Database Management Systems

Lecture 13 Problems - I

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. 52 buffer pages are available. Compute the cost of:
SELECT *
FROM R INNER JOIN S ON R.a = S.b
using *page-oriented nested loops join* and *block nested loops join*; S is the outer relation.
R – 10,000 records; a page can hold 10 R records => 1,000 pages
S – 2,000 records; a page can hold 10 S records => 200 pages

• page-oriented nested loops join
  - **• 200 + 200 * 1000 = 200,200 I/Os**

• block nested loops join
  - **block size: 50 => [200/50]=4 S blocks**
  - **200 + 4 * 1000 = 4,200 I/Os**

Solution:
S – outer table; R- inner join table
R – 10,000 records; a page can hold 10 R records => 1,000 pages - 10000 records / 10 records per page = 1000 pages =N
S – 2,000 records; a page can hold 10 S records => 200 pages – 2000 records / 10 records per page = 200 pages = **M**

M – cost of scanning table S
N – cost of scanning table R

• page-oriented nested loops join
  - • 200 + 200 * 1000 = 200,200 I/Os

Cost: M+M*N = 200 + 200*1000 = 200200 I/Os

Lecture 6: Page-oriented nested loops join

Page-Oriented Nested Loops Join

```
foreach page pe ∈ E do
    foreach page ps ∈ S do
        if e_i == s_j then add <e, s> to the result
```

- cost
  - **M + M*N** = 1000 + 1000*500 I/Os = 501.000 I/Os
    - M I/Os – cost of scanning E; N I/Os – cost of scanning S
    - S is scanned M times
    - significantly lower than the cost of Simple Nested Loops Join (improvement - factor of p_E)
  - if the smaller table (*S*) is chosen as outer table:

  => cost = 500 + 500 * 1000 I/Os = 500.500 I/Os

* E - M pages, p_E records / page *   * 1000 pages * * 100 records / page*
* S - N pages, p_S records / page *   * 500 pages * * 80 records / page * Sabina S. CS

• block nested loops join
  - block size: 50 => [200/50]=4 S blocks
  - 200 + 4 * 1000 = 4,200 I/Os

52 buffer pages are available; we suppose that the buffer has 50 pages available for relation S (2 pages remain for relation R) (due to input and output pages need it).
A block can hold 50 pages (= block size).

Follows that: **number of block** = [number of pages in outer table / size of a block] = [200 / 50] = 4 S blocks
Cost: scan of outer table + number of blocks in outer table * scan of inner table
= 200 + 4*1000 = 4200 I/Os (M + number of blocks * N)

Lecture 6: Block Nested Loops Join

Block Nested Loops Join
- cost
  - scan of outer table + number of blocks in outer table * scan of inner table
  - number of outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Exams (E), a block can hold 100 pages
    - scan cost for E: 1000 I/Os
    - number of blocks: $\left\lceil \frac{1000}{100} \right\rceil$ = 10
    - foreach block in E, scan Students (S): 10*500 I/Os

  => total cost = 1000 + 10 * 500 = **6000 I/Os**

* E - M pages, p_E records / page *   * 1000 pages * * 100 records / page*
* S - N pages, p_S records / page *   * 500 pages * * 80 records / page *

Sabina S. CS

Block Nested Loops Join
- cost
  - scan of outer table + number of blocks in outer table * scan of inner table
  - number of outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Exams (E)
    - suppose the buffer has 90 pages available for E, i.e., block of 90 pages

    => number of blocks: $\left\lceil \frac{1000}{90} \right\rceil$ = 12

    => S is scanned 12 times
    - scan cost for E: 1000 I/Os
    - foreach block in E, scan Students (S): 12*500 I/Os

  => total cost = 1000 + 12 * 500 = **7000 I/Os**

* E - M pages, p_E records / page *   * 1000 pages * * 100 records / page*
* S - N pages, p_S records / page *   * 500 pages * * 80 records / page *

Sabina S. CS

Block Nested Loops Join
- cost
  - scan of outer table + number of blocks in outer table * scan of inner table
  - number of outer blocks = $\left\lceil \frac{\text{number of pages in outer table}}{\text{size of block}} \right\rceil$
  - outer table: Students (S), block of 100 pages
    - scan cost for S: 500 I/Os
    - number of blocks: $\left\lceil \frac{500}{100} \right\rceil$ = 5
    - for each block in S, scan E: 5 * 1000 I/Os

