TRANSACTIONS

-> logical units of work in a database
-> sequence of operations: read, write, commit, abort
-> ensure reliability and consistency of database operations

ACID Properties

- **atomicity:**
  - a transaction is atomic and treated as indivisible unit
  - either all operations succeed or none do
  - if a failure occurs, changes are rolled back to maintain consistency

- **consistency:**
  - transactions preserve the database's predefined rules and constraints
  - do not violate integrity constraints
  - violating transactions are rolled back to maintain data integrity

- **isolation:**
  - transactions are isolated from each other
  - intermediate states of transactions are not visible to others until committed
  - this prevents interference and maintains consistent results

- **durability:**
  - committed transactions' changes are permanent
  - changes survive subsequent failures or system crashes
  - data persistence is ensured by storing changes reliably

Scheduling Transactions

→ ordering of operations from different transactions

SERIAL SCHEDULE

- transactions executed one after another
- no overlap in execution
- maintains integrity and isolation
- ensures data consistency
- no parallelism

NON-SERIAL SCHEDULE

- allows concurrent execution of transactions
- enables parallelism
- requires careful design for correctness

SERIALIZABILITY

- property of a schedule
- behavior equivalent to a serial schedule

CONCURENCY CONTROL

→ manages concurrent access to the database
→ ensures correctness, consistency and serializability
→ prevents conflicts and maintains data integrity

CONFLICT SERIALIZABILITY

→ condition for serializability
→ determines equivalence to a serial schedule
→ order of conflicting operations preserved
→ final result is equivalent to some serial execution

PRECEDENCE GRAPH

→ graphical representation of a schedule
→ shows ordering of conflicting operations
→ nodes represent transactions
→ directed edges represent operation precendence
→ no cycles = serializable

$R_1(A) \rightarrow W_2(A)$
$W_1(A) \rightarrow R_2(A)$
$W_1(A) \rightarrow W_2(A$

VIEW SERIALIZABILITY

→ condition for serializability
→ determines equivalence to a serial schedule
→ considers the visibility of data items
→ final results is equivalent to some serial execution

view equivalence = two schedules are view equivalent if they produce the same results for all transactions

RECOVERABLE SCHEDULES

→ ensures that, if a transaction T1 reads from transaction T2, T2 must commit before T1
→ prevents dirty reads, only committed data is read

dirty read = occurs when a transaction reads data that has been modified but not yet committed

AVOIDING CASCADE ABORTS

→ cascading aborts occur when an initial aborted transaction causes other transactions depending on it to also abort
→ techiques used to avoid cascading aborts include strict two-phases locking and timestamp ordering

LOCK-BASED CONCURENCY CONTROL

→ manages concurrent access to shared resources using locks
→ ensures data consistency, isolation and integrity

Types of locks

SHARED LOCK

- (s-lock) allows multiple transactions to read a resource concurrently; read-only access

EXCLUSIVE LOCK

- (X-lock) grants exclusive access to resource
- write access, preventing other transactions from reading or writing

Locking protocals

STRICT TWO-PHASE LOCKING (2PL)

- aquires locks in two phases:
  - growing phase: aquires locks on data items as needed during the transaction's execution
  - shrinking phase: releases locks only after the transaction has completed
- ensures conflict serializability
- prevents unserializable schedules and deadlocks
- holds all locks until the end of a transaction
- prevents cascading aborts and ensures consistency

TWO-PHASE LOCKING (2PL)

- aquires locks in two phases:
  - growing phase: aquires locks on data items as needed during the transaction's execution
  - shrinking phase: releases locks after the last lock request has been made
- ensures conflict serializability
- prevents unserializable schedules and deadlocks
- allows the possibility of cascading aborts
- locks are released during the transaction's execution, but new locks are not acquired

Deadlocks

→ circular waiting between transactions
→ can result in a permanent blocking of transactions

PREVENTION

- strategies to avoid deadlocks
- techniques include deadlock detection, timeouts, and resource ordering

DETECTION

- techniques to identify deadlocks in a sytem
- algorithms include resource allocation and wait-for graph

Joins

Simple Nested Loops Join

$$M + p_E * M * N$$

- cost of scanning E
- cost of scanning S
- → S is scanned $p_E * M$ times
  - * E - M pages, p_E records / page *
  - * S - N pages, p_S records / page *

Page-Oriented Nested Loops Join

$$M + M * N$$

- cost of scanning E
- cost of scanning S
- → S is scanned M times
  - * E - M pages, p_E records / page *
  - * S - N pages, p_S records / page *

Block Nested Loops Join

$$SOT + NBOT * SIT$$

- scan of outer table
- number of blocks in outer table
- scan of inner table

$$NOB = \left[\frac{NPOT}{SB} \right]$$

- number of outer table
- size of block
- number of pages in outer table

Index Nested Loops Join

$$M + ((M * P_E) * CFCRS)$$

→ cost of finding corresponding records in S

* E - M pages, P_E records / page *
* S - N pages, P_S records / page *

SORTING

Simple Two-Way Merge Sort

→ pass 0: N runs

→ number of passes:

- $\lceil \log_2 N \rceil + 1$
  - N → number of pages in the file to be sorted

→ total cost:

- $2 * \text{number of pages} * \text{number of passes}$
- $2 * N * (\lceil \log_2 N \rceil + 1)$ I/Os

External Merge Sort

→ pass 0:

- $\lceil \frac{N}{B} \rceil$ runs
  - N → number of pages in the file
  - B → available buffer pages

→ number of passes:

- $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$
  - N → number of pages in the input file
  - B → available pages in buffer

