

Database Tuning

*Principles, Experiments and
Troubleshooting Techniques*

<http://www.mkp.com/dbtune>

Dennis Shasha (shasha@cs.nyu.edu)

Philippe Bonnet (bonnet@diku.dk)

Database Tuning

Database Tuning is the activity of making a database application run more quickly.

“More quickly” usually means higher throughput, though it may mean lower response time for time-critical applications.

Application

Programmer

(e.g., business analyst,
Data architect)

Application

Sophisticated
Application
Programmer

(e.g., SAP admin)

Query Processor

Indexes

Storage Subsystem

Concurrency Control

Recovery

Operating System

Hardware

[Processor(s), Disk(s), Memory]

© Dennis Shasha,

Philippe Bonnet, 2002

Outline

1. Basic Principles
 2. Tuning the guts
 3. Indexes
 4. Relational Systems
-
5. Application Interface
 6. Ecommerce Applications
 7. Data warehouse Applications
 8. Distributed Applications
 9. Troubleshooting

Goal of the Tutorial

- To show:
 - Tuning principles that port from one system to the other and to new technologies
 - Experimental results to show the effect of these tuning principles.
 - Troubleshooting techniques for chasing down performance problems.

Tuning Principles *Leitmotifs*

- Think globally, fix locally (does it matter?)
- Partitioning breaks bottlenecks (temporal and spatial)
- Start-up costs are high; running costs are low (disk transfer, cursors)
- Be prepared for trade-offs (indexes and inserts)

Experiments -- why and where

- Simple experiments to illustrate the performance impact of tuning principles.
- <http://www.diku.dk/dbtune/experiments> to get the SQL scripts, the data and a tool to run the experiments.

Experimental DBMS and Hardware

- Results presented throughout this tutorial obtained with:
 - SQL Server 7, SQL Server 2000, Oracle 8i, Oracle 9i, DB2 UDB 7.1
 - Three configurations:
 1. Dual Xeon (550MHz, 512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb) 2 Ultra 160 channels, 4x18Gb drives (10000RPM), Windows 2000.
 2. Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.
 3. Pentium III (1 GHz, 256 Kb), 1Gb RAM, Adapter 39160 with 2 channels, 3x18Gb drives (10000RPM), Linux Debian 2.4.

Tuning the Guts

- Concurrency Control
 - How to minimize lock contention?
- Recovery
 - How to manage the writes to the log (to dumps)?
- OS
 - How to optimize buffer size, process scheduling, ...
- Hardware
 - How to allocate CPU, RAM and disk subsystem resources?

Isolation

- Correctness vs. Performance
 - Number of locks held by each transaction
 - Kind of locks
 - Length of time a transaction holds locks

Isolation Levels

- Read Uncommitted (No lost update)
 - Exclusive locks for write operations are held for the duration of the transactions
 - No locks for read
- Read Committed (No dirty retrieval)
 - Shared locks are released as soon as the read operation terminates.
- Repeatable Read (no unrepeatable reads for read/write)
 - Two phase locking
- Serializable (read/write/insert/delete model)
 - Table locking or index locking to avoid phantoms

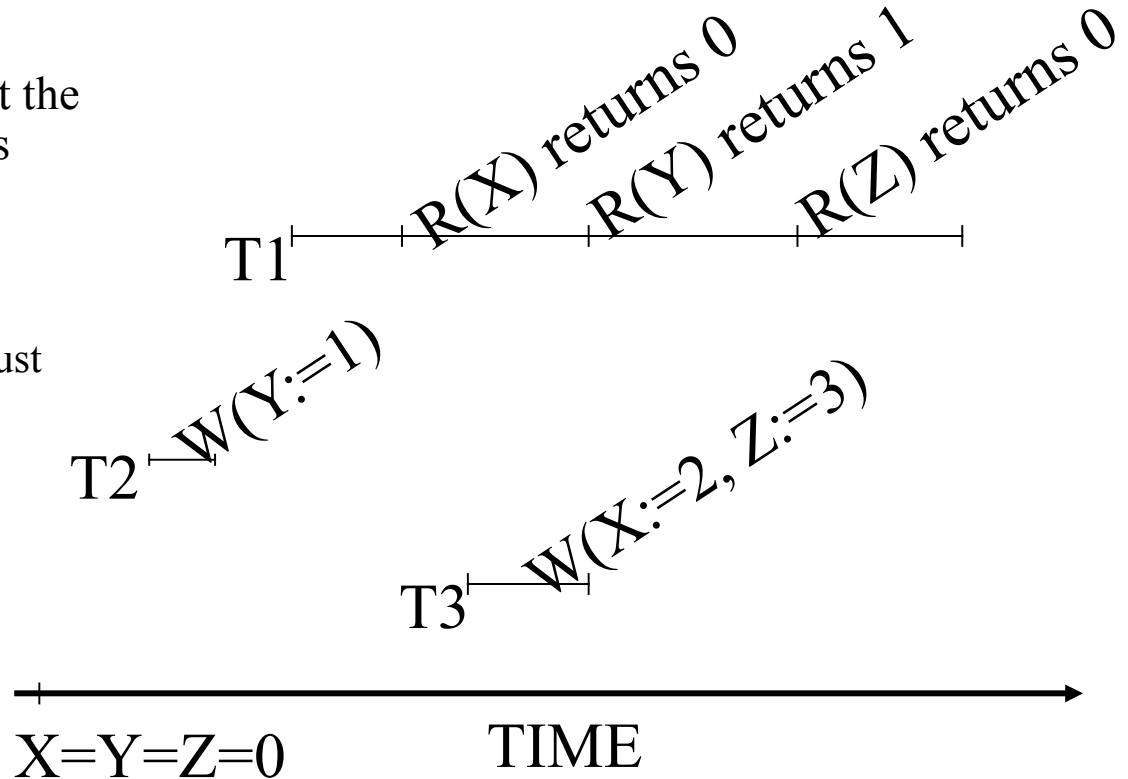
Snapshot isolation

- Each transaction executes against the version of the data items that was committed when the transaction started:

- No locks for read
- Costs space (old copy of data must be kept)

- Almost serializable level:

- T1: $x:=y$
- T2: $y:=x$
- Initially $x=3$ and $y=17$
- Serial execution:
 $x,y=17$ or $x,y=3$
- Snapshot isolation:
 $x=17, y=3$ if both transactions start at the same time.



Value of Serializability -- Data

Settings:

- ```
accounts (number, branchnum, balance);
create clustered index c on accounts(number);
```
- 100000 rows
  - Cold buffer; same buffer size on all systems.
  - Row level locking
  - Isolation level (SERIALIZABLE or READ COMMITTED)
  - SQL Server 7, DB2 v7.1 and Oracle 8i on Windows 2000
  - Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# Value of Serializability -- transactions

## Concurrent Transactions:

- T1: summation query [1 thread]  
`select sum(balance) from accounts;`
- T2: swap balance between two account numbers (in order of scan to avoid deadlocks) [N threads]

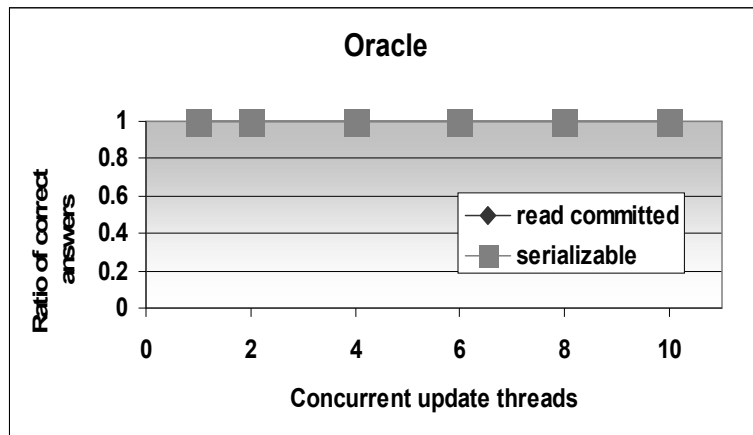
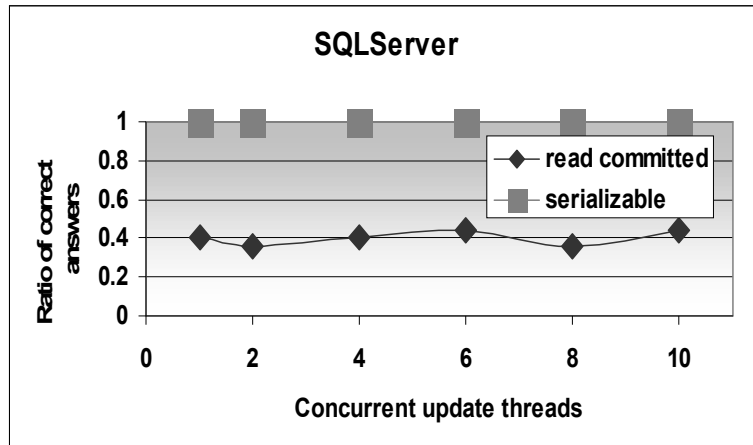
```
valX:=select balance from accounts where
number=X;
```

```
valY:=select balance from accounts where
number=Y;
```

```
update accounts set balance=valX where number=Y;
```

