Database Processing
CS 451 / 551

Lecture 10:
Sorting and Transactions

Suyash Gupta
Assistant Professor
Distopia Labs and ONRG
Dept. of Computer Science
(E) suyash@uoregon.edu
(W) gupta-suyash.github.io

Assignment 2 is Out!
Deadline: Nov 13, 2025 at 11:59pm

Quiz 2: Nov 6, 2025 (in class)

Query Processing

- Last Lecture we looked at Query Processing
  - How to measure the cost of a select operation?
  - The cost of select operation helps us to decide what type of indexes to use, what operators and attributes to access.

Query Processing

- Last Lecture we looked at Query Processing
  - How to measure the cost of a select operation?
  - The cost of select operation helps us to decide what type of indexes to use, what operators and attributes to access.

- Next, we look at Sorting.

Why is Sorting Interesting to Us?

Why is Sorting Interesting to Us?

- SQL queries can require the output be sorted.

- Efficient query processing:
  - Operations like Joins and searching can be implemented.

Ways to Sort a set of Keys

- **Is it possible to sort a set of keys/records without running a sorting algorithm?**

Ways to Sort a set of Keys

- **Is it possible to sort a set of keys/records without running a sorting algorithm?**

- Yes, what if we have access to a B^+-tree.
  - All the leaves in a B^+-tree are sorted!

Ways to Sort a set of Keys

- **Is it possible to sort a set of keys/records without running a sorting algorithm?**

- Yes, what if we have access to a B^+-tree.
  - All the leaves in a B^+-tree are sorted!

  - **Can we do something more?**

Top-N Heap Sort

- Say a query contains an **ORDER BY** clause with a **LIMIT.**
  - The DBMS only needs to scan the data once to find the required number of elements.

  - Specifically, a query asks you to output only first **N** elements.

Top-N Heap Sort

- Say a query contains an **ORDER BY** clause with a **LIMIT.**
  - The DBMS only needs to scan the data once to find the required number of elements.
  - Specifically, a query asks you to output only first **N** elements.
  - This is also an ideal candidate for **Heap Sort**!

Top-N Heap Sort

- Say a query contains an **ORDER BY** clause with a **LIMIT.**
  - The DBMS only needs to scan the data once to find the required number of elements.
  - Specifically, a query asks you to output only first **N** elements.
  - This is also an ideal candidate for **Heap Sort**!
  - We are just going to scan data once and maintain an in-memory sorted **priority queue**.

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid
  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

23

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

23 34

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

12 23 34

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

12 23 34 67

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

8 12 23 34

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

6	8	12	23

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

6	8	12	23

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

6 8 12 23

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

6	8	12	23

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid
  asc fetch first 4 **rows with ties**

Original Data

23	34	12	67	8	6	76	98	78	19	83	19

Sorted Data

6	8	12	19

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

6 8 12 19

Top-N Heap Sort

- Say, our query is:

  select * from enrolled **order by** sid

  asc fetch first 4 **rows with ties**

Original Data

23 34 12 67 8 6 76 98 78 19 83 19

Sorted Data

6 8 12 19 19

Tracks all the duplicates!

Ways to Sort a set of Key

- **But, having a B^+-tree and Top-N Heap Sort sufficient for all the sorting queries?**
- **What are we still missing?**

Ways to Sort a set of Key

- **But, is having just a B^+-tree sufficient to satisfy all the sorting queries?**
- **What are we still missing?**
- B^+-tree is just an index.
- Your data needs to be physically sorted too!
- Remember, the benefits of sequential data access only comes when data is stored in a sorted manner.

Ways to Sort a set of Key

- Imagine you have stored the data in your disk in unsorted manner but you are now trying to fetch it in sorted manner?
- Pretty bad performance as too many disk accesses!

Ways to Sort a set of Key

- So, depending on where your data resides (in-memory or disk, you select a sorting algorithm.

- **Why this difference?**

Ways to Sort a set of Key

- So, depending on where your data resides (in-memory or disk, you select a sorting algorithm.

- **Why this difference?**

- Because, an in-memory sorting algorithm does not need to worry about expensive data swapping operations!

Ways to Sort a set of Key

- So, depending on where your data resides (in-memory or disk, you select a sorting algorithm.

- **Why this difference?**

- Because, an in-memory sorting algorithm does not need to worry about expensive data swapping operations!

- In-memory swapping algorithms? → Quick Sort

- Our focus → Disk sorting algorithms.

Ways to Sort a set of Key

- So, depending on where your data resides (in-memory or disk, you select a sorting algorithm.

