Database Systems

Tutorial Week 6

Objectives

I. Storage and indexing review
II. Exercises

Background

- DBMS stores information on disks (normally hard disks)
- Involves many READ and WRITE operations when data is accessed
- READ operation: transfer of data from disk to main memory (RAM)
- WRITE operation: transfer of data from RAM to disk
- Both high cost, but required for storing data on secondary storage

Records, Pages, Files

- Record
  - An individual row of a table
  - Has a unique *rid*
    - Identifies where the record is
      - Which page and which record on the page
    - *Rid* = (pg #, record # on page)
    - E.g. *rid* = (3, 7) refers to the 7th record on the 3rd page

Records, Pages, Files

- Page
  - An allocation of space on disk / in memory
  - Contains records
  - Typically, every page is the same size

Records, Pages, Files

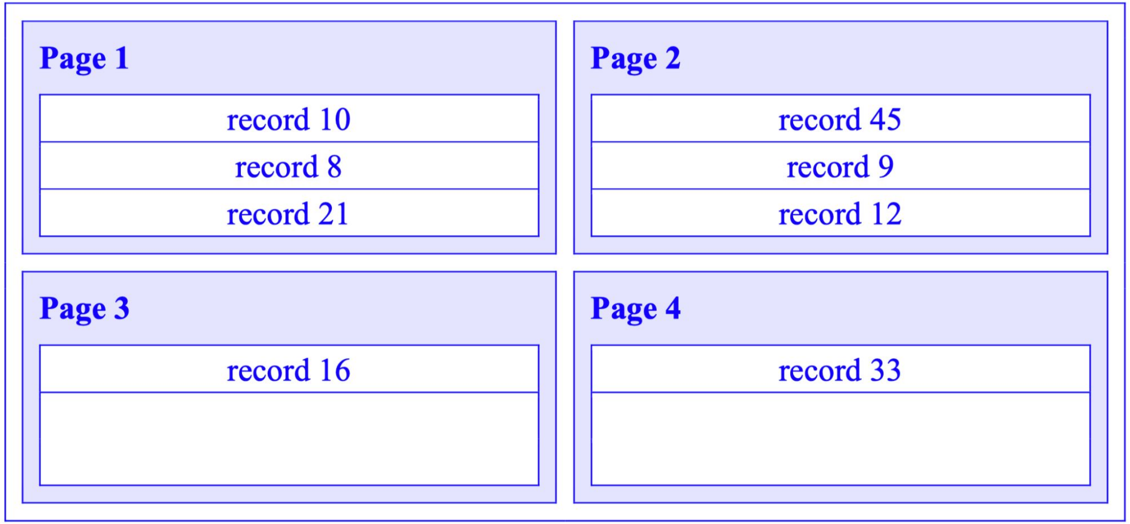
- File
  - A collection of pages containing records
  - Corresponds to a single table

File Organisations

- A file organisation determines how records are stored on disk / mapped onto pages
- Typically, files are organised in 3 ways
- What are they???
  - Heap file organisation
  - Sorted file organisation
  - Index file organisation