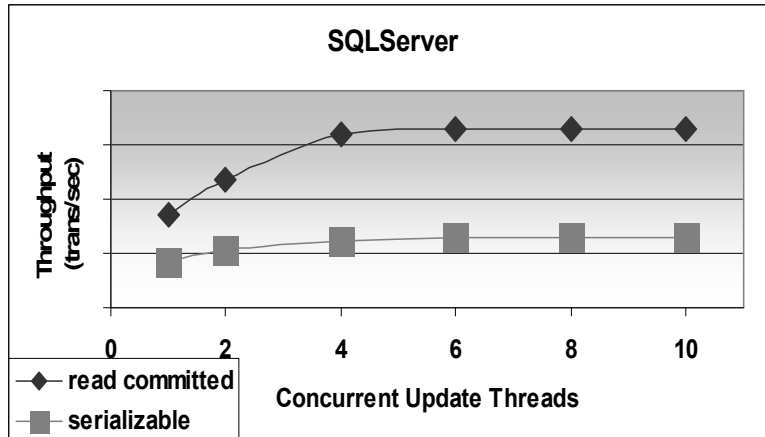
```
update accounts set balance=valY where number=X;
```

# Value of Serializability -- results

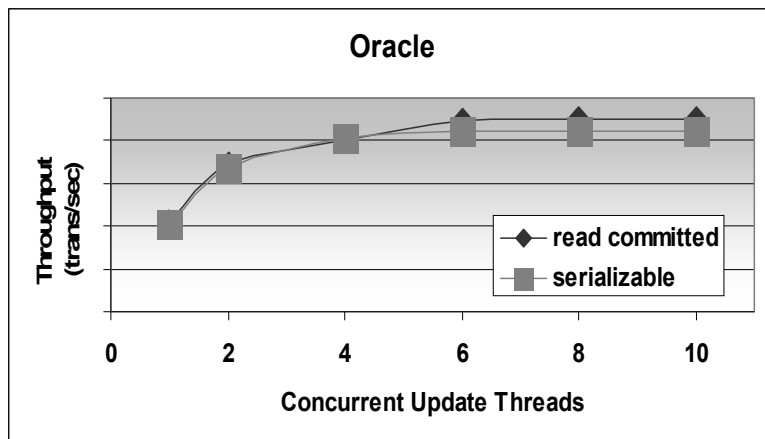


- With SQL Server and DB2 the scan returns incorrect answers if the read committed isolation level is used (default setting)
- With Oracle correct answers are returned (snapshot isolation), but beware of swapping

# Cost of Serializability



Because the update conflicts with the scan, correct answers are obtained at the cost of decreased concurrency and thus decreased throughput.





# Locking Overhead -- data

## Settings:

```
accounts (number, branchnum, balance);
create clustered index c on accounts(number);
```

- 100000 rows
- Cold buffer
- SQL Server 7, DB2 v7.1 and Oracle 8i on Windows 2000
- No lock escalation on Oracle; Parameter set so that there is no lock escalation on DB2; no control on SQL Server.
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# Locking Overhead -- transactions

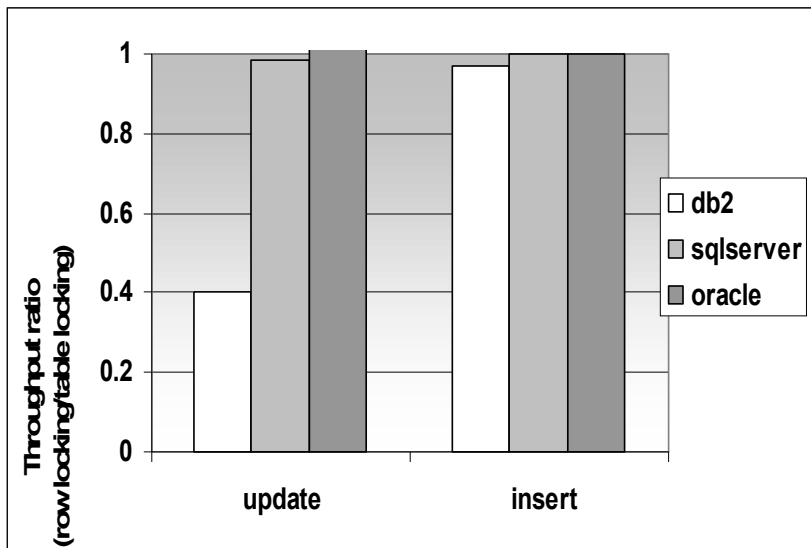
## No Concurrent Transactions:

- Update [10 000 updates]  
`update accounts set balance = Val;`
- Insert [10 000 transactions], e.g. typical one:  
`insert into accounts  
values (664366, 72255, 2296.12);`

# Locking Overhead

Row locking is barely more expensive than table locking because recovery overhead is higher than row locking overhead

- Exception is updates on DB2 where table locking is distinctly less expensive than row locking.



# Logical Bottleneck: Sequential Key generation

- Consider an application in which one needs a sequential number to act as a key in a table, e.g. invoice numbers for bills.
- Ad hoc approach: a separate table holding the last invoice number. Fetch and update that number on each insert transaction.
- Counter approach: use facility such as Sequence (Oracle)/Identity(MSSQL).

# Counter Facility -- data

## Settings:

```
accounts (number, branchnum, balance);
create clustered index c on accounts(number);
```

```
counter (nextkey);
insert into counter values (1);
```

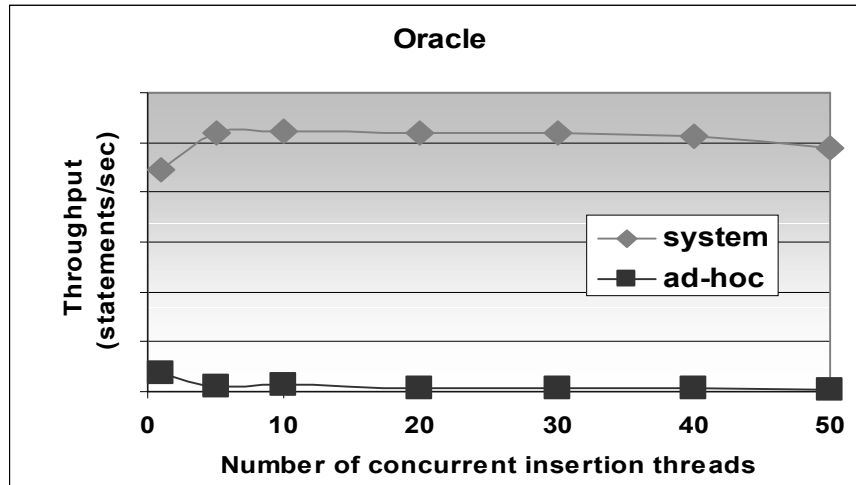
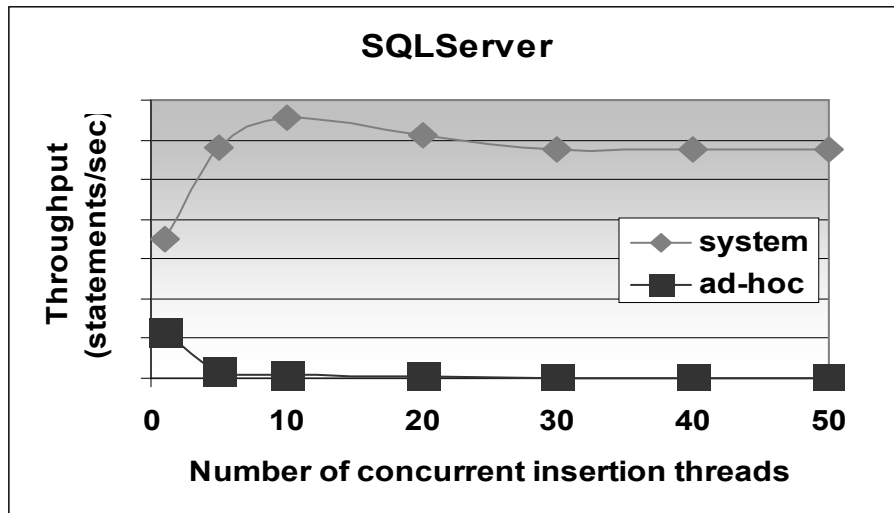
- default isolation level: READ COMMITTED; Empty tables
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# Counter Facility -- transactions

## No Concurrent Transactions:

- System [100 000 inserts, N threads]
  - SQL Server 7 (uses Identity column)  
`insert into accounts values (94496,2789);`
  - Oracle 8i  
`insert into accounts values (seq.nextval,94496,2789);`
- Ad-hoc [100 000 inserts, N threads]  
`begin transaction`  
`NextKey:=select nextkey from counter;`  
`update counter set nextkey = NextKey+1;`  
`commit transaction`  
`begin transaction`  
`insert into accounts values(NextKey,?,?);`  
`commit transaction`

# Avoid Bottlenecks: Counters



- System generated counter (system) much better than a counter managed as an attribute value within a table (ad hoc).
- The Oracle counter can become a bottleneck if every update is logged to disk, but caching many counter numbers is possible.
- Counters may miss ids.