  => total cost = 500 + 5 * 1000 = **5500 I/Os**

* E - M pages, p_E records / page *   * 1000 pages * * 100 records / page*
* S - N pages, p_S records / page *   * 500 pages * * 80 records / page *

2. Compute the cost of sorting R using *external merge sort* with 200 buffer pages.
- **2*1000*2=4,000I/Os**
- $\mathbf{2 * N * \left(\left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil + 1\right)}$ **I/Os**

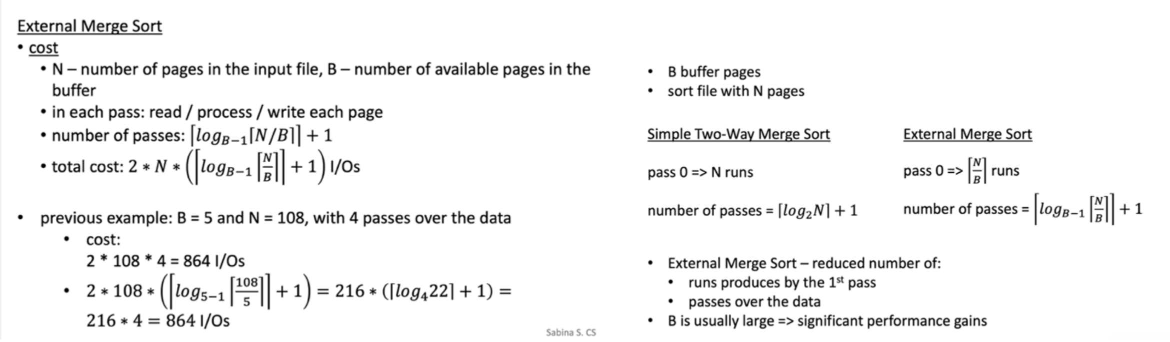
Solution:
- 2*1000*2=4,000I/Os
- $2 * N * \left(\left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil + 1\right)$ I/Os

Total Cost: 2 * N * $\left(\left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil + 1 \right\rceil\right)$ I/Os, where $\left(\left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil + 1 \right\rceil\right)$ is the number of passes.
B=52 buffer pages available, and so:

$\left(\left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil + 1 \right\rceil\right)$= $\left(\left\lceil log_{52-1} \left\lceil \frac{1000}{52} \right\rceil + 1 \right\rceil\right)$=([$log_{51}20 + 1$])=[0,… + 1] = 2

Total cost: 2 * 1000 * 2 = 4000 I/Os

Lecture 7: External Merge sort



3. R is stored at București, S is stored at Cluj-Napoca. Compute the cost of:
SELECT *
FROM R INNER JOIN S ON R.a = S.b
using *simple nested loops join (tuple-oriented)* in Cluj-Napoca, without caching; S is the outer relation.

- **t_d time to R/W a page from / to disk**
- **t_s time to ship a page**

• $200t_d + 2000 * 1000 (t_d + t_s) = 200t_d + 2{,}000{,}000 (t_d + t_s) = 2{,}000{,}200t_d + 2{,}000{,}000t_s$

We use the cost formula for Simple nested loops + the adapt it to distributed databases.

Simple Nested Loops Cost: M + p_S*M*N, where p_s is the number of records per page.

Extra, from Distributed Databases, we have:
- t_d – time to read / write a page to disk
- t_s – time to ship a page from a site to another (Cluj-Napoca to Bucuresti)

Cost: $M * t_d + p_S*M*N * (t_d + t_s) = 200\ t_d + 10 * 200 * 1000\ (t_d + t_s) =$

where $10 * 200 = p_S*M = 2000$ = the total number of S records

$= 200\ t_d + 2000 * 1000\ (t_d + t_s) = 200\ t_d + 2000000\ (t_d + t_s) = 2000200\ t_d + 2000000\ t_s$

Discussion: S – stored at Cluj-Napoca, R – stored at Bucuresti

- **Case 1:** S – outer table, cost computed in Cluj-Napoca:

Cost = $M * t_d + p_S*M*N * (t_d + t_s)$

- **Case 2:** S – outer table, cost computed in Bucuresti:

Cost = $M * (t_d + t_s) + p_S*M*(t_d + t_s)*N$

- **Case 3:** R – outer table, cost computed in Bucuresti:

Cost = $N * t_d + p_R*N*M * (t_d + t_s)$

- **Case 4:** R – outer table, cost computed in Cluj-Napoca:

Cost = $N * (t_d + t_s) + p_S*N*(t_d + t_s)*M$

Anyway, it depends on the algorithm… please, try to analyze all the time, the 2 aspects: cost of I/O (read/write) operation at the site + cost of shiping ☺

Lecture 6

Simple Nested Loops

Simple Nested Loops Join

```
foreach tuple e ∈ E do
    foreach tuple s ∈ S do
        if e_i == s_j then add <e, s> to the result
```

- for each record in the outer relation E, scan the entire inner relation S
- cost
  - **M + p_E* M * N** = 1000 + 100*1000*500 I/Os = 1000 + (5 * 10^7) I/Os
    - M I/Os – cost of scanning E
    - N I/Os – cost of scanning S
    - S is scanned p_E* M times (there are p_E* M records in the outer relation E)

* E - M pages, p_E records / page *   * 1000 pages *   * 100 records / page*
* S - N pages, p_S records / page *   * 500 pages *   * 80 records / page *

Sabina S.

Lecture 10: Distributed Databases

Distributed Query Processing

Researchers(RID: integer, Name: string, ImpactF: integer, Age: real)
AuthorContribution(RID: integer, PID: integer, Year: integer, Descr: string)

- Researchers
  - 1 tuple - 50 bytes
  - 1 page - 80 tuples
  - 500 pages
- AuthorContribution
  - 1 tuple - 40 bytes
  - 1 page - 100 tuples
  - 1000 pages

- estimate the cost of evaluation strategies:
  - number of I/O operations and number of pages shipped among sites, i.e., take into account communication costs
  - use t_d to denote the time to read / write a page from / to disk
  - use t_s to denote the time to ship a page from one site to another (e.g., from Skopje to Caracas)

Sabina S. CS

Distributed Query Processing
* join queries in a distributed DBMS
- Researchers R - New York, AuthorContribution A - Lisbon, R join A

* fetch as needed
- page-oriented nested loops in New York
  - Researchers - outer relation
  - for each page in Researchers, bring in all the AuthorContribution pages from Lisbon
  - cost
    - scan Researchers: 500t_d
    - scan AuthorContribution + ship all AuthorContribution pages (for each Researchers page): 1000(t_d + t_s)

=> total cost: 500t_d + 500,000(t_d + t_s)

Sabina S. CS

- query not submitted at New York
  => add the cost of shipping the result to the query site
- RID - key in Researchers
  => the result has 100,000 tuples (the number of tuples in AuthorContribution)
- the size of a tuple in the result
  - 40 + 50 = 90 bytes
- the number of result tuples / page
  - 4000 / 90 = 44

Sabina S. CS

- query not submitted at New York
- number of pages necessary to hold all the result tuples
  - 100,000/44 = 2273 pages
- the cost of shipping the result to another site (if necessary)
  - 2273 t_s
  - higher than the cost of shipping both Researchers and AuthorContribution to the site (1500 t_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**
Secret key**: metallica**

- Secret key has **9 characters**.

a. consider the table with the codes given.

	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

b. divide the message into blocks of length **L** = the number of characters into the secret key = 9.

d	e		g	u	s	t	i	b	u	s		n	o	n		d	i	s	p	u	t	a	n	d	u	m

- replace every character in the message and every character in the secret key with their associated values (from a.).

Data:

d	e		g	u	s	t	i	b	u	s		n	o	n		d	i	s	p	u	t	a	n	d	u	m
04	05	27	07	21	19	20	09	02	21	19	27	14	15	14	27	04	09	19	16	21	20	01	14	04	21	13

Secret key:

m	e	t	a	l	l	i	c	a
13	05	20	01	12	12	09	03	01

- put together all the data: (data + secret key) add every number that corresponds to a character in the block with the number of the corresponding character in the secret key.
- Let us consider **n=27**. If the obtained value is greater than **n** compute the remainder of the division by **n**.

