keys to find the range K belongs in and follow that pointer.
- For leaf nodes (nodes with no child nodes), search the keys to find K and follow the pointer to find the data record.

Storage and Indexing

B-tree index

- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Storage and Indexing review

Any questions?

Exercises

1. Choosing an index

You are asked to create an index on a suitable attribute. What are the important aspects you will analyse to make this decision? To get you started, the following might help you by providing scaffolding to the discussion:

a. Primary vs. secondary index

Primary: records are retrieved based on the value of primary key.
Secondary: fields that are frequently queried.

Generally, a table should always have a primary index (in fact, MySQL creates one automatically).

Exercises

b. Clustered vs. unclustered index

Clustered: consists of a frequently-executed condition to check for a **range,** however expensive to maintain
Unclustered: fields that are frequently queried.

Equality conditions: same if the search key does not have duplicate values.

More than one combination of columns is used in range queries, choose the **most frequently** used combination and make those fields search keys of the clustered index

Exercises

c. Hash vs. tree indexes

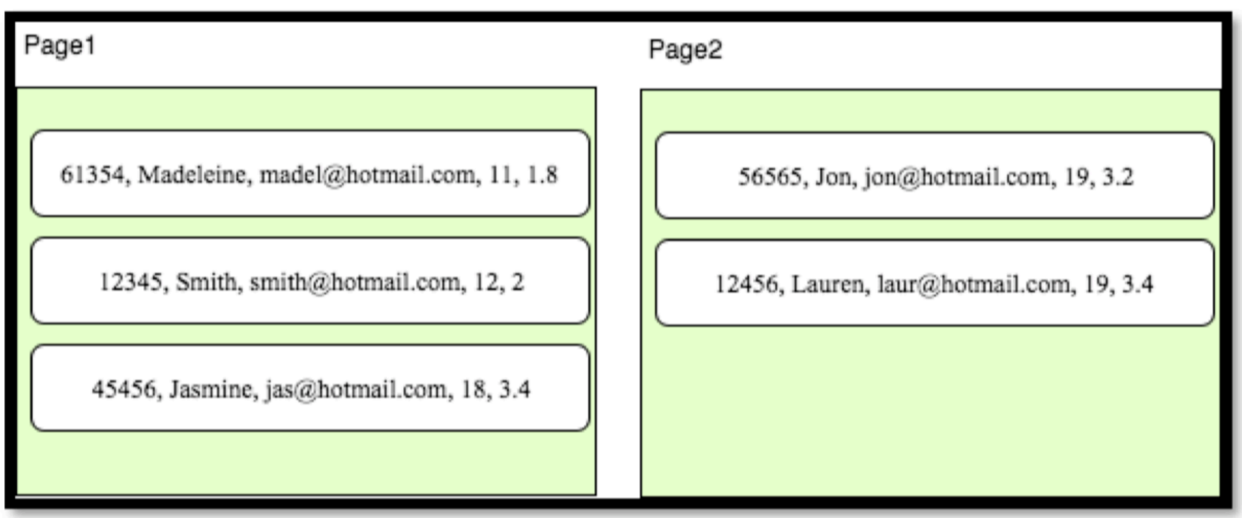
Hash: **equality** queries, faster than B-tree

Tree: **range** queries, creating a B-tree index

Exercises

1. Data entries of an index:

SID	Name	Email	Age	GPA
61354	Madeleine	madel@hotmail.com	11	1.8
12345	Smith	smith@hotmail.com	12	2.0
45456	Jasmine	jas@hotmail.com	18	3.4
56565	Jon	jon@hotmail.com	19	3.2
12456	Lauren	laur@hotmail.com	19	3.4



1. tuples sorted by age

2. order of tuple is the same when stored on disk

3. each page can contain only 3 records

Exercises

1. Data entries of an index:

Show what the *data entries* of the index will look like for:

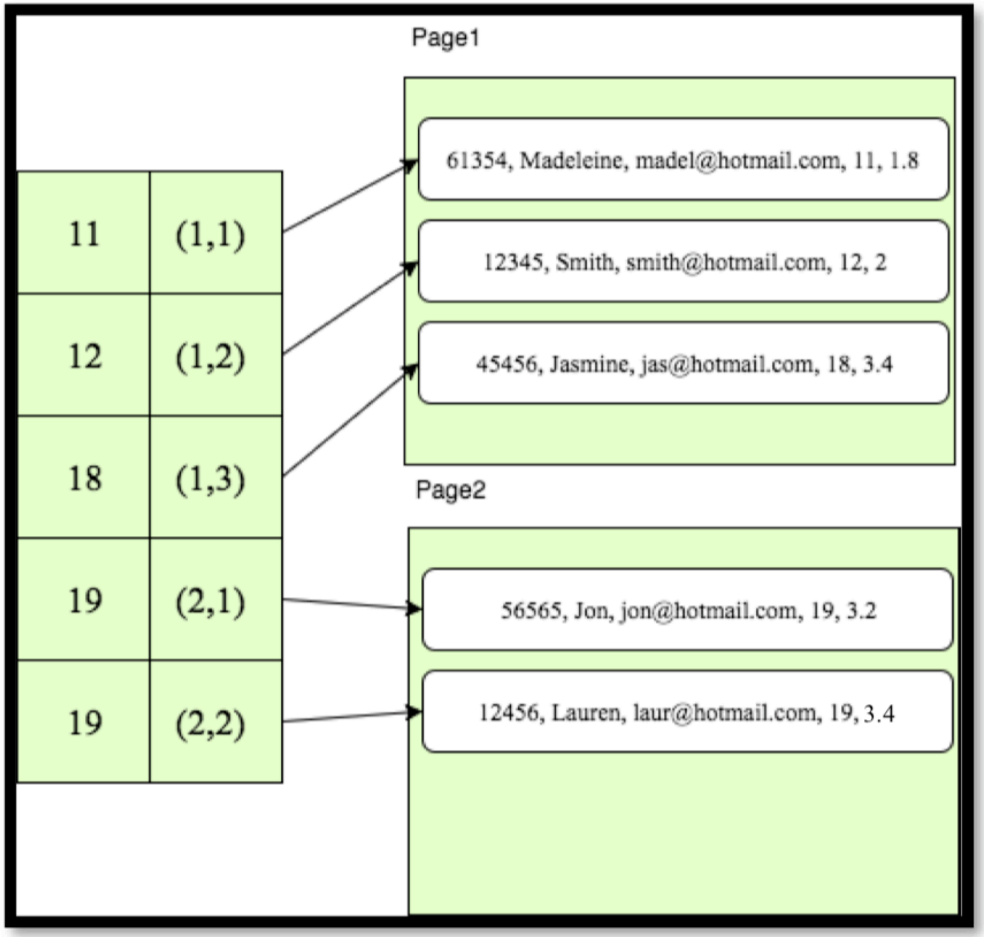
a. An index on Age

search key and *rid* in the format (*a*, *b*)

a is the page number and

b is the record number.

clustered index



Exercises

1. Data entries of an index:

Show what the *data entries* of the index will look like for:

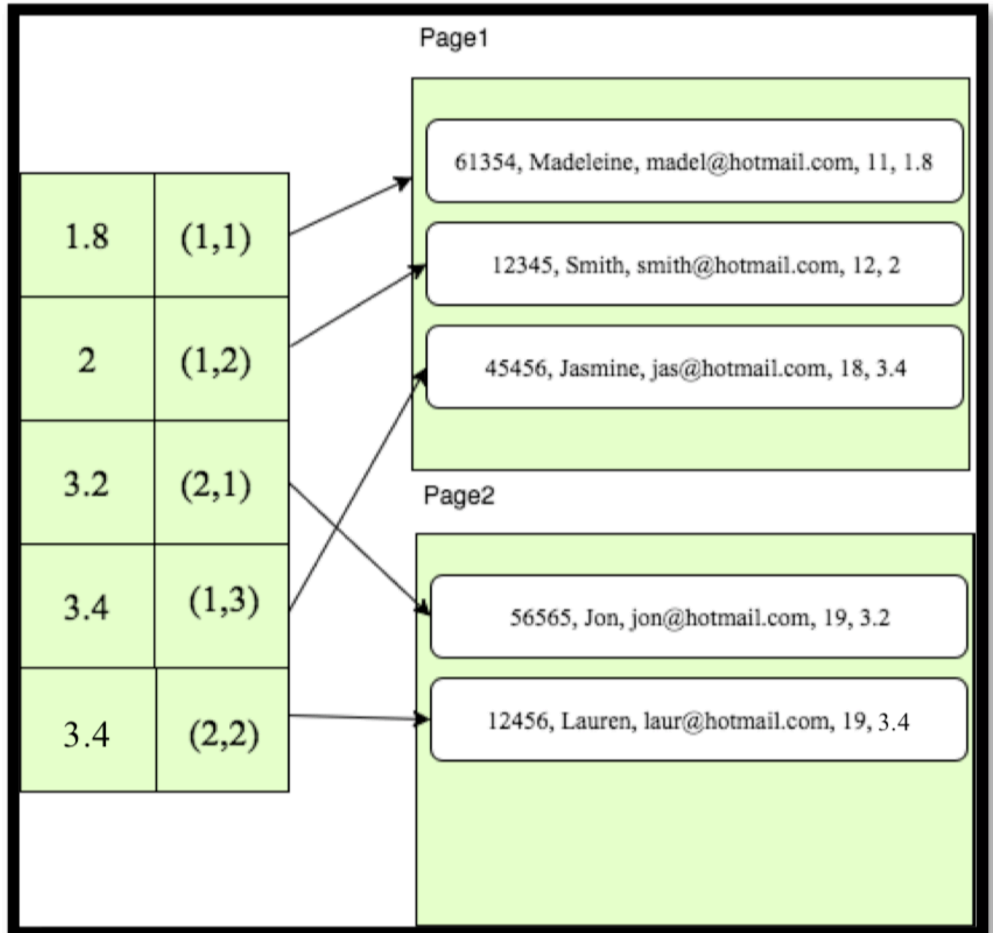
b. An index on GPA

search key and *rid* in the format (*a*, *b*)

a is the page number and

b is the record number.

unclustered index



Exercises

1. Consider the following relations:

Employee (EmployeeID, EmployeeName, Salary, Age, DepartmentID (FK))

Department (DepartmentID, DepartmentBudget, DepartmentFloor, ManagerID (FK))

In the database, the salary of employees ranges from AUD10,000 to AUD100,000, age varies from 20-80 years and each department has 5 employees on average. In addition, there are 10 floors, and the budgets of the departments vary from AUD10,000 to AUD 1million.

Given the following two queries frequently used by the business, which index would you prefer to speed up the query? Why?

Exercises

1. Consider the following relations:

Employee (<u>EmployeeID</u>, EmployeeName, Salary, Age, DepartmentID) — FK: DepartmentID

Department (<u>DepartmentID</u>, DepartmentBudget, DepartmentFloor, ManagerID) — FK: ManagerID

a. **SELECT**DepartmentID
FROM Department
WHERE DepartmentFloor = 10
AND DepartmentBudget < 15000;

A) Clustered hash index on DepartmentFloor
B) Unclustered hash Index on DepartmentFloor
C) Clustered B+ tree index on (DepartmentFloor, DepartmentBudget)
D) Unclustered hash index on DepartmentBudget
E) No need for an index

Exercises

1. Consider the following relations:

Employee (<u>EmployeeID</u>, EmployeeName, Salary, Age, DepartmentID) — FK: DepartmentID

Department (<u>DepartmentID</u>, DepartmentBudget, DepartmentFloor, ManagerID) — FK: ManagerID

a. **SELECT**DepartmentID
FROM Department
WHERE DepartmentFloor = 10
AND DepartmentBudget < 15000;

A) Clustered hash index on DepartmentFloor
B) Unclustered hash Index on DepartmentFloor
C) Clustered B+ tree index on (DepartmentFloor, DepartmentBudget)
D) Unclustered hash index on DepartmentBudget
E) No need for an index

Range query!

Exercises

1. Consider the following relations:

Employee (<u>EmployeeID</u>, EmployeeName, Salary, Age, DepartmentID (FK))

Department (<u>DepartmentID</u>, DepartmentBudget, DepartmentFloor, ManagerID (FK))

b. **SELECT** EmployeeName,Age,Salary
FROM Employee;

A) Clustered hash index on (EmployeeName, Age, Salary)
B) Unclustered hash index on (EmployeeName, Age, Salary)
C) Clustered B+ tree index on (EmployeeName, Age, Salary)
D) Unclustered hash index on (EmployeeID, DepartmentID)
E) No need for an index

Exercises

1. Consider the following relations:

Employee (<u>EmployeeID</u>, EmployeeName, Salary, Age, DepartmentID) — FK: DepartmentID

Department (<u>DepartmentID</u>, DepartmentBudget, DepartmentFloor, ManagerID) — FK: ManagerID

b. **SELECT** EmployeeName,Age,Salary
FROM Employee;

A) Clustered hash index on (EmployeeName, Age, Salary)
B) Unclustered hash index on (EmployeeName, Age, Salary)
C) Clustered B+ tree index on (EmployeeName, Age, Salary)
D) Unclustered hash index on (EmployeeID, DepartmentID)
E) No need for an index

get requested attributes with an index-only scan (and we can avoid accessing the table completely)

Exercises

Any questions?