Performance Measure of disks

Access time

1. Seek time: time to reposition the arm to the correct track
2. Rotational latency: time it takes for the sector to be accessed under the head

Data-transfer rate

Mean time to failure

**Optimization of Disk-block access**

Read/Write by block

sizes range from a sector to hundreds of sector

typically 4kb to 16kb

Disk-arm scheduling

elevator algorithm

File organization

group the related information

defragment

Nonvolatile write buffers

log disk

**RAID**

high capacity and high speed

high reliability

RAID-0

non-redundant, most performance, potentially data loss

RAID-1

mirroring, most writing performance, higher cost

RAID-5

partitions data and parity among all N+1 disks

preferred with low update rate

Record

**Fix-length Records**

each record has the same length

insert

do not allow cross block record

delete

use a linked list to link all free slots

**Variable-length Records**

Slotted page header

each record has a pointer in the header

has free space between header and records

Sequential File Organization

Delete - use pointer chains

Insert - locate, insert in free space : insert overflow space

need to reorganize the file from time to time to restore the sequence

Multi-table clustering file organization

good for queries involving department \* instructor

Buffer Manager…

Dense Index File

index every thing

generally faster

but larger and hard to maintain

Sparse Index File

index some key-value

actually index block

easy to maintain, slower

a good tradeoff

Secondary indices

we want to find all records in a certain filed

have to be dense

when file is modified each index on the file must be updated

**B/B+ tree**

Leaf node: values

Non-leaf node: children

root: at least two children

**Static Hashing**

obtain the bucket of a record directly

random or uniform

Deficiencies

the number of buckets, if large then probably wastes

if small then probably not enough, and performance degrade due to too much overflows

**Dynamic Hashing**

use prefix i to map to buckets

database shrinks and extends dynamically

Benefits

Performance doesn't degrade with growth of file

Minimal space overhead

Disadvantage

Extra level of indirection to find desired record

Address table maybe very big(large than memory)

Basic steps in query processing

Parsing and translation

Optimization

Evaluation

How to measure costs

Generally measured as total elapsed time

disk access, CPU, network communication

Disk access

number of seeks

number of blocks read

number of blocks written

For simplicity

number of block transfers

number of seeks

cost for b block transfers plus S seeks

ignore CPU costs

**Selection**

Linear Search

Scan each file block and test all records.

if selection is a key attribute

denotes number of blocks containing records from relation r

Primary index, equality on key

Primary index, equality on non-key

Secondary index, equality on non-key

if retrieve a single record if the search-key is a candidate key

retrieve multiple records if search-key is not a candidate key

Primary index, comparison

for use index to find first tuple >=v and scan relation sequentially

for just scan relation sequentially till first tuple > v; do not use index.

Secondary index, comparison

**External Sort-Merge**

Total number of merge passes required:

Total number of block transfer for external sorting:

Cost of seeks

**Join Operation**

Examples use the following information

Number of records of student: 5,000 takes: 10,000

Number of blocks of student: 100 takes: 400

r is called outer relation, s is inner relation of the join

**Nested-Loop Join**

block transfers, plus

seeks

**Block Nested-Loop Join**

worst case:

Best case:

Improvement:

**Indexed Nested-Loop Join**

C is the cost of traversing index and fetching all matching s tuples for one tuple or r

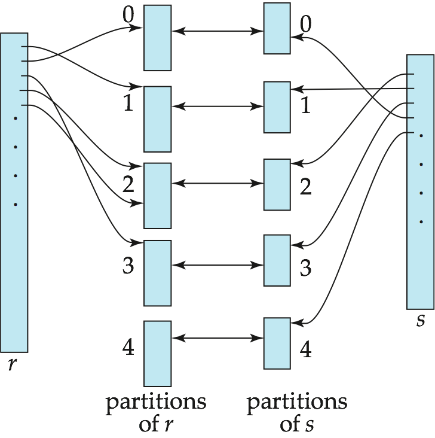
c can be estimated as cost of a single selection on s using the join condition

**Merge Join**

Sort first, then join the two relation

Can be used only for equal-joins and natural joins

Hash Join



Evaluation of Expressions

1. Materialization

2. Pipelining

**Query Optimization**

1. Pushing Selections

2. Pushing Projections

reduce the size of the relation to be joined

3. Join order matters

Evaluation Plan example:

(sort to remove duplicates) 
11 
customer—name 
(hash join) 
(merge join) 
depositor 
pipeline 
branch_city = Brooklyn 
(use index I) 
branch 
pipeline 
G balance < 1000 
(use linear scan) 
account 

Statistical Information for Cost Estimation

**Transaction**

State

Active - the initial state.

