Lecture 27:

Friday, December 6, 2002

Outline

- Cost estimation: 16.4
- Recovery using undo logging 17.2

Estimating Sizes

- Need size in order to estimate cost
- Example:
  – Cost of partitioned hash-join E1 ⋈ E2 is 3B(E1) + 3B(E2)
  – B(E1) = T(E1) * record size/ block size
  – B(E2) = T(E2) * record size/ block size
  – So, we need to estimate T(E1), T(E2)

Estimating Sizes

Estimating the size of a projection

- Easy: $T(\Pi_L(R)) = T(R)$
- This is because a projection doesn't eliminate duplicates

Estimating Sizes

Estimating the size of a selection

- $S = \sigma_{A=c}(R)$
  – T(S) san be anything from 0 to T(R) – V(R,A) + 1
  – Mean value: T(S) = T(R)/V(R,A)
- $S = \sigma_{A<c}(R)$
  – T(S) can be anything from 0 to T(R)
  – Heuristics: T(S) = T(R)/3

Estimating Sizes

Estimating the size of a natural join, $R \bowtie_A S$

- When the set of A values are disjoint, then $T(R \bowtie_A S) = 0$
- When A is a key in S and a foreign key in R, then $T(R \bowtie_A S) = T(R)$
- When A has a unique value, the same in R and S, then $T(R \bowtie_A S) = T(R)\ T(S)$

Estimating Sizes

Assumptions:

- *Containment of values*: if V(R,A) <= V(S,A), then the set of A values of R is included in the set of A values of S
  - Note: this indeed holds when A is a foreign key in R, and a key in S
- *Preservation of values*: for any other attribute B, V(R $\bowtie_A$ S, B) = V(R, B) (or V(S, B))

7

Estimating Sizes

Assume V(R,A) <= V(S,A)

- Then each tuple t in R joins *some* tuple(s) in S
  - How many ?
  - On average T(S)/V(S,A)
  - t will contribute T(S)/V(S,A) tuples in R $\bowtie_A$ S
- Hence T(R $\bowtie_A$ S) = T(R) T(S) / V(S,A)

In general: T(R $\bowtie_A$ S) = T(R) T(S) / max(V(R,A),V(S,A))

8

Estimating Sizes

Example:

- T(R) = 10000, T(S) = 20000
- V(R,A) = 100, V(S,A) = 200
- How large is R $\bowtie_A$ S ?

Answer: T(R $\bowtie_A$ S) = 10000 20000/200 = 1M

9

Estimating Sizes

Joins on more than one attribute:

- T(R $\bowtie_{A,B}$ S) =

  T(R) T(S)/(max(V(R,A),V(S,A))*max(V(R,B),V(S,B)))

10

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

11

Histograms

Employee(ssn, name, salary, phone)

- Maintain a histogram on salary:

Salary:	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
Tuples	200	800	5000	12000	6500	500

- T(Employee) = 25000, but now we know the distribution

12

Histograms

Ranks(rankName, salary)

- Estimate the size of Employee $\bowtie_{Salary}$ Ranks

Employee	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	200	800	5000	12000	6500	500

Ranks	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	8	20	40	80	100	2

13

Histograms

V(Employee, Salary) = 200 V(Ranks, Salary) = 250

Employee $\bowtie_{Salary}$ Ranks=
Employee$_1$ $\bowtie_{Salary}$ Ranks$_1$$\cup$ ... $\cup$Emplyee$_6$ $\bowtie_{Salary}$ Ranks$_6$

- A tuple t in Employee$_1$ joins with so many tuples in Ranks$_1$:
  T(Employee$_1$)/T(Employee) * T(Employee)/250 = T_1 / 250
- Then T(Employee $\bowtie_{Salary}$ Ranks) =
  = $\Sigma_{i=1,6} T_i T_i' / 250$
  = (200x8 + 800x20 + 5000x40 +
  12000x80 + 6500x100 + 500x2)/250
  = ….