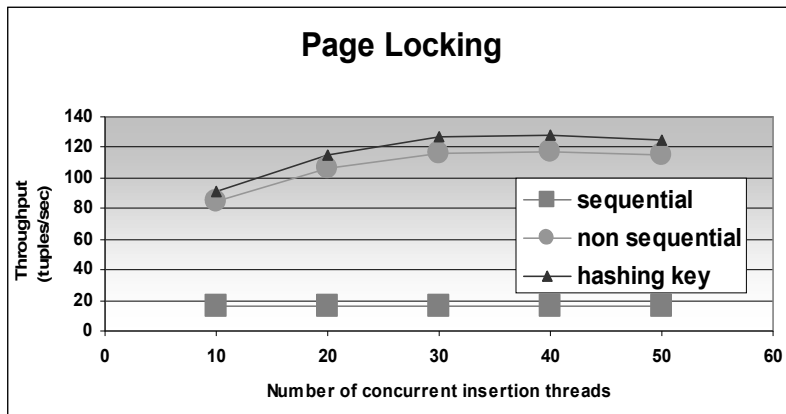
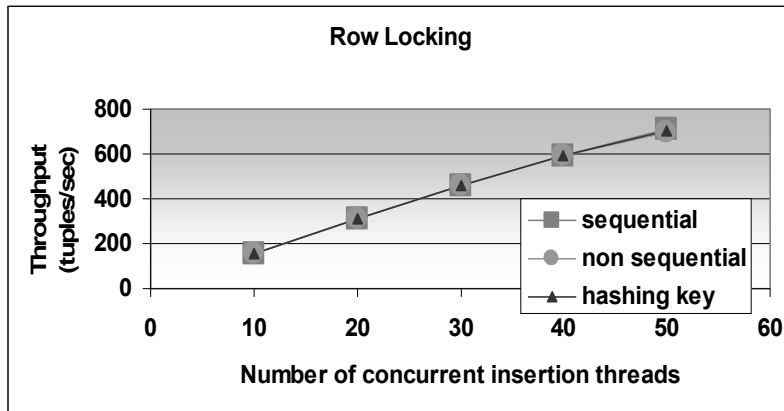
# Insertion Points -- transactions

## No Concurrent Transactions:

- Sequential [100 000 inserts, N threads]  
Insertions into account table with clustered index on ssnnum  
Data is sorted on ssnnum  
Single insertion point
- Non Sequential [100 000 inserts, N threads]  
Insertions into account table with clustered index on ssnnum  
Data is not sorted (uniform distribution)  
100 000 insertion points
- Hashing Key [100 000 inserts, N threads]  
Insertions into account table with extra attribute att with clustered index on (ssnnum, att)  
Extra attribute att contains hash key (1021 possible values)  
1021 insertion points

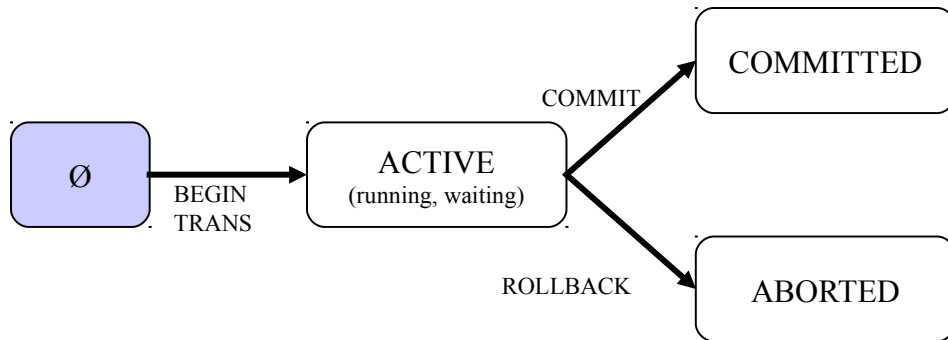


# Insertion Points

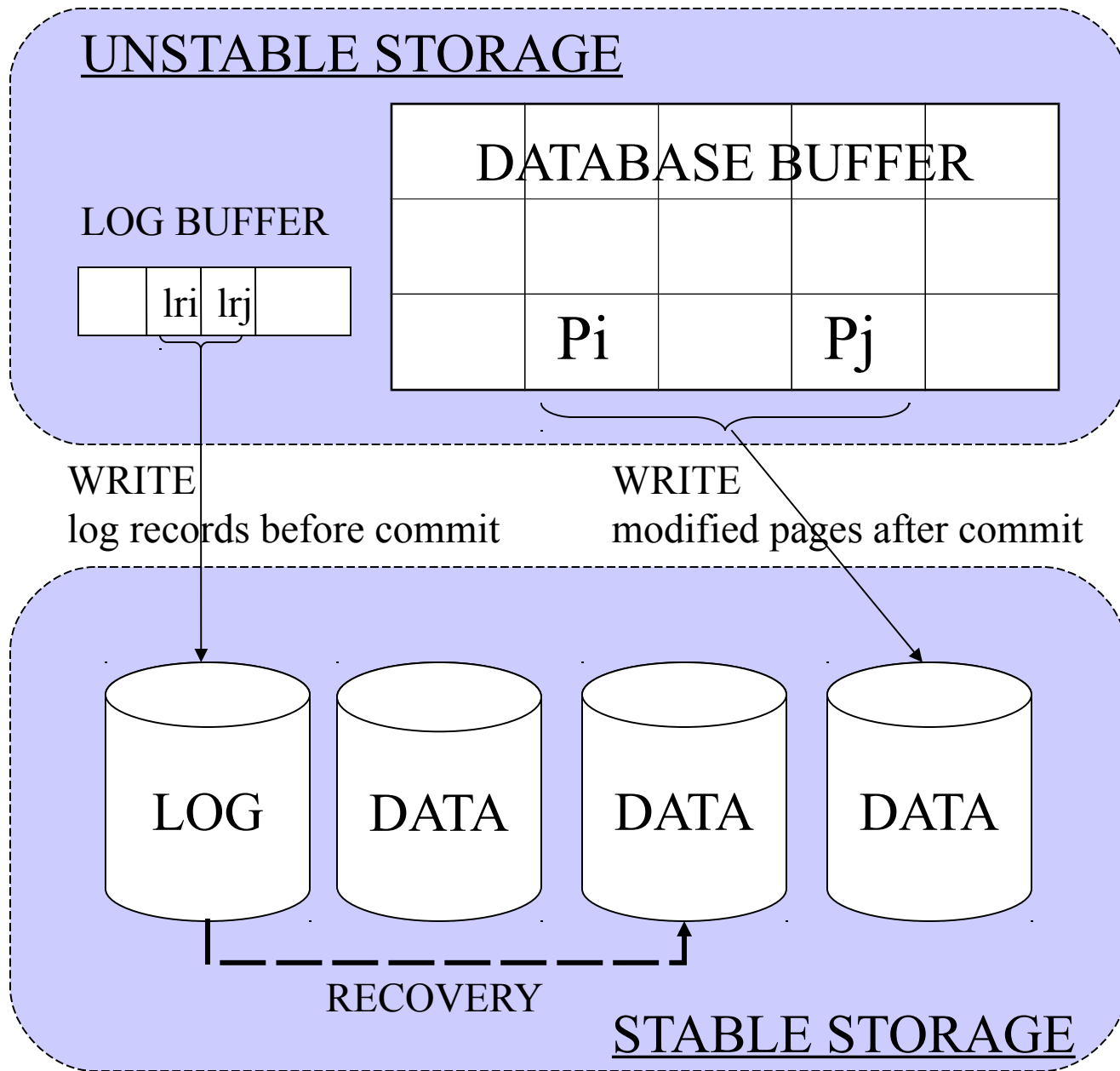


- Page locking: single insertion point is a source of contention (sequential key with clustered index, or heap)
- Row locking: No contention between successive insertions.
- DB2 v7.1 and Oracle 8i do not support page locking.

# Atomicity and Durability



- Every transaction either commits or aborts. It cannot change its mind
- Even in the face of failures:
  - Effects of committed transactions should be permanent;
  - Effects of aborted transactions should leave no trace.