Partially committed - after the final statement has been executed.

Failed - after finding the normal execution can no longer proceed.

Aborted - after the transaction has been rolled back

Committed - after successful completion

**Serializability**

conflict serializability

conflict equivalent: a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions.

conflict serializable: if it is conflict equivalent to a serial schedule

view serializability

S and S' be two schedules with the same set of transactions.

if S read the initial value of Q, then S' must read the initial value of Q.

if S read the data written by T\_j, then S' must also read it.

if S performs the final write(Q) operation in schedule S must also perform the final write(Q) operation in schedule S'.

view equivalence is also based purely on reads and writes.

conflict serializable => view serializable

**Testing for Conflict Serializability**

**Precedence graph**

a direct graph where the vertices are the transactions(names)

We draw an arc from to if the two transaction conflict, and accessed the data item on which the conflict arose earlier.

A schedule if conflict serializable if and only is its precedence graph is acyclic

Serializability order can be obtained by a topological sorting.

Test for View Serializability

falls in the class of NP-complete problems.

has some sufficient conditions for view serializability

**Recoverable Schedules**

If a transaction reads a data item previously written by a transaction , then the commit operation of appears before the commit operation of .

The following schedule (Schedule 11) is not recoverable if T 9 commits 
immediately after the read 
read (A) 
write (A) 
read (B) 
read (A) 
commit 

if should abort, would have read an inconsistent database state.

**Cascading Rollbacks**

a single transaction failure leads to a series of transaction rollbacks.

Cascade-less Schedules: cascading rollbacks cannot occur; for each pair of transactions and such that reads a data item previously written by , the commit operation of appears before the read operation of

**Concurrency Control**

**Lock-Based Protocols**

exclusive mode

shared mode

Pitfalls:

deadlock, request each other

Starvation, can be prevented though.

**Two-Phase Locking Protocol**

Growing Phase

may obtain locks

may not release locks

Shrinking Phase

may release locks

may not release locks

Lock points: from obtaining locks to releasing locks.

strict two-phase locking, must hold all its exclusive locks till it commits/aborts.

rigorous two-phase locking, all locks are held till commit/aborts.

**Graph-Based Protocols**

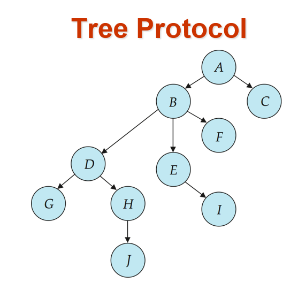
if then any transaction accessing both and must access before access .

Only exclusive locks are allowed.

The first lock by may be on any data item. Subsequently, a data Q can be locked by only if the parent of Q is currently locked by

Data item can be unlocked at any time

A data item that has been locked and unlocked by cannot subsequently be relocked by



**Deadlock Detection**

18 
19 
20 
18 
17 
19 
20 
Wait-for graph without a cycle 
Wait-for graph with a cycle 

Deadlock Recovery

Total rollback: Abort the transaction and then restart it.

**Multiple Granularity**

intention-shared(IS)

intention-exclusive(IX)

shared and intention-exclusive(SIX)

Mode 
NL 
IS 
IX 
SIX 
X 
NL 
Yes 
Yes 
Yes 
Yes 
Yes 
Yes 
IS 
Yes 
Yes 
Yes 
Yes 
Yes 
No 
IX 
Yes 
Yes 
Yes 
No 
No 
No 
s 
Yes 
Yes 
No 
Yes 
No 
No 
SIX 
Yes 
Yes 
No 
No 
No 
No 
x 
Yes 
No 
No 
No 
No 
No 

1. The look compatibility matrix must be observed.
2. The root of the tree must be locked first, and may be locked in any mode.
3. A node Q can be locked by in S or IS mode only is the parent of Q is currently locked by in either IX or IS mode.
4. A node Q can be locked by in X, SIX, or IX mode only if the parent of Q is currently locked by in either IX or SIX mode.
5. can lock a node only if it has not previously unlocked and node(two-phase)
6. can unlock a node Q only if none of the children of Q are currently locked by

Locks are acquired in root-to-leaf order, whereas they are released in leaf-to-root order.