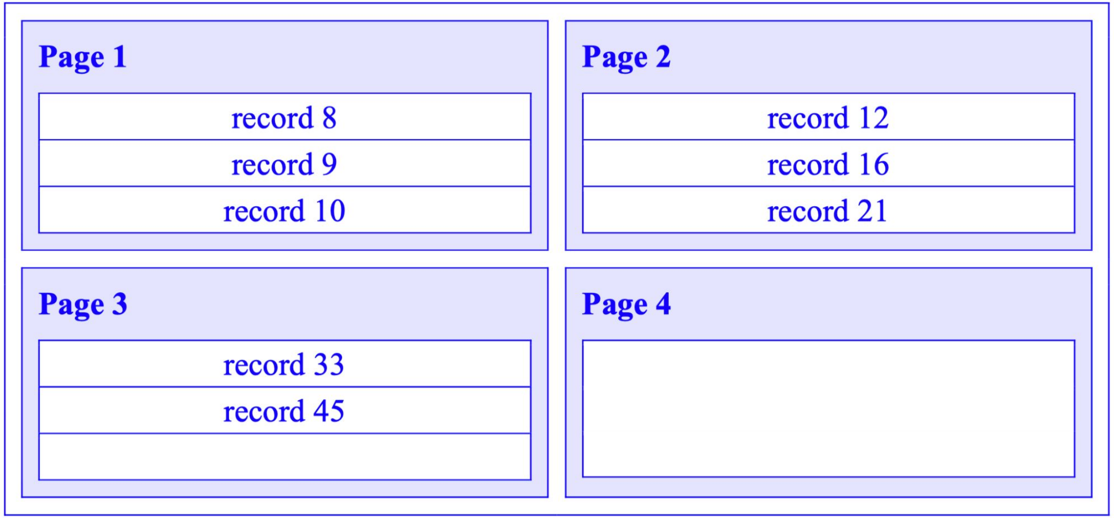
File Organisations

- Heap file organisation
  - Heap file records are stored in an arbitrary, unsorted manner
  - Heap file records are placed anywhere in allocated memory area (next “available” slot)



File Organisations

- Sorted file organisation
  - Almost the same structure as a heap file organisation
  - But records are ordered!



File Organisations

- Index file organisation
  - When you have an index over the records
  - Used for efficient querying

What is an Index???

- Consider a book with an index in the back
  - You find a keyword, and then the relevant pages containing that keyword
  - Keywords are in alphabetical order
  - Makes it easy for us to find the keyword and relevant pages quickly
- Similarly, database indexing speeds up queries
- So, what’s a database index?

What is an Index???

- An index is a data structure built on top of data pages
- It's built over specific field(s), *search key fields*
  - Search key ≠ "key" in modelling topic!
  - Search keys don't have to be unique
  - Any subset of fields can be the search key for an index
  - **The index speeds up selections on the search key fields**
- E.g. find all students with a GPA > 3
  - We can build an index on "GPA"
  - "GPA" would be the search key
  - This index speeds up selections on GPA

What is an Index???

- An index is made up of data entries which refer to the data in the actual relation
- Data entries are usually represented as (*k*, *rid*) where *k* = search key value and *rid* = record ID
    - E.g. search key is “age” and the 7th record on page 3 has age = 10
    - This data entry would be represented as (10, (3, 7))
- Indexes are stored in an *index file*
    - Compare to the *data file* which contains the actual records

What is an Index???

- Indexes can be *clustered* or *unclustered*
  - Clustered index
    - *Data records* in the data file and *data entries of the index* in the index file have the **same order**
  - Unclustered index
    - *Data records* in the data file and *data entries of the index* in the index file **don't** have the same order

What is an Index???

- Indexes can be *primary* or *secondary*
  - Primary index
    - Indexes are built on the primary key of the relation
    - I.e. search key = primary key
  - Secondary index
    - Indexes are built on any other set of attributes that aren’t the primary key of the relation
    - I.e. search key ≠ primary key

What is an Index???

- Indexes make
  - SELECT queries on the search key faster
    - Consider book analogy
  - UPDATES slower
    - For example, you need to update your B+ tree and rebalance it with every insertion or deletion
- They also require additional disk space

Hash-Based Indexing

- Use a hash function *h(r)* to index records, where *r* = value of the search key
- Hash index maps search key value to a bucket corresponding to the output of *h(r)*
- Buckets contain a representation (*k, rid*) for data entries
- Suppose *h(r) = r%5* (the remainder once you divide by 5)
- Insert 200, 22, 119, 8 and 33 into a hash table

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

Hash-Based Indexing

- Now, find the record where $r = 200$
- Apply $h(200) = 200\%5 = 0$
- Look in bucket 0 which will have (200, *rid*) — *rid* will point you to the record where $r = 200$