# Log IO -- data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
L_LINENUMBER , L_QUANTITY, L_EXTENDEDPRICE ,
L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
L_SHIPDATE, L_COMMITDATE,
L_RECEIPTDATE, L_SHIPINSTRUCT ,
L_SHIPMODE , L_COMMENT);
```

- READ COMMITTED isolation level
- Empty table
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

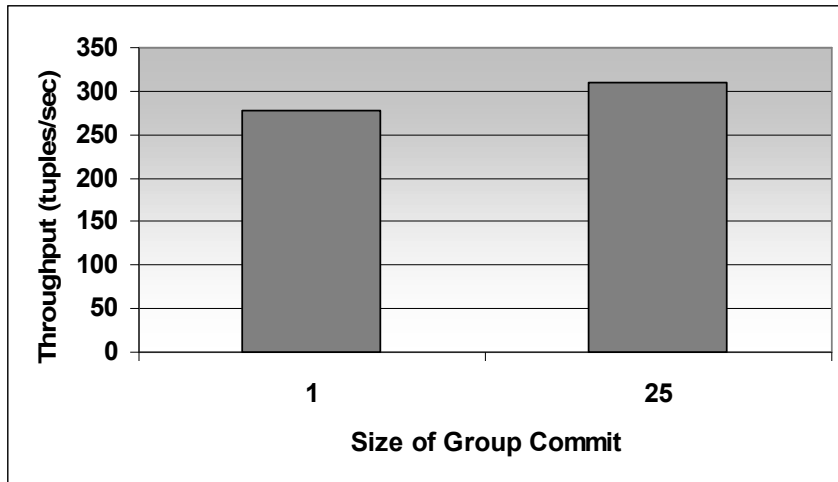
# Log IO -- transactions

## No Concurrent Transactions:

Insertions [300 000 inserts, 10 threads], e.g.,

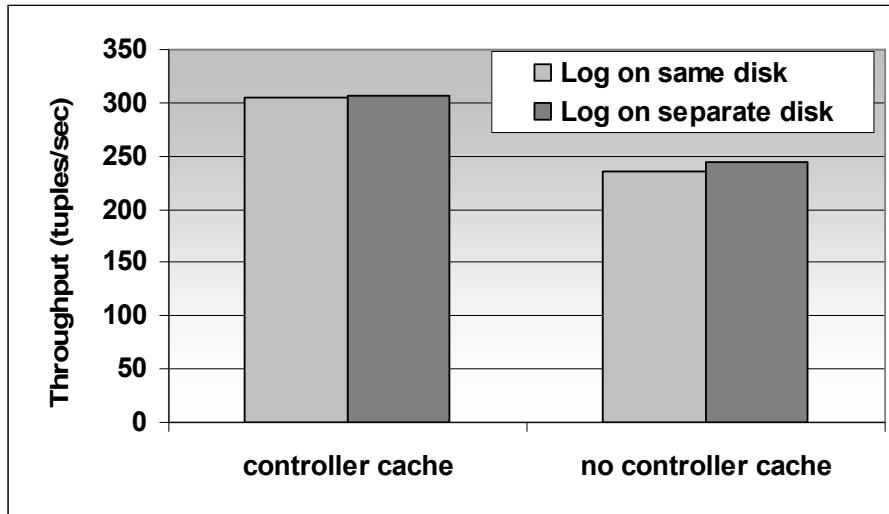
```
insert into lineitem values
(1,7760,401,1,17,28351.92,0.04,0.02,'N','O',
'1996-03-13','1996-02-12','1996-03-
22','DELIVER IN PERSON','TRUCK','blithely
regular ideas caj');
```

# Group Commits



- DB2 UDB v7.1 on Windows 2000
- Log records of many transactions are written together
  - Increases throughput by reducing the number of writes
  - at cost of increased minimum response time.

# Put the Log on a Separate Disk



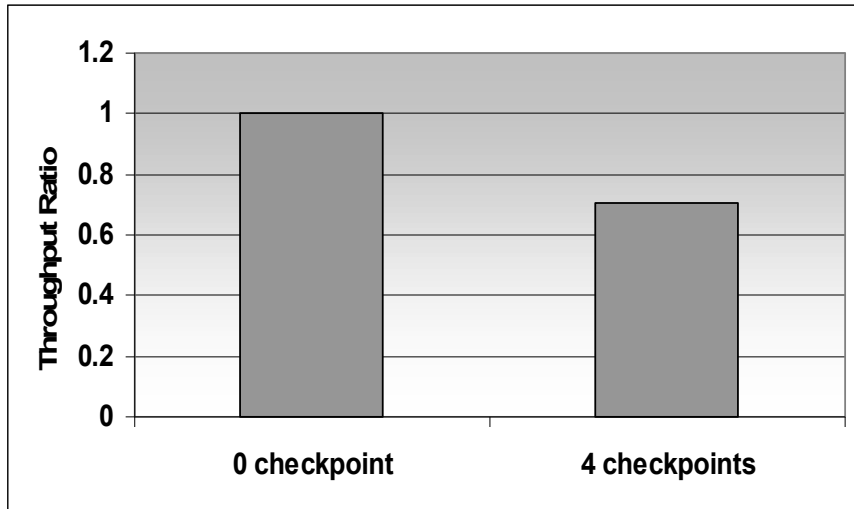
- DB2 UDB v7.1 on Windows 2000
- 5 % performance improvement if log is located on a different disk
- Controller cache hides negative impact
  - mid-range server, with Adaptec RAID controller (80Mb RAM) and 2x18Gb disk drives.

# Tuning Database Writes

- Dirty data is written to disk
  - When the number of dirty pages is greater than a given parameter (Oracle 8)
  - When the number of dirty pages crosses a given threshold (less than 3% of free pages in the database buffer for SQL Server 7)
  - When the log is full, a checkpoint is forced. This can have a significant impact on performance.

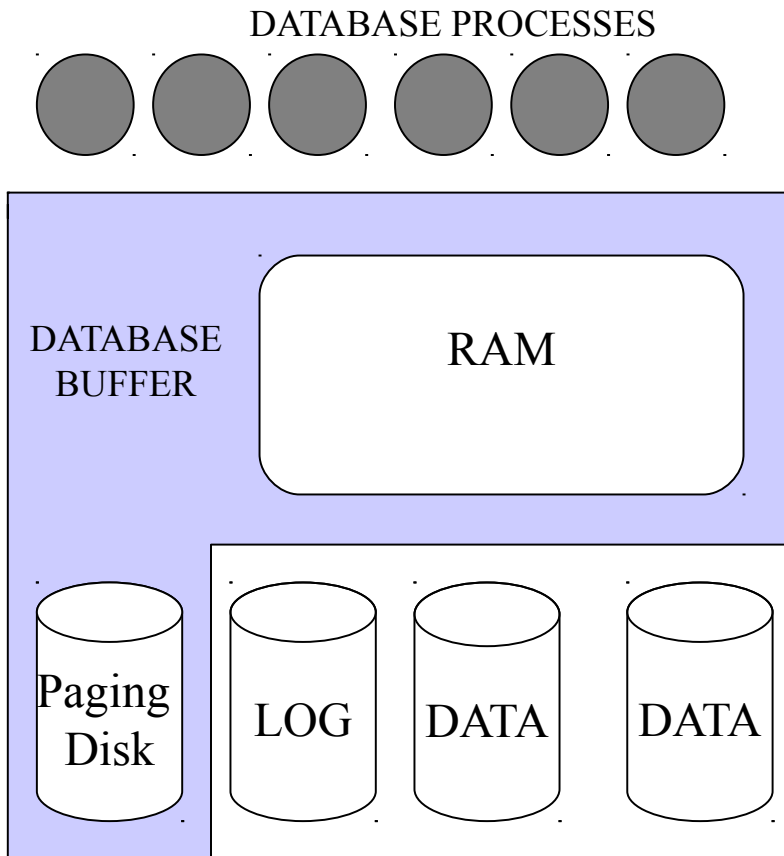


# Tune Checkpoint Intervals



- Oracle 8i on Windows 2000
- A checkpoint (partial flush of dirty pages to disk) occurs at regular intervals or when the log is full:
  - Impacts the performance of on-line processing
  - + Reduces the size of log
  - + Reduces time to recover from a crash

# Database Buffer Size



- Buffer too small, then hit ratio too small

*hit ratio* =

$$\frac{(\text{logical acc.} - \text{physical acc.})}{(\text{logical acc.})}$$

- Buffer too large, paging
- Recommended strategy: monitor hit ratio and increase buffer size until hit ratio flattens out. If there is still paging, then buy memory.

# Buffer Size -- data

## Settings:

```
employees (ssnum, name, lat, long, hundreds1,
 hundreds2) ;
```

```
clustered index c on employees(lat) ; (unused)
```

- 10 distinct values of lat and long, 100 distinct values of hundreds1 and hundreds2
- 20000000 rows (630 Mb);
- Warm Buffer
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000 RPM), Windows 2000.

# Buffer Size -- queries

## Queries:

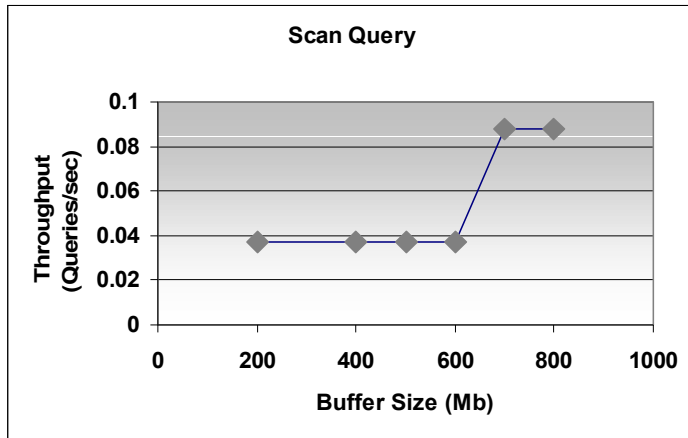
- Scan Query