In general, can be a good practice to consider **n=the greatest value from the table with the codes given +1.** (for example n=27+1=28)

d	e		g	u	s	t	i	b	u	s		n	o	n		d	i	s	p	u	t	a	n	d	u	m
04	05	27	07	21	19	20	09	02	21	19	27	14	15	14	27	04	09	19	16	21	20	01	14	04	21	13
13	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**	**13**	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**	**13**	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**
17	***10***	***20***	***08***	***06***	***04***	***02***	***12***	***03***	***07***	***24***	***20***	***15***	***27***	***26***	***09***	***07***	***10***	***05***	***21***	***14***	***21***	***13***	***26***	***13***	***24***	***14***

where:
d: 04+13 = 17
e: 05+05 = 10
-: 27+20 = 47 div 27 = 20
g: 07+01 = 08
u: 21+12 = 33 div 27 = 06
s: 19+12 = 31 div 27 = 04
t: 20+09 = 29 div 27 = 02
i: 09+03 = 12
b: 02+01 = 03
…

c. Replace the obtained numbers with their corresponding characters in the table of codes. We obtain a string that can be stored / transmitted.

d	e		g	u	s	t	i	b	u	s		n	o	n		d	i	s	p	u	t	a	n	d	u	m
04	05	27	07	21	19	20	09	02	21	19	27	14	15	14	27	04	09	19	16	21	20	01	14	04	21	13
13	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**	**13**	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**	**13**	**05**	**20**	**01**	**12**	**12**	**09**	**03**	**01**
17	***10***	***20***	***08***	***06***	***04***	***02***	***12***	***03***	***07***	***24***	***20***	***15***	***27***	***26***	***09***	***07***	***10***	***05***	***21***	***14***	***21***	***13***	***26***	***13***	***24***	***14***
q	j	t	g	f	d	b	l	c	g	x	t	o		z	i	g	j	e	u	n	u	m	z	m	x	n

The obtained string in the example is: ***qjtgfdblcgxto zigjeunumzmxn***

Lecture 5

* encryption algorithm - example

- use a secret encryption key

data:	disciplina baze de date
secret key:	student

a. create a table of codes

- every character is associated with a number, for instance:

	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

* encryption algorithm - example

b. divide the message into blocks of length *L*, where *L* is the number of characters in the key

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e

s	t	u	d	e	n	t

- replace every character in the message and every character in the key with their associated codes

	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

s	t	u	d	e	n	t
19	20	21	04	05	14	20

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e
04	09	19	03	09	16	12	09	14	01	00	02	01	26	05	00	04	05	00	04	01	20	05

Sabina S. CS

* encryption algorithm - example

b. add every number that corresponds to a character in a block with the number of the corresponding character in the key

- if the obtained value is greater than *n* (in the example, n = 27), compute the remainder of the division by *n*

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e
04	09	19	03	09	16	12	09	14	01	00	02	01	26	05	00	04	05	00	04	01	20	05
19	20	21	04	05	14	20	19	20	21	04	05	14	20	19	20	21	04	05	14	20	19	20
23	02	13	07	14	03	05	01	07	22	04	07	15	19	24	20	25	09	05	18	21	12	25

c. replace the obtained numbers with their corresponding characters in the table of codes

=> a string that can be stored / transmitted

- the obtained string in the example: *wbmgnceagvdgosxtyieruly*

II

1. T1 and T2 are 2 concurrent transactions, both active at time *t*. Choose the correct answer(s):

a. The following execution describes a *write read* conflict: At time *t*, T2 is reading a data object previously written by T1.

b. The following execution describes a *write read* conflict: At time *t*, T2 is writing a data object previously read by T1.

c. The following execution describes a *read write* conflict: At time *t*, T2 is reading a data object previously written by T1.

d. The following execution describes a *read write* conflict: At time *t*, T2 is writing a data object previously read by T1.

e. none of the above answers is correct.

Solution: ***AD*** - WR (a) + RW(d)

Lecture 1:

Interleaved Executions - Anomalies

- two transactions are only reading a data object => no conflict, order of execution not important
- two transactions are reading and / or writing completely separate data objects => no conflict, order of execution not important
- two transactions are operating on the same data object, and at least one of them performs a write operation => order of execution is important
  - WR conflict
    - T2 is reading a data object previously written by T1
  - RW conflict
    - T2 is writing a data object previously read by T1
  - WW conflict
    - T2 is writing a data object previously written by T1

Sabina S. CS

2. A schedule S:
a. is conflict serializable if and only if its precedence graph has exactly one cycle.
b. is conflict serializable if and only if its precedence graph is acyclic.
c. is conflict serializable if and only if its precedence graph has exactly two cycles.
d. is conflict serializable if and only if its precedence graph has exactly three cycles.
e. none of the above answers is correct.
Solution: B

Lecture 2: Conflict serializable

Conflict Serializability - Precedence Graph
- let *S* be a schedule in *Sch(C)*
- the *precedence graph* (*serializability graph*) of *S* contains:
  - one node for every committed transaction in *S*
  - an arc from T_i to T_j if an action in T_i precedes and conflicts with one of the actions in T_j
- Theorem:
  - a schedule $S \in Sch(C)$ is conflict serializable if and only if its precedence graph is acyclic