→ total cost:

- $2 * N * \left(\lceil \log_{B-1} \lceil \frac{N}{B} \rceil \rceil + 1 \right)$ I/Os

Sort-Merge Join

→ sorting E cost: $O(M \log M)$ * E - M pages

→ sorting S cost: $O(N \log N)$ * S - N pages

→ cost of merging: $M + N$ I/Os

→ worst-case scenario: $O(M * N)$ I/Os

Hash Join

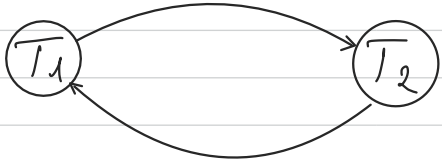
→ partitioning cost: $2 * (M + N)$ I/Os
→ probing cost: $M + N$ I/Os
→ total cost: $3 * (M + N)$ I/Os
  * E - M pages, P_E records / page *
  * S - N pages, P_S records / page *

1. Consider the schedule below:

T1	T2
read(A)	
A = A + 50	
write(A)	
	read(A)
	A = A * 4
	write(A)
	read(B)
	B = B * 4
	write(B)
read(B)	
B = B + 50	
write(B)	

time

The following integrity constraint must hold: $A = B$. Before the execution of T1 and T2, A = 100 and B = 100. Is the schedule serializable? Justify your answer.



There is a cycle between T1 and T2 which means the schedule is not serializable.

T1: read A which is 100
T1: 100 + 50 = 150
T1: write A with new value 150

T2: read A which is 150
T2: 150 * 4 = 600
T2: write A with new value 600
T2: read B which is 100
T2: 100 * 4 = 400
T2: write B with new value 400

T1: read B which is 400
T1: 400 + 50 = 450
T1: write B with new value 450

The integrity constraint A = B might be violated.

2. Consider the following schedule over transactions T1, T2, T3, T4 (all transactions commit): (1p)

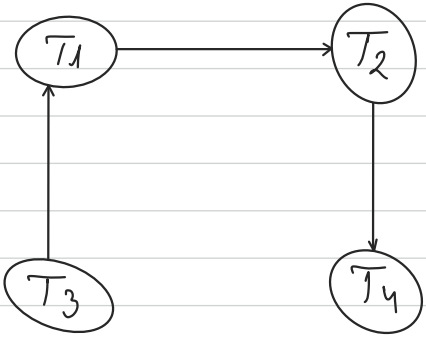
T1	T2	T3	T4
W(A)			R(C)
	R(B)		
		W(D)	
	R(A)		
R(D)			
			W(B)
R(C)			

time

$R_1(A) \rightarrow W_2(A)$
$W_1(A) \rightarrow R_2(A)$
$W_1(A) \rightarrow W_2(A)$

a. Draw the precedence graph.
b. Is the schedule conflict serializable? Justify your answer.

a.



b. Because the precedence graph has no cycles, that means the schedule is conflict serializable.
Also, $(W(T1, A), R(T2, A))$ belongs to the conflict relation of S.

4. Encode the data *de gustibus non disputandum* using the secret encryption key *metallica* and the table of codes below. Write the last 5 characters in the result.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	-
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

data: de gustibus non disputandum
key: metallica

m	e	t	a	l	l	i	c	a
13	05	20	01	12	12	09	03	01
d	e		g	u	s	t	i	b
04	05	00	07	21	19	20	09	02
u	s		n	o	n		d	i
21	19	00	14	15	14	00	04	09
s	p	u	t	a	n	d	u	m
19	16	21	20	01	14	04	21	13

04	05	00	07	21	19	20	09	02
13	05	20	01	12	12	09	03	01
17	10	20	08	06	04	02	12	03
21	19	00	14	15	14	00	04	09
13	05	20	01	12	12	09	03	01
07	24	20	15	00	26	09	07	10
19	16	21	20	01	14	04	21	13
13	05	20	01	12	12	09	03	01
05	21	14	21	13	26	13	24	14

=> the final word is: gjthfdblcgxtozigjeunuumzmxm

4. Let R and S be 2 relations. R has 10.000 records; a page can hold 10 R records. S has 2.000 records; a page can hold 10 S records. (1.5p)
a. 52 buffer pages are available. Compute the cost of $R \otimes_{R.a=S.b} S$ using *page-oriented nested loops join* and *block nested loops join*; S is the outer relation.
b. Compute the cost of sorting R using *external merge sort* with 200 buffer pages.
c. R is stored at Madrid, S is stored at Athens. Compute the cost of $R \otimes_{R.a=S.b} S$ using *simple nested loops join (tuple-oriented)* in Athens, without caching; S is the outer relation.

a. SELECT *
FROM R INNER JOIN S ON R.a=S.b

R - 10000 records; a page can hold 10R records => 1000 pages
S - 2000 records; a page can hold 10S records => 200 pages

- page oriented nested loops:
  200 + 200 * 1000 = 200200 I/Os

- block nested loops join:
  block size: 50 => $\left\lceil \frac{200}{50} \right\rceil = 4$ blocks
  200 + 4 * 1000 = 4200 I/Os

b. $2 * 1000 * 2 = 4000$ I/Os
$2 * N * \left(\left\lceil \log_{B-1} [N/B] \right\rceil + 1 \right)$ I/Os

C. SELECT *
FROM R INNER JOIN S ON R.a=S.b

- t_d time to R/W a page from / to disk
- t_s time to ship a page
  - $200 t_d + 2000 * 1000 (t_d + t_s) = 200 t_d + 2\,000\,000 (t_d + t_s)$