```
select sum(long) from employees;
```

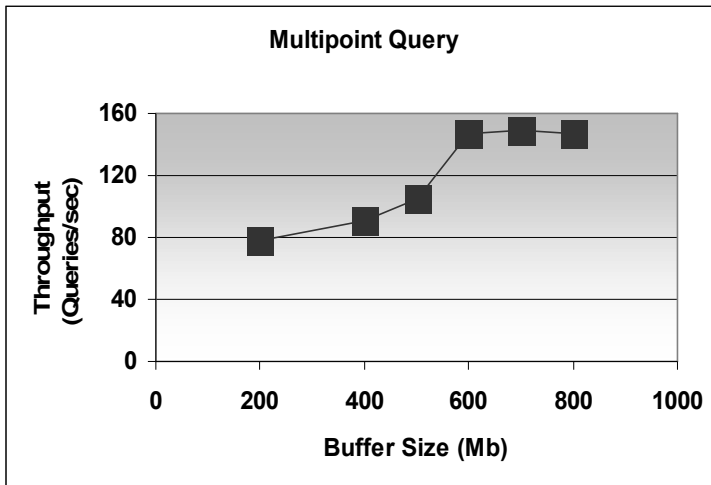
- Multipoint query

```
select * from employees where lat = ?;
```

# Database Buffer Size



- SQL Server 7 on Windows 2000
- Scan query:
  - LRU (least recently used) does badly when table spills to disk as Stonebraker observed 20 years ago.
- Multipoint query:
  - Throughput increases with buffer size until all data is accessed from RAM.



# Scan Performance -- data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
 L_LINENUMBER , L_QUANTITY, L_EXTENDEDPRICE ,
 L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
 L_SHIPDATE, L_COMMITDATE,
 L_RECEIPTDATE, L_SHIPINSTRUCT ,
 L_SHIPMODE , L_COMMENT);
```

- 600 000 rows
- Lineitem tuples are ~ 160 bytes long
- Cold Buffer
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

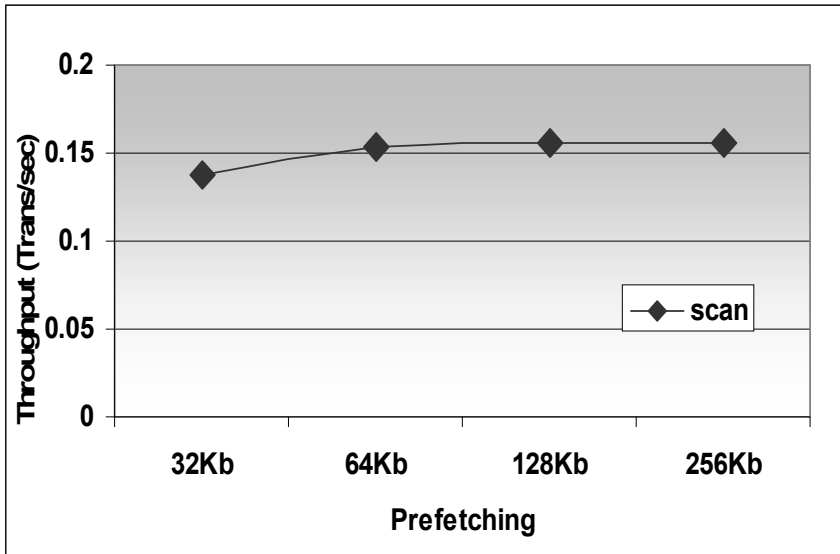
# Scan Performance -- queries

## Queries:

```
select avg(l_discount) from lineitem;
```

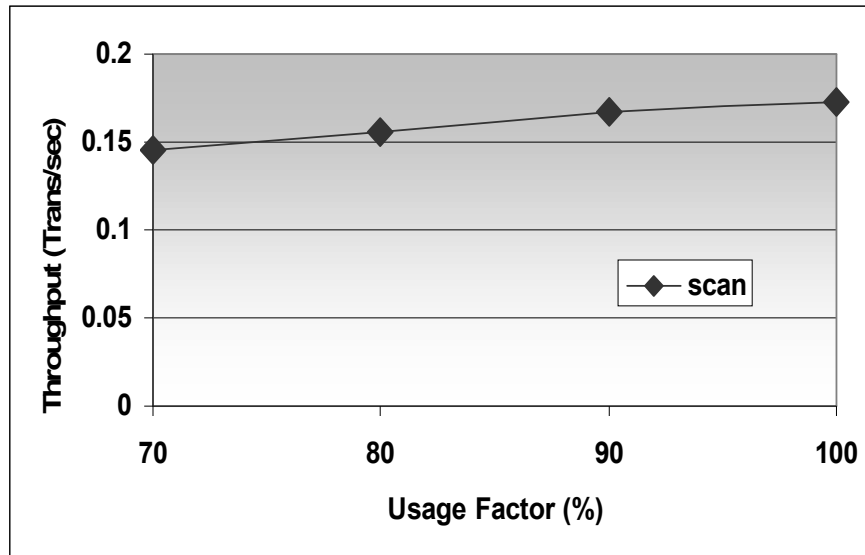
# Prefetching

- DB2 UDB v7.1 on Windows 2000
- Throughput increases up to a certain point when prefetching size increases.





# Usage Factor



- DB2 UDB v7.1 on Windows 2000
- Usage factor is the percentage of the page used by tuples and auxiliary data structures (the rest is reserved for future)
- Scan throughput increases with usage factor.

# RAID Levels

- RAID 0: striping (no redundancy)
- RAID 1: mirroring (2 disks)
- RAID 5: parity checking
  - Read: stripes read from multiple disks (in parallel)
  - Write: 2 reads + 2 writes
- RAID 10: striping and mirroring
- Software vs. Hardware RAID:
  - Software RAID: run on the server's CPU
  - Hardware RAID: run on the RAID controller's CPU

# Why 4 read/writes when updating a single stripe using RAID 5?

- Read old data stripe; read parity stripe (2 reads)
- XOR old data stripe with replacing one.
- Take result of XOR and XOR with parity stripe.
- Write new data stripe and new parity stripe (2 writes).

# RAID Levels -- data

## Settings:

```
accounts (number, branchnum, balance);
create clustered index c on accounts(number);
```

- 100000 rows
- Cold Buffer
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# RAID Levels -- transactions

## No Concurrent Transactions:

- Read Intensive:

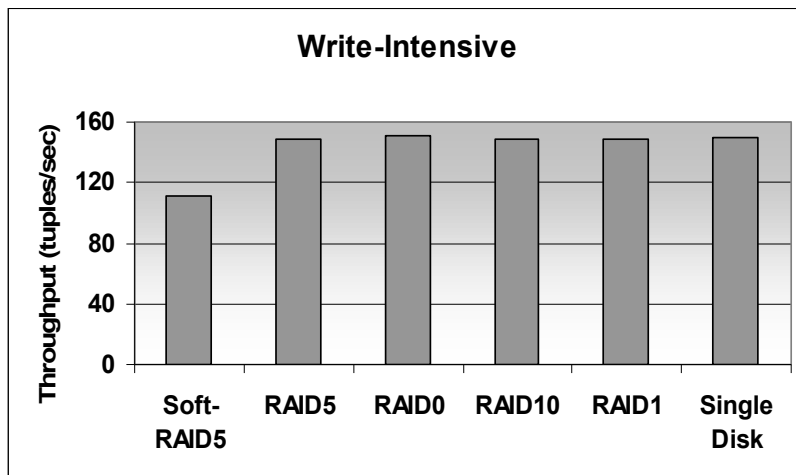
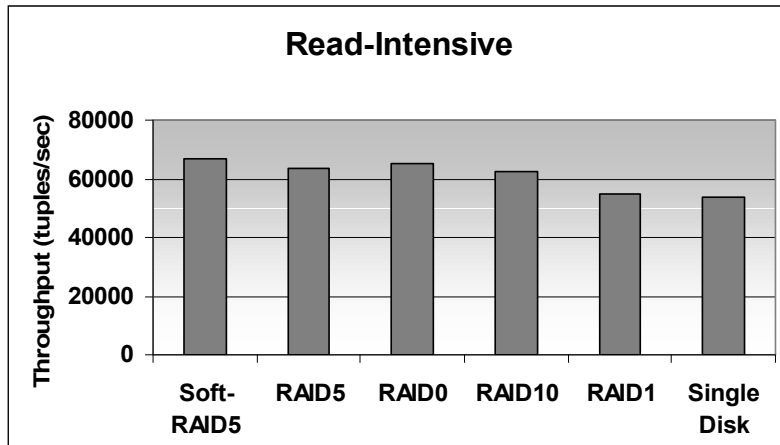
```
select avg(balance) from accounts;
```

- Write Intensive, e.g. typical insert:

```
insert into accounts values (690466,6840,2272.76);
```

Writes are uniformly distributed.