- **Why this difference?**

- Because, an in-memory sorting algorithm does not need to worry about expensive data swapping operations!

- In-memory swapping algorithms? → Quick Sort

- Our focus → Disk sorting algorithms.

External Merge-Sort

- Sorting of relations that do not fit in memory is called external sorting.
- A divide and conquer algorithm!
- Split data into parts (also called as runs).
- Sort each run individually.
- Merge the runs!

External Merge-Sort

- **What are the challenges for External Merge-Sort?**

External Merge-Sort

- **What are the challenges for External Merge-Sort?**
- Number of runs.
- Size of each run.
- Size of memory.
- All these factors work in conjunction to determine how fast can we perform external merge-sort.

N-Way External Merge-Sort

- We are going to run an external merge sort where:
  - **N** total number of runs
  - We will assume each run consists of **M blocks**.
  - We will assume the memory can store **N+1 blocks**.
  - We will fetch **one block from each run** at a time.
  - Remember → Often you would decide on the right number of runs based on how many threads you have, and how many blocks you can store.

2-Way External Merge-Sort → N=2

23 34 12 67 8 6 76 98 78 19 83 19

Assume that we can have only 2 runs at a time and total of 3 blocks in memory.

2-Way External Merge-Sort → N=2

23 34 12 67 8 6 76 98 78 19 83 19

Run 1 **Run 2**

Initially, Each Run is of size 1.

Each run is implicitly sorted!

2-Way External Merge-Sort → N=2

23 34 12 67 8 6 76 98 78 19 83 19

Run 1 Run 2

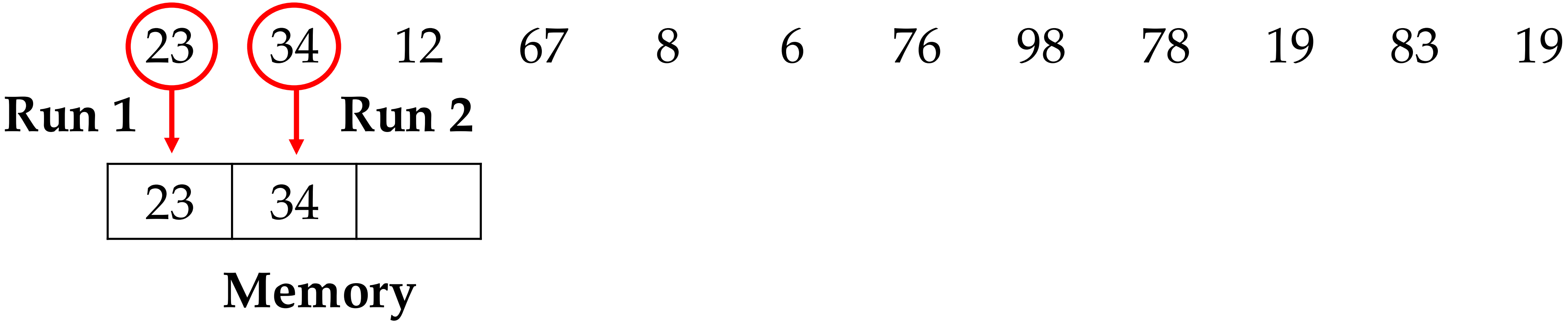
23	34	

Memory

Next, we merge two runs at a time.

We can have only 3 blocks in memory

2-Way External Merge-Sort → N=2



Use the extra block to sort.

Write back to memory

2-Way External Merge-Sort → N=2

23	34	12	67	8	6	76	98	78	19	83	19

12	67	

Memory

Use the extra block to sort.

Write back to memory

2-Way External Merge-Sort → N=2

23 34 12 67 8 6 76 98 78 19 83 19

8	6	

Memory

Use the extra block to sort.

Write back to memory

2-Way External Merge-Sort → N=2

23	34	12	67	6	6	76	98	78	19	83	19

8	6	8

Memory

Use the extra block to sort.

Write back to memory

2-Way External Merge-Sort → N=2

23	34	12	67	6	8	76	98	78	19	83	19

8	6	8

Memory

Use the extra block to sort.