3. In SQL Server, under the READ UNCOMMITTED isolation level:
a. S locks must be acquired to perform read operations.
b. read operations are performed without acquiring S locks.
c. X locks must be acquired to perform write operations.
d. write operations are performed without acquiring X locks.
e. none of the above answers is correct.
Solution: BC

The schema of isolation levels and the corresponding concurrency issues: (**Seminar 3)**

Isolation Levels in SQL Server

concurrency probl. / isolation level	Chaos	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Lost Updates?	Yes	No	No	No	No
Dirty Reads?	Yes	Yes	No	No	No
Unrepeatable Reads?	Yes	Yes	Yes	No	No
Phantoms?	Yes	Yes	Yes	Yes	No

Locking in SQL Server
- lock types:
- *Shared* (S)
  - read operations
- *Update* (U)
  - deadlock avoidance mechanism
- *Exclusive* (X)
  - write operations
  - incompatible with other locks

	S	X
S	Yes	No
X	No	No

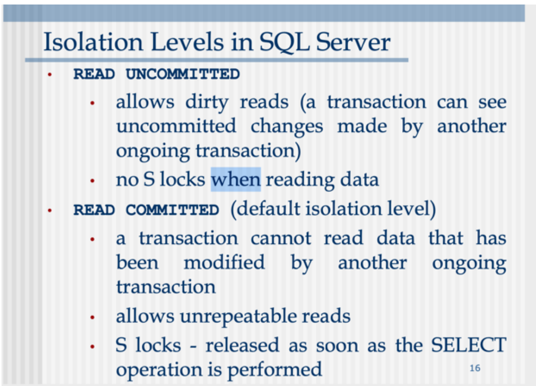
Locking in SQL Server
- lock types:
- *Exclusive* (X)
  - read operations by other transactions can be performed only when using the NOLOCK hint or the READ UNCOMMITTED isolation level
  - a transaction always acquires exclusive locks to modify data (regardless of the isolation level)
  - exclusive locks are released when the transaction completes execution

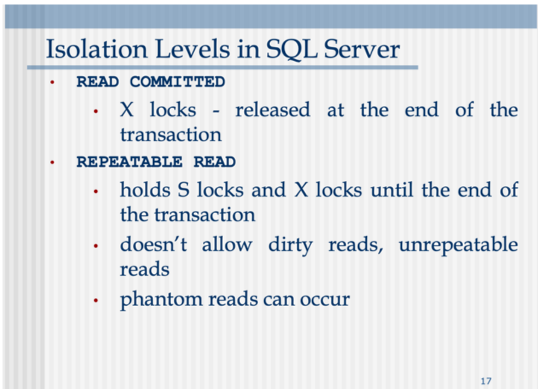
11

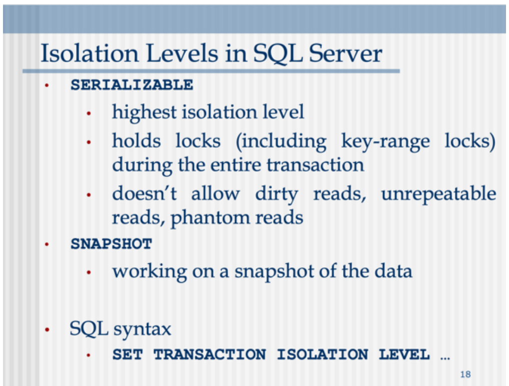
S – Shared Lock – read operation (SELECT)
X – Exclusive Lock – write operation (UPDATE / INSERT)

S – NOT acquired at Read Uncommitted isolation level
(S – NOT released at Read Uncommitted isolation level)
S – acquired at Read Committed, Repeatable Read, Serializable isolation levels
S – released as soon as the SELECT operation is performed for Read Committed isolation level
S – released at the end of the transaction for Repeatable Read and Serializable isolation levels

X – acquired at all the isolation levels: Read Uncommitted, Read Committed, Repeatable Read, Serializable
X – released at the end of the transaction for all the isolation levels: Read Uncommitted, Read Committed, Repeatable Read, Serializable







4. In horizontal fragmentation:
a. the reconstruction operator is the natural join.
b. the union of the horizontal fragments must be equal to the original relation.
c. fragmentation is performed with projection operators.
d. fragmentation is performed with selection predicates.
e. none of the above answers is correct.
Solution: BC