# RAID Levels



- SQL Server7 on Windows 2000 (SoftRAID means striping/parity at host)
- Read-Intensive:
  - Using multiple disks (RAID0, RAID 10, RAID5) increases throughput significantly.
- Write-Intensive:
  - Without cache, RAID 5 suffers. With cache, it is ok.

# RAID Levels

- Log File
  - RAID 1 is appropriate
    - Fault tolerance with high write throughput. Writes are synchronous and sequential. No benefits in striping.
- Temporary Files
  - RAID 0 is appropriate.
    - No fault tolerance. High throughput.
- Data and Index Files
  - RAID 5 is best suited for read intensive apps or if the RAID controller cache is effective enough.
  - RAID 10 is best suited for write intensive apps.

# Controller Prefetching no, Write-back yes.

- Read-ahead:
  - Prefetching at the disk controller level.
  - No information on access pattern.
  - Better to let database management system do it.
- Write-back vs. write through:
  - Write back: transfer terminated as soon as data is written to cache.
    - Batteries to guarantee write back in case of power failure
  - Write through: transfer terminated as soon as data is written to disk.



# SCSI Controller Cache -- data

## Settings:

```
employees (ssnum, name, lat, long, hundreds1,
 hundreds2) ;
```

```
create clustered index c on
 employees (hundreds2) ;
```

- Employees table partitioned over two disks; Log on a separate disk; same controller (same channel).
- 200 000 rows per table
- Database buffer size limited to 400 Mb.
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# SCSI (not disk) Controller Cache

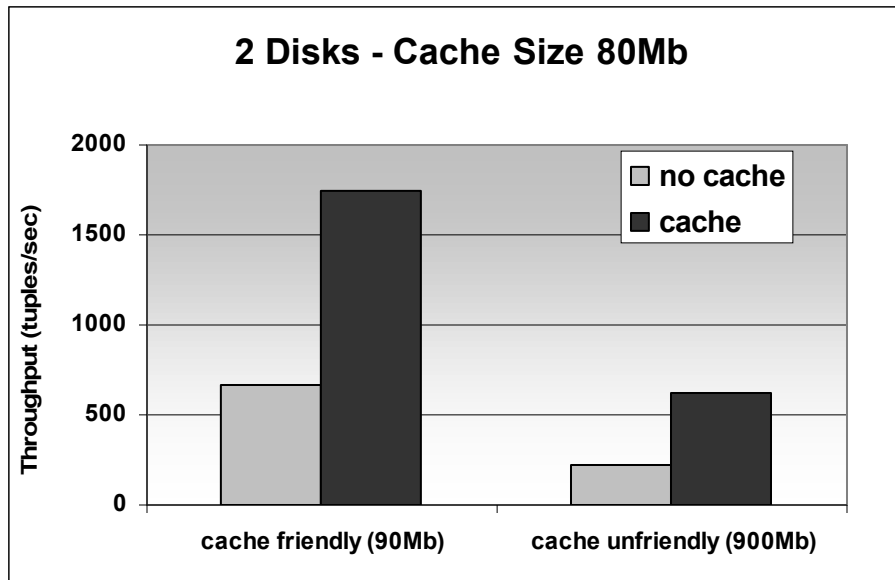
## -- transactions

### No Concurrent Transactions:

```
update employees set lat = long, long = lat
where hundreds2 = ?;
```

- cache friendly: update of 20,000 rows (~90Mb)
- cache unfriendly: update of 200,000 rows (~900Mb)

# SCSI Controller Cache



- SQL Server 7 on Windows 2000.
- Adaptec ServerRaid controller:
  - 80 Mb RAM
  - Write-back mode
- Updates
- Controller cache increases throughput whether operation is cache friendly or not.
  - Efficient replacement policy!

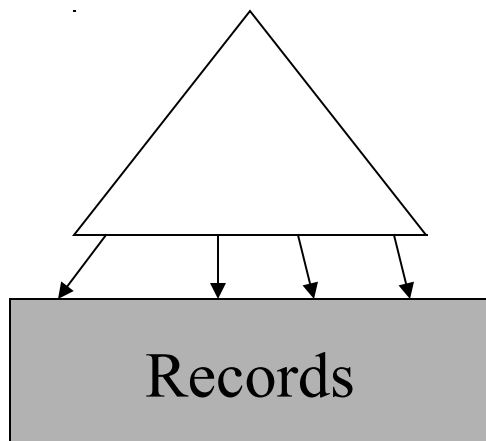
# Index Tuning

- Index issues
  - Indexes may be better or worse than scans
  - Multi-table joins that run on for hours, because the wrong indexes are defined
  - Concurrency control bottlenecks
  - Indexes that are maintained and never used

# Clustered / Non clustered index

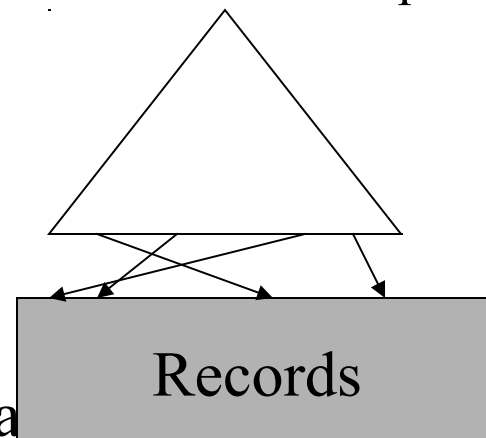
- Clustered index  
(primary index)

- A clustered index on attribute X co-locates records whose X values are *near* to one another.



- Non-clustered index  
(secondary index)

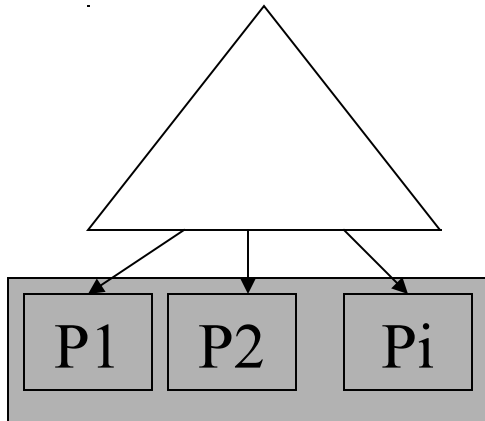
- A non clustered index does not constrain table organization.
- There might be several non-clustered indexes per table.



# Dense / Sparse Index

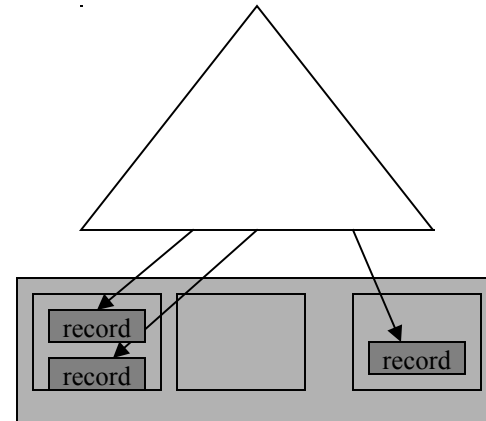
- Sparse index

- Pointers are associated to pages



- Dense index

- Pointers are associated to records
- Non clustered indexes are dense



# Index Implementations in some major DBMS

- SQL Server
  - B+-Tree data structure
  - Clustered indexes are sparse
  - Indexes maintained as updates/insertions/deletes are performed
- DB2
  - B+-Tree data structure, spatial extender for R-tree
  - Clustered indexes are dense
  - Explicit command for index reorganization
- Oracle
  - B+-tree, hash, bitmap, spatial extender for R-Tree
  - clustered index
    - Index organized table (unique/clustered)
    - Clusters used when creating tables.
- TimesTen (Main-memory DBMS)
  - T-tree

# Types of Queries

- Point Query

```
SELECT balance
FROM accounts
WHERE number = 1023;
```

- Multipoint Query

```
SELECT balance
FROM accounts
WHERE branchnum = 100;
```

- Range Query

```
SELECT number
FROM accounts
WHERE balance > 10000 and
balance <= 20000;
```

- Prefix Match Query

```
SELECT *
FROM employees
WHERE name = 'J*';
```



# More Types of Queries

- Extremal Query

```
SELECT *
FROM accounts
WHERE balance =
 max(select balance from accounts)
```

- Ordering Query

```
SELECT *
FROM accounts
ORDER BY balance;
```

- Grouping Query

```
SELECT branchnum, avg(balance)
FROM accounts
GROUP BY branchnum;
```