14

Recovery

Type of Crash	Prevention
Wrong data entry	Constraints and Data cleaning
Disk crashes	Redundancy: e.g. RAID, archive
Fire, theft, bankruptcy…	Buy insurance, Change jobs…
System failures: e.g. power	DATABASE RECOVERY

Most frequent

15

System Failures

- Each transaction has *internal state*
- When system crashes, internal state is lost
  - Don't know which parts executed and which didn't
- Remedy: use a **log**
  - A file that records every single action of the transaction

16

Transactions

A **transaction** = piece of code that must be executed atomically

- In ad-hoc SQL
  - one command = one transaction
- In embedded SQL
  - Transaction starts = first SQL command issued
  - Transaction ends =
    - COMMIT
    - ROLLBACK (=abort)

17

Transactions

- Assumption: the database is composed of ***elements***
  - Usually 1 element = 1 block
  - Can be smaller (=1 record) or larger (=1 relation)
- Assumption: each transaction reads/writes some elements

18

Primitive Operations of Transactions

- INPUT(X)
  - read element X to memory buffer
- READ(X,t)
  - copy element X to transaction local variable t
- WRITE(X,t)
  - copy transaction local variable t to element X
- OUTPUT(X)
  - write element X to disk

19

Example

READ(A,t); t := t*2;WRITE(A,t); READ(B,t); t := t*2;WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
REAT(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

20

The Log

- An append-only file containing log records
- Note: multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo some transaction that didn't commit
  - Undo other transactions that didn't commit
- Three kinds of logs: undo, redo, undo/redo

21

Undo Logging

Log records

- <START T>
  - transaction T has begun
- <COMMIT T>
  - T has committed
- <ABORT T>
  - T has aborted
- <T,X,v>
  - T has updated element X, and its *old* value was v

22

Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before X is output to disk

U2: If T commits, then <COMMIT T> must be written to disk only after all changes by T are output to disk

- Hence: OUTPUTs are done *early*, before the transaction commits

23

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						<COMMIT T>

24

Recovery with Undo Log

After system's crash, run recovery manager

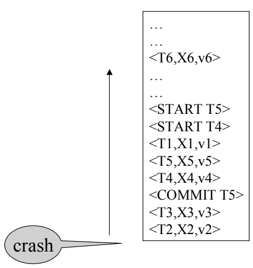
- Idea 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>…. = yes
  - <START T>….<ABORT T>……. = yes
  - <START T>……………………… = no
- Idea 2. Undo all modifications by incomplete transactions

Recovery with Undo Log

Recovery manager:

- Read log <u>from the end</u>; cases:
  - <COMMIT T>: mark T as completed
  - <ABORT T>: mark T as completed
  - <T,X,v>: if T is not completed
then write X=v to disk
else ignore
  - <START T>: ignore

Recovery with Undo Log



Question1 in class: Which updates are undone ?

Question 2 in class: How far back do we need to read in the log ?

Recovery with Undo Log

- Note: all undo commands are <u>*idempotent*</u>
  - If we perform them a second time, no harm is done
  - E.g. if there is a system crash during recovery, simply restart recovery from scratch

Recovery with Undo Log

When do we stop reading the log ?

- We cannot stop until we reach the beginning of the log file
- This is impractical
- Better idea: use checkpointing

Checkpointing

Checkpoint the database periodically

- Stop accepting new transactions
- Wait until all current transactions complete
- Flush log to disk
- Write a <CKPT> log record, flush
- Resume transactions

Undo Recovery with Checkpointing

During recovery,
Can stop at first
<CKPT>

…
…
<T9,X9,v9>
…
…
(all completed)
<CKPT>
<START T2>
<START T3
<START T5>
<START T4>
<T1,X1,v1>
<T5,X5,v5>
<T4,X4,v4>
<COMMIT T5>
<T3,X3,v3>
<T2,X2,v2>

other transactions

transactions T2,T3,T4,T5

31

Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active

32

Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Continue normal operation
- When all of T1,…,Tk have completed, write <END CKPT>

33

Undo Recovery with Nonquiescent Checkpointing

During recovery,
Can stop at first
<CKPT>

…
…
…
…
…
…
<START CKPT T4, T5, T6>
…
…
…
…
<END CKPT>
…
…
…

earlier transactions plus T4, T5, T5

T4, T5, T6, plus later transactions

later transactions

Q: why do we need <END CKPT> ?

34