Lecture 10: Distributed Databases

Storing Data in a Distributed DBMS

* *fragmentation*: break a relation into smaller relations (fragments); store the fragments instead of the relation itself

- *horizontal* / *vertical* / *hybrid*

* example – relation Accounts(accnum, name, balance, branch):

- horizontal fragmentation
  - fragment: subset of rows
  - n selection predicates => n fragments (n record sets)
  - horizontal fragments should be disjoint
  - reconstruct the original relation: take the union of the horizontal fragments
    - $\sigma_{branch='Eroilor'}$(Accounts), $\sigma_{branch='Napoca'}$(Accounts), $\sigma_{branch='Motilor'}$(Accounts) =>

R
1, Radu, 250, Eroilor 2, Ana, 200, Napoca 3, Ionel, 150, Motilor 4, Maria, 400, Eroilor 5, Andi, 600, Napoca 6, Calin, 250, Eroilor 7, Iulia, 350, Motilor

R1	1, Radu, 250, Eroilor 4, Maria, 400, Eroilor 6, Calin, 250, Eroilor
R2	2, Ana, 200, Napoca 5, Andi, 600, Napoca
R3	3, Ionel, 150, Motilor 7, Iulia, 350, Motilor

Sabina S. CS

5. I is an index with search key <C1, C2, C3, C4>.
a. If I is a hash index, I matches condition *C1 > 10 AND C2 > 7.*
b. If I is a hash index, I matches condition *C1 = 10 AND C2 = 7 AND C3 = 1 AND C4 = 5.*
c. If I is a B+ tree index, I matches condition *C1 = 10 AND C2 = 7.*
d. If I is a B+ tree index, I matches condition *C2 = 7 AND C3 = 9.*
e. none of the above answers is correct.
Solution: BCD

Lecture 7 - hash index on = conditions, not >, <.

Selection

- general selections
  - selections without disjunctions
- C - CNF condition without disjunctions
  - evaluation options:

  1. use the most selective access path
     - if it's an index I:
       - apply conjuncts in C that match I
       - apply rest of conjuncts to retrieved tuples
     - example
       - *c < 100 AND a = 3 AND b = 5*
         - can use a B+ tree index on *c* and check *a = 3 AND b = 5* for each retrieved tuple
         - can use a hash index on *a* and *b* and check *c < 100* for each retrieved tuple

Sabina S. CS

Selection

- general selections - selections without disjunctions
  - evaluation options:

  2. use several indexes - when several conjuncts match indexes using a2 / a3
     - compute sets of rids of candidate tuples using indexes
     - intersect sets of rids, retrieve corresponding tuples
     - apply remaining conjuncts (if any)
     - example: *c < 100 AND a = 3 AND b = 5*
       - use a B+ tree index on *c* to obtain rids of records that meet condition *c < 100* (R_1)
       - use a hash index on *a* to retrieve rids of records that meet condition *a = 3* (R_2)
       - compute $R_1 \cap R_2 = R_{int}$
       - retrieve records with rids in R_{int} (*R*)
       - check *b = 5* for each record in *R*

Sabina S. CS

Selection

- general selections
  - selections with disjunctions
- C - CNF condition with disjunctions, i.e., some conjunct *J* is a disjunction of terms
  - if some term *T* in *J* requires a file scan, testing *J* by itself requires a file scan
    - example: *a < 100* ∨ *b = 5*
      - hash index on *b*, hash index on *c*

    => check both terms using a file scan (i.e., best access path: file scan)
  - compare with the example below:
    - *(a < 100* ∨ *b = 5)* ∧ *c = 7*
    - hash index on *b*, hash index on *c*

    => use index on *c*, apply *a < 100* ∨ *b = 5* to each retrieved tuple (i.e., most selective access path: index)

Sabina S. CS

6. Let R be a relation with P pages. The cost of sorting R using *simple two-way merge sort* (i.e., with 3 pages in the buffer pool) is:
a. π^P
b. $2P(\log_4 P + 1)$

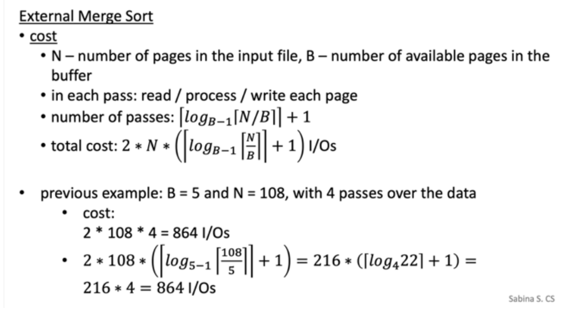
c. 2P($\log_2$P +1)