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

Hash-Based Indexing

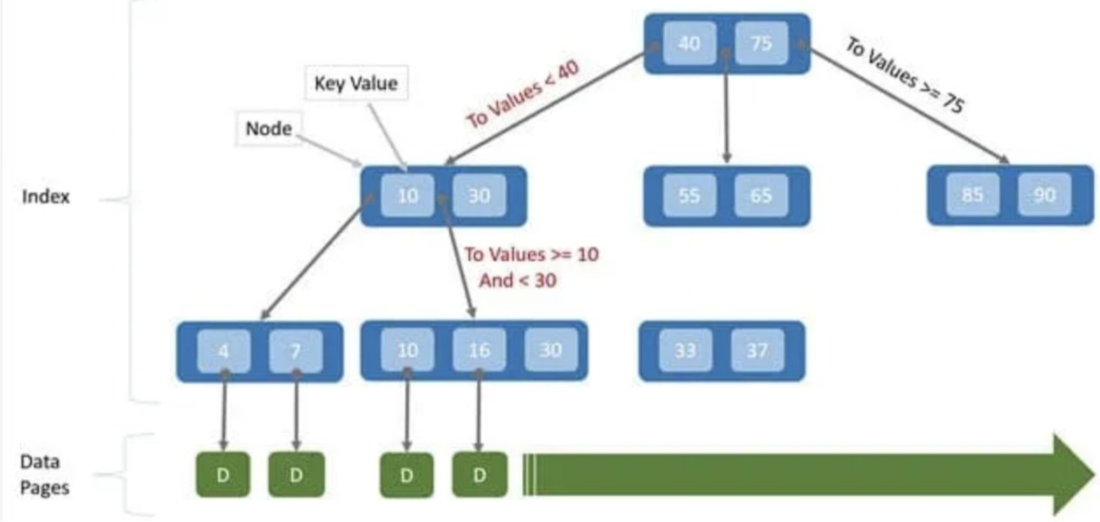
- Hash-based indexing is best for ***equality*** selections!!!
  - WHERE x = y
- Usually unclustered

B-Tree Indexing

- B-tree index is created by sorting data on a *search key* and maintaining a B+ tree
    - B+ tree is a self-balancing tree
        - LHS height differs to RHS height by at most one
    - A tree data structure where records are found by traversing child nodes and making comparisons
    - Search left of the tree for lower values and right for higher values
    - Leaves of the tree contain index data entries

B-Tree Indexing

- Find the record(s) where $k < 9$
  - 9 < 40, go left and 9 < 10, go left
  - Found $k = 4$ and $k = 7$
    - These leaves contain (4, *rid)* and (7, *rid*), where *rid* points to the records where $k = 4$ and $k = 7$



B-Tree Indexing

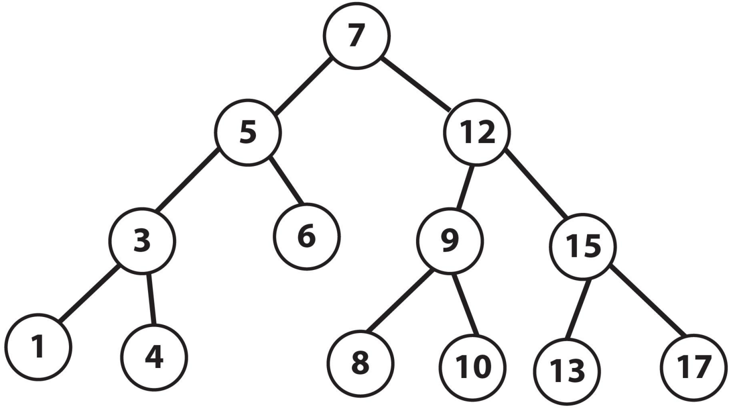
- Cost of insertion: fast + cost to balance
- Cost of deletion: fast + cost to balance
- Cost of querying: super fast

- Use for ***range*** queries!!!
    - WHERE x > y or x < y

- When in doubt, go with B+ tree

B-Tree Indexing

- B+ trees are excellent for *index-only scans*
  - An index-only scan is an index scan without subsequently accessing data pages
- If we only need the index, then we can just traverse the tree
- E.g. SELECT id FROM Relation WHERE id > 12
  - Returns 13, 15 and 17
- Don’t need to access data pages since we have all the information we need already



B-Tree Indexing

[B+ Tree Visualisation]

Exercises

Q1. Choosing an index (5 mins)

You are asked to create an index on a suitable attribute. What are the important aspects you will analyse to make this decision? Consider:

- Primary vs. secondary index
- Clustered vs. unclustered index
- Hash vs. tree indexes

Exercises

Q1. 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? Consider:

- Primary vs. secondary index
    - Choose primary index when records are often retrieved based on value of PK
    - Choose secondary index when there are attribute(s) other than the PK frequently used in queries

Exercises

Q1. 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? Consider:

- Clustered vs. unclustered index
    - Clustered index
        - Can only be applied to a table that uses sorted file organisation
        - Preferred when **range queries** are frequently executed e.g. a > b or a < b
        - Also speeds up equality queries
        - But expensive to maintain — update/delete/insert might lead to reordering of records
    - Unclustered index
        - Preferred when **equality queries** are frequently executed
            - We don’t care about ordering
            - Less expensive to maintain

Exercises

Q1. 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? Consider:

- Hash vs. tree indexes
  - Choose a hash index
    - When your query is frequently of the type x = y
    - Slow for range queries
  - Choose a B+ tree index
    - When your query is frequently of the type x > y or x < y
    - Also fast for x = y, but slower than hash-based indexing
- When in doubt, B+ tree is the answer

Exercises

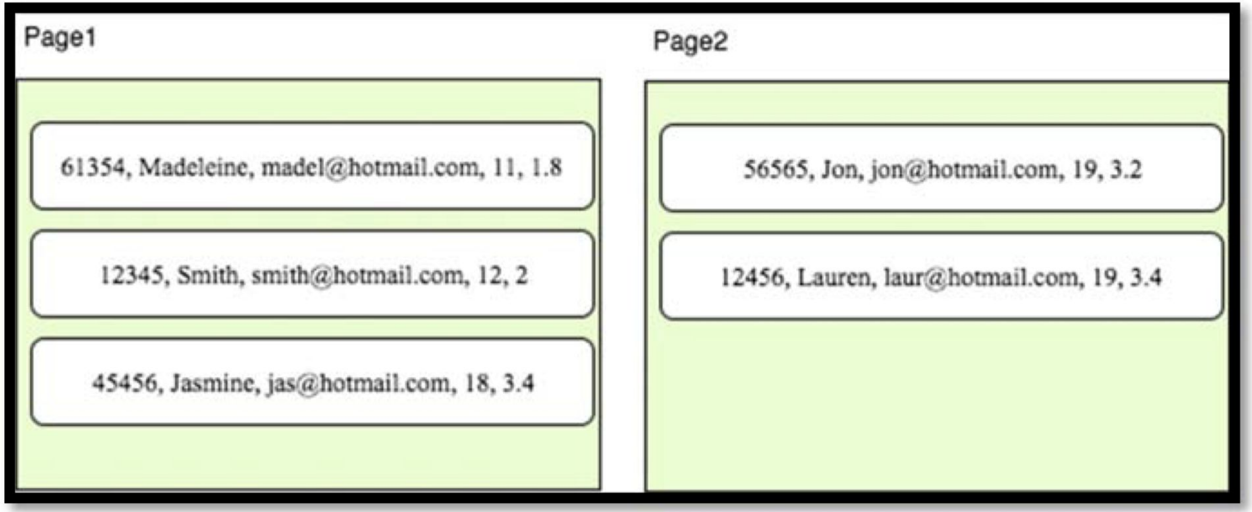
Q2. Data entries of an index

Consider the following instance of the relation
Student(SID, Name, Email, Age, GPA), where tuples are sorted by age:

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

Exercises

Let's assume that the order of tuples is the same when stored on disk. The first record is on page 1 and each page can contain only 3 records. The arrangement of the records is shown below:



Exercises

Recall:

- Data entries of the index are of the form (*k*, *rid*) where *k* = search key value and *rid* = (page #, record # on page)
- Assume it's a B+ tree index, so data entries of the index are in the same order as the search key

Exercises

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

- In this case, the actual records in the file are sorted on age
- We want to build an index on age
- Data entries of the index are in the same order as age, the search key
- So the data entries of the index will have the same order as the data records
    - Clustered index!