- Join Query

```
SELECT distinct branch.adresse
FROM accounts, branch
WHERE
 accounts.branchnum =
 branch.number
and accounts.balance > 10000;
```

# Index Tuning -- data

## Settings:

```
employees (ssnum, name, lat, long, hundreds1,
 hundreds2);
```

```
clustered index c on employees (hundreds1)
 with fillfactor = 100;
```

```
nonclustered index nc on employees (hundreds2);
```

```
index nc3 on employees (ssnum, name, hundreds2);
```

```
index nc4 on employees (lat, ssnum, name);
```

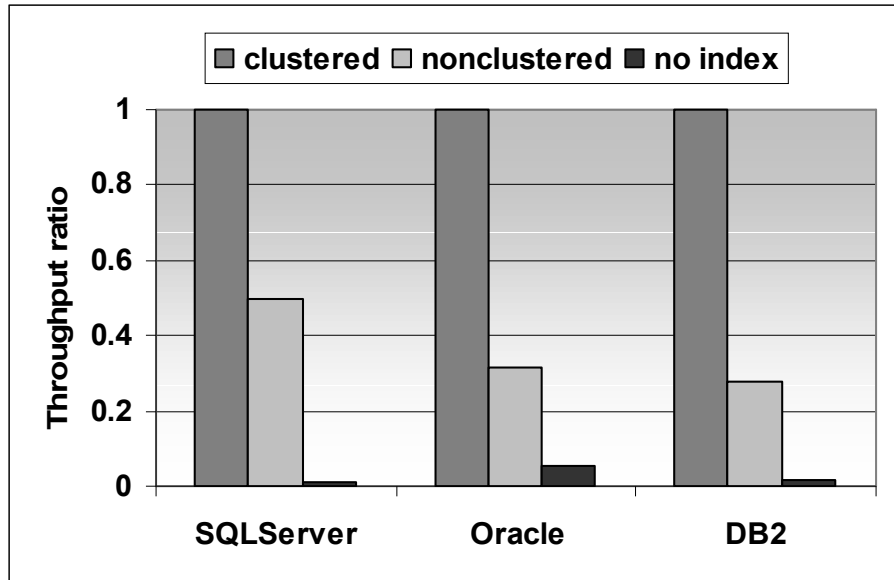
- 1000000 rows ; Cold buffer
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# Index Tuning -- operations

## Operations:

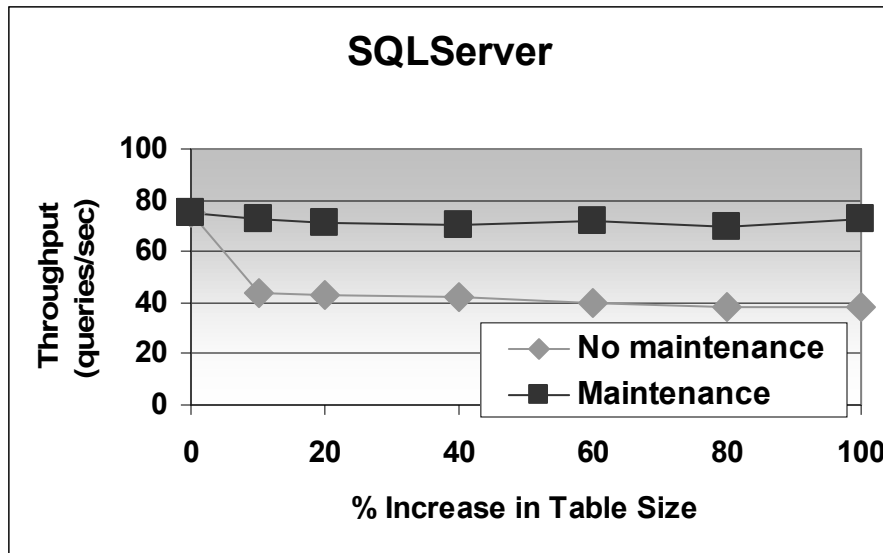
- Update:  
`update employees set name = 'XXX' where ssnun = ?;`
- Insert:  
`insert into employees values  
(1003505, 'polo94064', 97.48, 84.03, 4700.55, 3987.2);`
- Multipoint query:  
`select * from employees where hundreds1= ?;  
select * from employees where hundreds2= ?;`
- Covered query:  
`select ssnun, name, lat from employees;`
- Range Query:  
`select * from employees where long between ? and ?;`
- Point Query:  
`select * from employees where ssnun = ?`

# Clustered Index



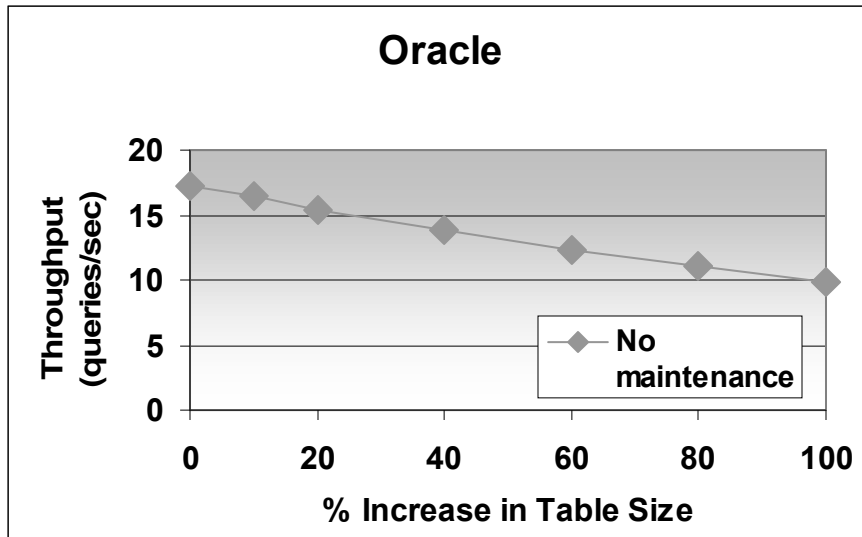
- Multipoint query that returns 100 records out of 1000000.
- Cold buffer
- Clustered index is twice as fast as non-clustered index and orders of magnitude faster than a scan.

# Index “Face Lifts”



- Index is created with fillfactor = 100.
- Insertions cause page splits and extra I/O for each query
- Maintenance consists in dropping and recreating the index
- With maintenance performance is constant while performance degrades significantly if no maintenance is performed.

# Index Maintenance

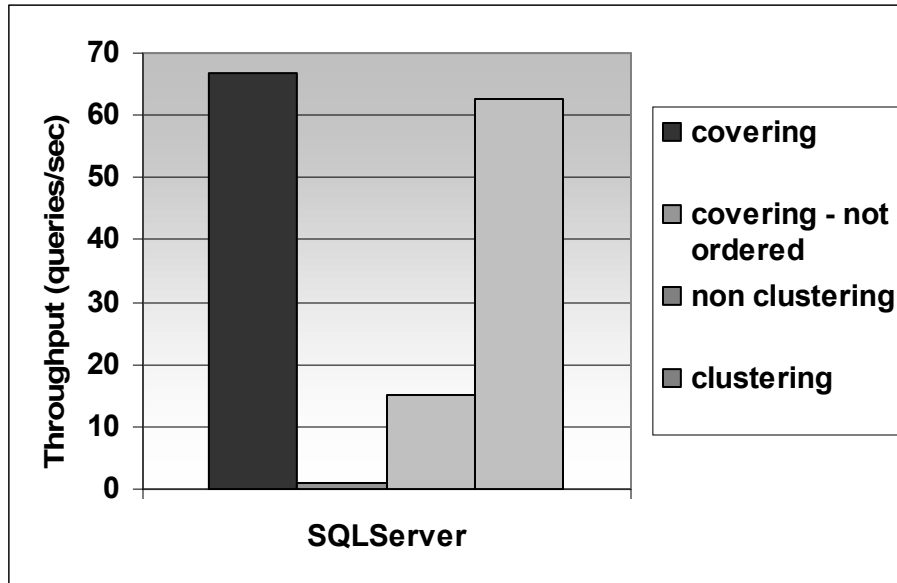


- In Oracle, clustered index are approximated by an index defined on a clustered table
- No automatic physical reorganization
- Index defined with `pctfree = 0`
- Overflow pages cause performance degradation

# Covering Index - defined

- Select name from employee where department = “marketing”
- Good covering index would be on (department, name)
- Index on (name, department) less useful.
- Index on department alone moderately useful.