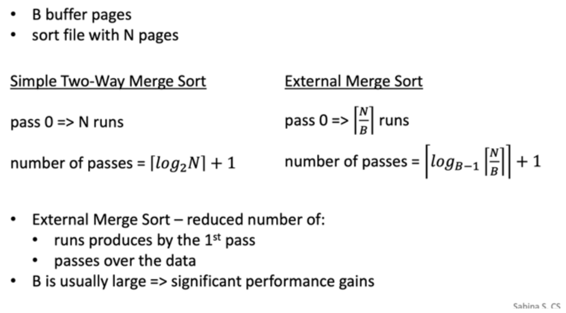
d. 2P($\log_3$P +1)

e. none of the above answers is correct.

Solution: C

Lecture 7:





7. Consider the query:
SELECT *
FROM R1, R2, R3
WHERE p1 AND p2 AND p3

The conditions tested by the predicates in the WHERE clause are statistically independent. The cardinality of a relation R is denoted by |R|. The reduction factor associated with predicate p is denoted by RF(p).

The cardinality of the query's result set can be estimated by:

a. $\frac{|R1|*|R2|*|R3|}{RF(p1)+RF(p2)+RF(p3)}$

b. |R1|*|R2|*|R3|*RF(p1)*RF(p2)*RF(p3)

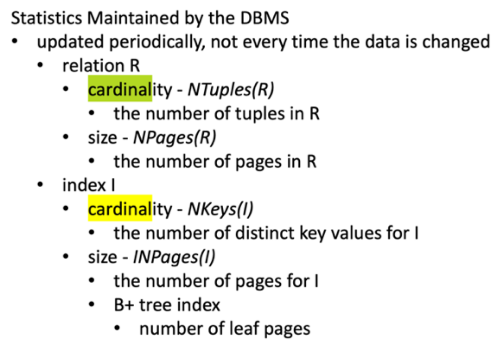
c. RF(p1)*RF(p2)*RF(p3) – (|R1|+|R2|+|R3|)

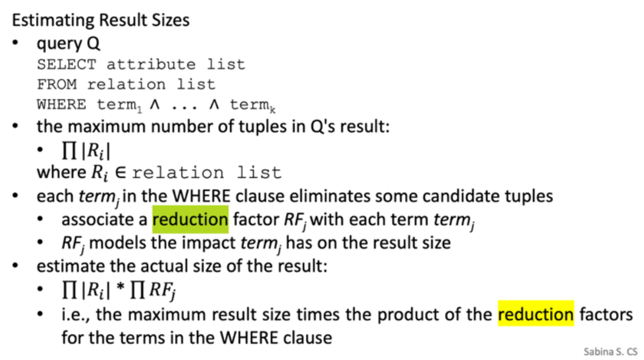
d. |R1|+|R2|+|R3|+ RF(p1)+RF(p2)+RF(p3)

e. none of the above answers is correct.

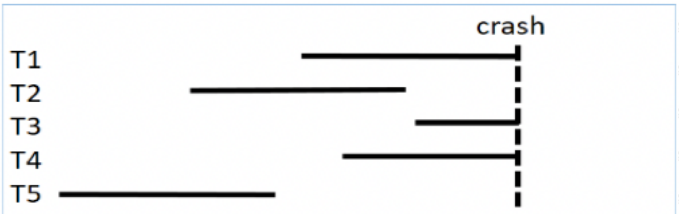
Solution: B

Lecture 9:





8. Consider the execution below. When the system comes back up after the crash, it must ensure that:



a. T1, T3, T4 are durable; T2 and T5 are undone.
b. T1, T3, T4 are undone; T2 and T5 are durable.
c. T1 is undone only if T2 and T4 are also undone.
d. T2 is durable only if T5 is undone.
e. none of the above answers is correct.