**Recovery System**

Failure Classification

Transaction failure:

Logical errors: internal error

System errors: killed by system due to deadlock, for example

System crash

Disk failure

**Log-Based Recovery**

Log is kept on stable storage, a sequence of log records

1. Register itself by writing a log record
2. Before executes write(X), a log record is written, where is the value of X before the write, and is the value to be written to X.
3. When finishes it last statement, the log record is written.

Deferred database modification

records all modification to the log, but defers all the writes to after partial commit

A write(X) operation do not need to log old value

Immediate Database Modification

**Undo and Redo of Transactions**

Undo restores the value of all data items updated by to their old values, going backwards from the last log record for

Redo set the value of all data items updated by to the new values, going forward from the first log record for

Checkpoint

checkpoint 
system failure 

can be ignored, and redone, undone.

Beginning of log 
< To start> 
<To, B, 2000, 2050> 
< To commit> 
< T, start> 
B, 2050, 2100> 
04, operation-begin> 
<checkpoint { 
C, 700, 400> 
04, operation-end, (C, 
< T2 start> 
End of 
< Tz 05, operation-begin> 
log at 
<T2, C, 400, 300> 
crash! 
<T2, C, 400> 
< T2 abort> 
Records 
C, 400, 700> 
added 
04, operation-abort> 
during 
<Tt, B, 2050> 
recovery 
< T, abort> 
Start log records 
found for all 
transactions in 
undo list 
Redo Pass 
Undo list: T t, T 2 Undo Pass 
Update of C was part of 05, undon 
physically during recovery since 
05 did not complete 
Logical undo of O, adds 300 to C 

**ARIES**

1. Uses log sequence number (LSN) to identify log records

Stores LSNs in pages to identify what updates have already been applied to a database page

1. Physiological redo
2. Dirty page table to avoid unnecessary redos during recovery
3. Fuzzy checkpointing that only records information about dirty pages, and does not require dirty pages to be written out at checkpoint time

More coming up on each of the above …

Pa elD Pa eLSN 
4894 
7567 
7200 
7565 
Dirty Page Table 
RecLSN 
7564 
7565 
Page 4894 
7565 
Page 7200 
Database Buffer 
Stable data 
4 
Page 4894 
Page 9923 
Page 9923 
44 4 
Page 7200 
7567: , 40, 
7566: <T143 commit> 
Log Buffer (PrevLSN and UndoNextLSN 
fields not shown) 
Stable log 
7565: 60, 
7564: , 20, 
7563: 

newer 
End of log at crash 
7571 : <T146 commit> 
7570: <T146, 2390.4, 50, 
7569: <T146 begin> 
7568: checkpoint 
PrevLSN 
pointers 
TX n 
Tl 45 
Pa elD 
4894 
7200 
lastLSN 
7567 
Pa eLSN 
7567 
7565 
RecLSN 
7564 
7565 
Undo 
pass 
7567: 40, 
7566: 
7563: 
<T143 commit> 
older 
7565: 
7564: 20, 
<T145 begin> 
7562: 60, 
Analysis 
pass 
CLR 
Redo 
pass 
UndoNextLSN 

ARIES data structures

1. Uses **log sequence number (LSN)** to identify log records

Stores LSNs in pages to identify what updates have already been applied to a database page

1. Physiological redo
2. Dirty page table to avoid unnecessary redos during recovery
3. Fuzzy checkpointing that only records information about dirty pages, and does not require dirty pages to be written out at checkpoint time

More coming up on each of the above …

XML部分没有包括，但事实上会考大题，请注意