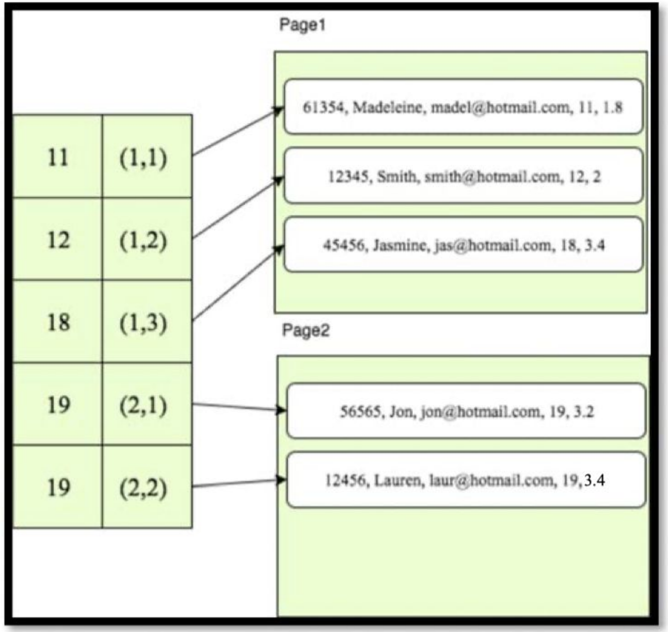
Exercises

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

k	*rid*
11	(1, 1)
12	(1, 2)
18	(1, 3)
19	(2, 1)
19	(2, 2)

Exercises

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



Exercises

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

- In this case, the actual records in the file are sorted on age
- We want to build an index on GPA
- Data entries of the index are in the same order as GPA, the search key
- So the data entries of the index will *not* have the same order as the data records
    - Unclustered index!

Exercises

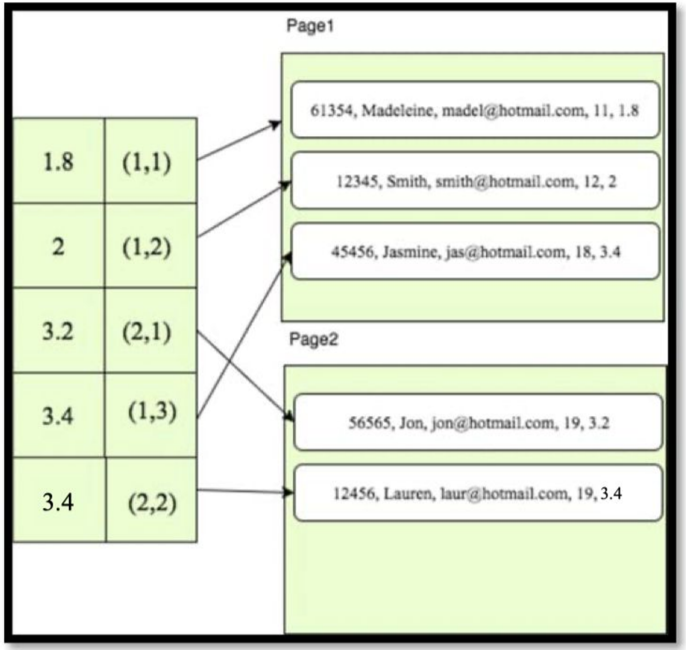
Show what the *data entries* of the index will look like for an index on GPA:

k	*rid*
1.8	(1, 1)
2	(1, 2)
3.2	(2, 1)
3.4	(1, 3)
3.4	(2, 2)

Note: if there's a tie, keep *rid* in order

Exercises

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



Exercises

Q3. Consider the following relations:

Employee ($\underline{\text{EmployeeID}}$, EmployeeName, Salary, Age, $\overset{\text{FK}}{\text{DepartmentID}}$)

Department ($\underline{\text{DepartmentID}}$, DepartmentBudget, DepartmentFloor, $\overset{\text{FK}}{\text{ManagerID}}$)

- Salary ranges from 10,000–100,000
- Age varies from 20–80 years
- Each department has 5 employees on average
- There are 10 floors
- Budgets of departments vary from 10,000–1 million

Exercises

Given the following two queries frequently used by the business, which index would you prefer to speed up the query? (7 mins)

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

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

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

Exercises

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

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

- Clustered, so records will be ordered on DepartmentFloor and DepartmentBudget
- Query will be executed such that you first traverse B+ tree and access records where DepartmentFloor = 10
- Then you continue to find records where DepartmentBudget < 15000… which is quick since DepartmentBudget is already ordered!

Exercises

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

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

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

- Records will be ordered on EmployeeName, Age and Salary
- Can perform an index-only scan!!!
- The data entries in the index already contain all the attributes we need
- We only need to access the index
- No need to access the data pages containing the actual records!

Week 6 Lab

- Canvas → Modules → Week 6 → Lab → L06 SQL 2 (PDF)
- Objectives:
  - Learn unary and outer joins in SQL
  - Self-test your SQL skills
- Breakout rooms, “ask for help” button if you need help or have any questions