Write back to memory

2-Way External Merge-Sort → N=2

23 34 12 67 6 8 76 98 19 78 19 83

Memory

Lets continue this till the end

2-Way External Merge-Sort → N=2

23 34 12 67 6 8 76 98 19 78 19 83

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

23 34 12 67 6 8 76 98 19 78 19 83

23	12	

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	34	12	67	6	8	76	98	19	78	19	83

23	67	

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	34	12	67	6	8	76	98	19	78	19	83

23	67	34

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	23	12	67	6	8	76	98	19	78	19	83

	67	34

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	23	34	67	6	8	76	98	19	78	19	83

	67	34

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	23	34	67	6	8	76	98	19	78	19	83

6	76	

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12 23 34 67 6 8 76 98 19 78 19 83

19	19	

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12 23 34 67 6 8 76 98 19 19 78 83

19	19	

Memory

Now, the size of each run is 2 blocks

2-Way External Merge-Sort → N=2

12	23	34	67	6	8	76	98	19	19	78	83

12	6	

Memory

Now, the size of each run is 4 blocks

2-Way External Merge-Sort → N=2

6 23 34 67 6 8 76 98 19 19 78 83

12	8	

Memory

Now, the size of each run is 4 blocks

2-Way External Merge-Sort → N=2

6	23	34	67	6	8	76	98	19	19	78	83

12	8	23

Memory

Now, the size of each run is 4 blocks

2-Way External Merge-Sort → N=2

6	8	34	67	6	8	76	98	19	19	78	83

12		23

Memory

Now, the size of each run is 4 blocks

2-Way External Merge-Sort → N=2

6 8 12 23 34 67 76 98 19 19 78 83

Memory

Now, the size of each run is 4 blocks

2-Way External Merge-Sort → N=2

6 8 12 23 34 67 76 98 19 19 78 83

Memory

Now, the size of one run is 8 blocks while other is 4 blocks

Transactions

Transactions

- Transactions are ubiquitous!

- Examples: Banking, Online shopping, Trading, Social media, and so on.

How to define a Transaction?

How to define a Transaction?

- Transaction is a collection of operations.

- For example:
  - Moving money from one checkings account to savings account.
  - Buying a product from Amazon.

How to define a Transaction?

- Transaction is a unit of program that reads and/or writes one or more data items.

- A common way to write a transaction in popular DBMS is by placing the body of the transaction between, “**begin transaction**” and “**end transaction**”.

How to define a Transaction?

- Transaction is a unit of program that reads and/or writes one or more data items.
- A common way to write a transaction in popular DBMS is by placing the body of the transaction between, “**begin transaction**” and “**end transaction**”.
- Also, the reason why transaction is termed as an indivisible unit.
  - It either executes in its entirety or nothing at all.

ACID Properties for a Transaction

?

ACID Properties for a Transaction

- Each database should provide the following four properties for transactions :

ACID Properties for a Transaction

- Each database should provide the following four properties for transactions :

- **Atomicity:** Either all operations of the transaction are reflected properly in the database, or none are.

ACID Properties for a Transaction

- Each database should provide the following four properties for transactions:

- **Atomicity:** Either all operations of the transaction are reflected properly in the database, or none are.

- **Consistency:** Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

ACID Properties for a Transaction

- Each database should provide the following four properties for transactions:

- **Atomicity:** Either all operations of the transaction are reflected properly in the database, or none are.

- **Consistency:** Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
    - Not referring to database consistency constraints.

- **Isolation**: For every pair of concurrent (executing at the same time) transactions Ti and Tj , either Ti finished execution before Tj started, or Ti started execution after Tj finished.
    - Transactions are unaware of other transactions executing

ACID Properties for a Transaction

- **Atomicity:** Either all operations of the transaction are reflected properly in the database, or none are.

- **Consistency:** Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
  - Not referring to database consistency constraints.

- **Isolation**: For every pair of concurrent (executing at the same time) transactions Ti and Tj , either Ti finished execution before Tj started, or Ti started execution after Tj finished.
  - Transactions are unaware of other transactions executing

- **Durability:** Once a transaction completes successfully, any changes it made to the database should persist, even if there are system failures.

ACID Properties for a Transaction

- **When do ACID properties come into play?**

ACID Properties for a Transaction

- **When do ACID properties come into play?**
  - **Concurrency!**

- In a concurrent system or database, two or more transactions may attempt to fetch the same data.

- **But, why is this an issue?**

ACID Properties for a Transaction

- **When do ACID properties come into play?**
  - **Concurrency!**

- In a concurrent system or database, two or more transactions may attempt to fetch the same data.

- **But, why is this an issue?**
  - Concurrency if not handled well can lead to ACID violations.
  - For instance. → Race conditions!

Two Concurrent Transactions

T1:

read(**A**);
A = **A** − 50;
write(**A**);
read(**B**);
B = **B** + 50;
write(**B**).