Solution: B
T2 + T5 – durabile – due to their finish before the crash
T1 + T3 + T4 – undone – due to the crash during their execution

9. In data replication:
a. *primary site replication* is an asynchronous replication technique. b. *primary site replication* is a synchronous replication technique.
c. *read-any write-all* is a synchronous replication technique.
d. *read-any write-all* is an asynchronous replication technique.
e. none of the above answers is correct.
Solution: AC

Lecture 10: Distributed Databases

Updating Distributed Data – Asynchronous Replication
- two approaches:
  - *primary site replication*
  - *peer-to-peer replication*
  - * difference: number of *updatable copies* (*master copies*)

Updating Distributed Data – Synchronous Replication
- 2 basic techniques: *voting* and *read-any write-all*

* *voting*
- to modify object O, a transaction T1 must write a majority of its copies
- when reading O, a transaction T2 must read enough copies to make sure it's seeing at least one current copy
- e.g., O has 10 copies; T1 changes O: suppose T1 writes 7 copies of O; T2 reads O: it should read at least 4 copies to make sure one of them is current
- each copy has a version number (the copy that is current has the highest version number)
- not an attractive approach in most cases, because reads are usually much more common than writes (and reads are expensive in this approach)

Sabina S. CS

10. A database access request contains:
a. the requesting user.
b. the criminal record of the requesting user.
c. the operation the user wants to perform.
d. the requested object.
e. none of the above answers is correct.
Solution: ACD

Lecture 5:

* the DBMS's authorization subsystem (security subsystem)
- checks any given access request against the applicable constraints
- access request:
    - requested object + requested operation + requesting user

  example: *Alice wants to delete 10 rows from table Customers.*
- identifying the applicable constraints for an access request:
    - the system must recognize the source of the request, i.e., the requesting user
    - an authentication mechanism is used for this purpose, e.g.:
        - password scheme
            - users supply their user ID (users say who they are) and their password (users prove they are who they say they are)
        - fingerprint readers, voice verifiers, retinal scanners, etc.

Sabina S. CS

11. Consider schedule S below over transactions T1, T2, T3, T4 (all transactions commit):

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

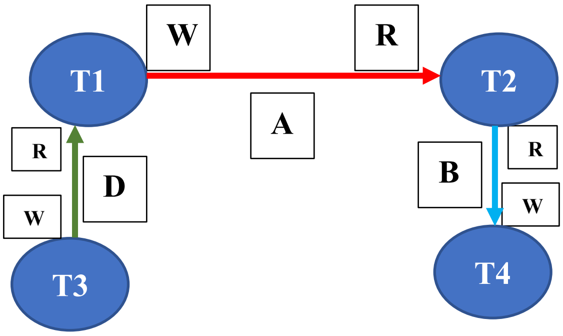
a. S is conflict serializable.
b. S is not conflict serializable.
c. (R(T4, C), R(T1, C)) belongs to the conflict relation of S.
d. (W(T1, A), R(T2, A)) belongs to the conflict relation of S.
e. none of the above answers is correct.
Solution: AD

All the time start from the transaction where is the first operation (R/W).
(e.g. T1- W(A) is first and then T2 – R(A)).

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

So,

- [T1 with W(A)] [T2 with R(A)] – from Precedence graph 2. (read after write): **(T1, T2) arc**
- [T3 with W(D)] [T1 with R(D)] – from Precedence graph 2. (read after write): **(T3, T1) arc**
- [T2 with R(B)] [T4 with W(B)] – from Precedence graph 3. (write after read): **(T2, T4) arc**
- [T4 with R(C)] [T1 with R(C)] – **no arc** due to Precedence graph (read after read)



Lecture 2:

Conflict Serializability - Precedence Graph

- let S be a schedule in $Sch(C)$
- the ***precedence graph*** (***serializability graph***) of S contains*:*
  - one node for every committed transaction in S
  - an arc from T_i to T_j if an action in T_i precedes and conflicts with one of the actions in T_j
- Theorem:
  - a schedule $S \in Sch(C)$ is conflict serializable if and only if its precedence graph is acyclic

Conflict Serializability - Precedence Graph

- algorithm to test the conflict serializability of a schedule $S \in Sch(C)$

1. create a node labeled T_i in the precedence graph for every committed transaction T_i in the schedule
2. create an arc (T_i,T_j) in the precedence graph if T_j executes a Read(A) after a Write(A) executed by T_i
3. create an arc (T_i,T_j) in the precedence graph if T_j executes a Write(A) after a Read(A) executed by T_i
4. create an arc (T_i,T_j) in the precedence graph if T_j executes a Write(A) after a Write(A) executed by T_i
5. S is conflict serializable if and only if the resulting precedence graph has no cycles

Sabina S. CS

GOOD LUCK ☺