T2:

read(**A**);
temp = **A** * 0.1;
A = **A** − temp;
write(**A**);
read(**B**);
B = **B** + temp;
write(**B**)

Notice that they are accessing the same variables A and B.

Conflicting Transactions

Two transactions T1 and T2 if they **concurrently access the same variable** and at least one of that access is a **write** operation, then they **conflict**!

Isolation

- Users submit transactions.
- Each transaction should execute as if it were running by itself.
- But running one transactions at a time will give poor performance.
- With the prevalence of multi-core architecture, DBMS should take advantage of the multiple cores.
- Concurrency permits interleaving the transaction operations.
    - Interleaving transactions also permits running one transaction when another is waiting for some resource (I/O, user input, or fetching data from disk).
    - Need a mechanism to interleave transactions but make it appear as if they ran one-at-a-time.

Concurrent Transactions

- Assume that these two transactions are undergoing concurrent execution.
- **What are the possible interleaving of these two transactions**?

T1:	T2:
read(**A**);	read(**A**);
A = **A** − 50;	**A** = **A** * 0.95;
write(**A**);	write(**A**);
read(**B**);	read(**B**);
B = **B** + 50;	**B** = **B** * 1.05 ;
write(**B**).	write(**B**)

Isolation Support: Concurrency Control

- A **concurrency control protocol** lays down the mechanism for the DBMS to decide a legal/valid schedule of transactions.

- What are the **types** of concurrency control protocols?

Isolation Support: Concurrency Control

- A concurrency control protocol lays down the mechanism for the DBMS to decide legal/valid schedule of transactions.

- What are the types of concurrency control protocols?

- **Pessimistic:** Prevent problems from arising in the first place.

Isolation Support: Concurrency Control

- A concurrency control protocol lays down the mechanism for the DBMS to decide legal/valid schedule of transactions.
- What are the types of concurrency control protocols?
- **Pessimistic:** Prevent problems from arising in the first place.
- **Optimistic:** Assume that conflicts are rare; deal with them after they occur.

Isolation Support: Concurrency Control

- A concurrency control protocol lays down the mechanism for the DBMS to decide legal/valid schedule of transactions.
- What are the types of concurrency control protocols?
- **Pessimistic:** Prevent problems from arising in the first place.
- **Optimistic:** Assume that conflicts are rare; deal with them after they occur.
- **But, how does a concurrency control protocol determine a valid schedule?**

Serializable Schedules

- **Serial Schedule** → A schedule that does not interleave the operations of different transactions.

- **Equivalent Schedule** → For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.

- **Serializable Schedule?**

Serializable Schedules

- **Serial Schedule** → A schedule that does not interleave the operations of different transactions.

- **Equivalent Schedule** → For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.

- **Serializable Schedule** → A schedule that is equivalent to some serial execution of the transactions (serial schedule).

Serializable Schedules

- **Serial Schedule →** A schedule that does not interleave the operations of different transactions.

- **Equivalent Schedule →** For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.

- **Serializable Schedule →** A schedule that is equivalent to some serial execution of the transactions (serial schedule).

- If each transaction preserves consistency, then the corresponding serializable schedule preserves consistency!

Isolation Levels vs. Consistency Levels

Isolation Levels	Consistency Levels

Isolation Levels vs. Consistency Levels

Isolation Levels	Consistency Levels
• **Correspond** to the **I in ACID**. • **Database isolation** is the ability of a database to allow a transaction to execute as if there are no other concurrently running transactions. • Greater the guaranteed isolation among the transactions, lesser the system performance. • **Isolation levels** trade off isolation guarantees for improved performance.	

Isolation Levels vs. Consistency Levels

Isolation Levels	Consistency Levels
• **Correspond** to the **I in ACID**. • **Database isolation** is the ability of a database to allow a transaction to execute as if there are no other concurrently running transactions. • Greater the guaranteed isolation among the transactions, lesser the system performance. • **Isolation levels** trade off isolation guarantees for improved performance.	• **Do not correspond to C in ACID**. • Unlike the C in ACID, the **database consistency** refers to the rules that make a **concurrent**, **distributed system** appear as a **single-threaded, centralized system**. • **Reads** at a particular point in time **must reflect the most recently completed write** (in real-time) of that data item, no matter which server processed that write. • **Consistency levels** trade off read results for improved performance.

Isolation Levels vs. Consistency Levels

- More simply said:
  - ➢Whenever you talk about transaction isolation, you will be talking about isolation levels.
  - ➢Whenever you talk about individual operations like read/write, you will talk about consistency levels.