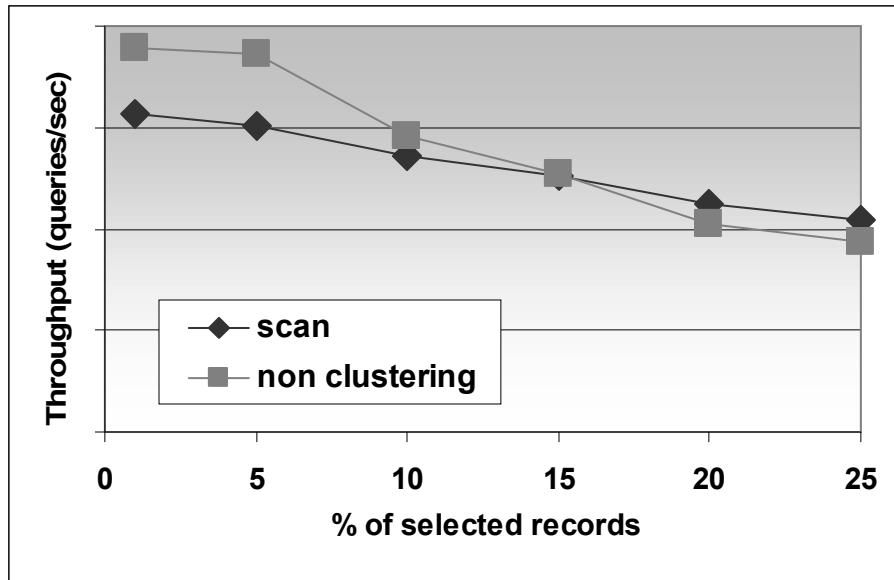
# Covering Index - impact



- Covering index performs better than clustering index when first attributes of index are in the where clause and last attributes in the select.
- When attributes are not in order then performance is much worse.

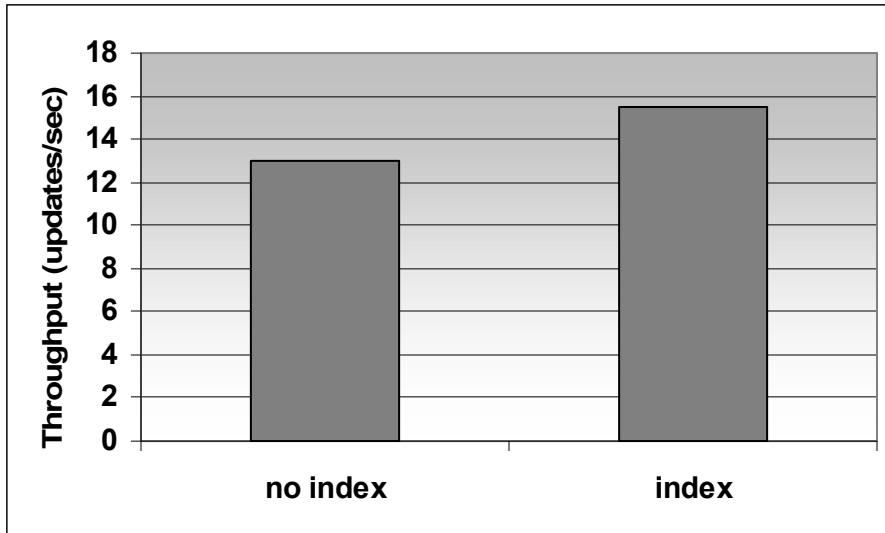


# Scan Can Sometimes Win



- IBM DB2 v7.1 on Windows 2000
- Range Query
- If a query retrieves 10% of the records or more, scanning is often better than using a non-clustering non-covering index. Crossover  $> 10\%$  when records are large or table is fragmented on disk – scan cost increases.

# Index on Small Tables



- Small table: 100 records, i.e., a few pages.
- Two concurrent processes perform updates (each process works for 10ms before it commits)
- No index: the table is scanned for each update. No concurrent updates.
- A clustered index allows to take advantage of row locking.

# Bitmap vs. Hash vs. B+-Tree

## Settings:

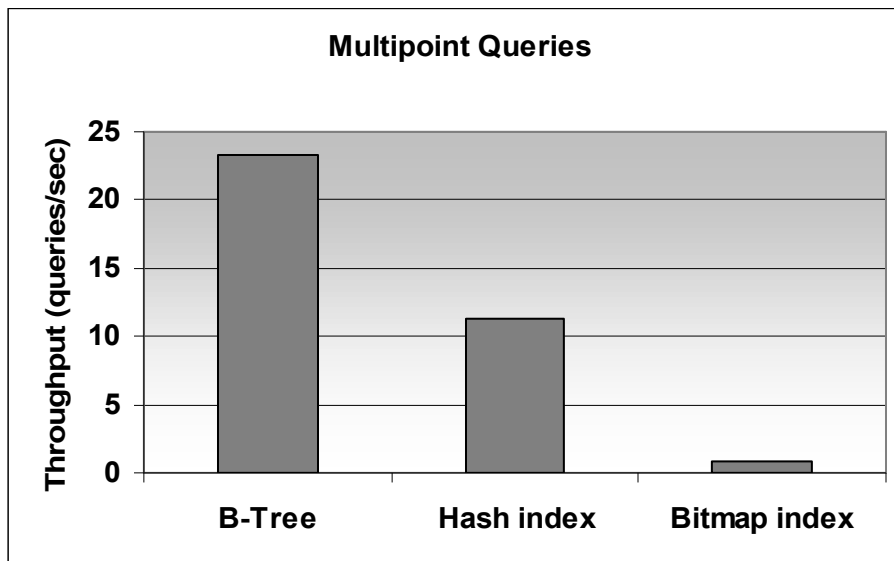
```
employees (ssnum, name, lat, long, hundreds1,
 hundreds2);
create cluster c_hundreds (hundreds2 number(8)) PCTFREE 0;
create cluster c_ssnum(ssnum integer) PCTFREE 0 size 60;

create cluster c_hundreds(hundreds2 number(8)) PCTFREE 0 HASHKEYS
 1000 size 600;
create cluster c_ssnum(ssnum integer) PCTFREE 0 HASHKEYS 1000000
 SIZE 60;

create bitmap index b on employees (hundreds2);
create bitmap index b2 on employees (ssnum);
```

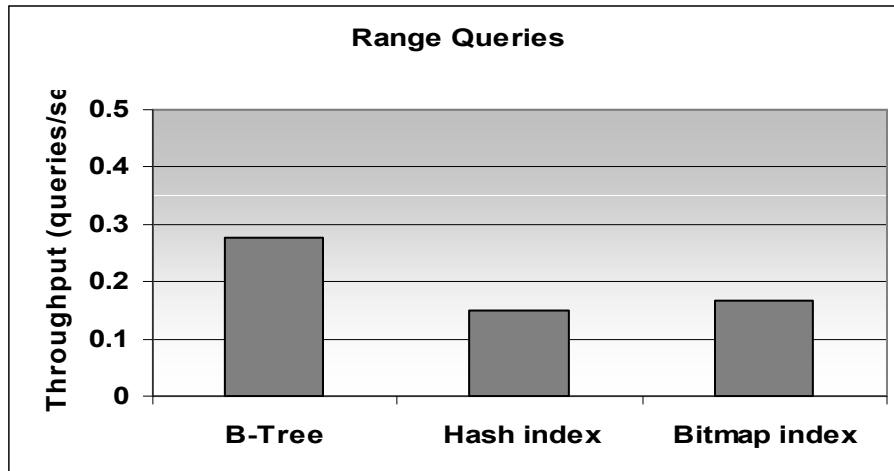
- 1000000 rows ; Cold buffer
- Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.

# Multipoint query: B-Tree, Hash Tree, Bitmap

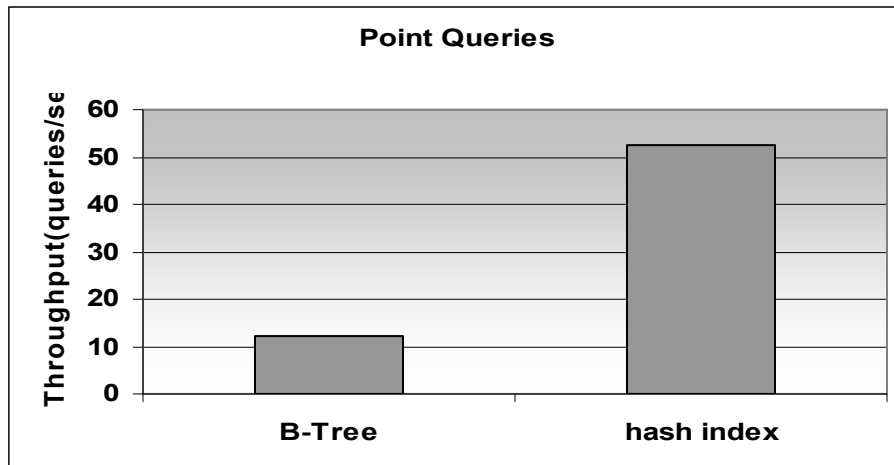


- There is an overflow chain in a hash index
- In a clustered B-Tree index records are on contiguous pages.
- Bitmap is proportional to size of table and non-clustered for record access.

# B-Tree, Hash Tree, Bitmap



- Hash indexes don't help when evaluating range queries



- Hash index outperforms B-tree on point queries

# Tuning Relational Systems

- Schema Tuning
  - Denormalization
- Query Tuning
  - Query rewriting
  - Materialized views

# Denormalizing -- data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
 L_LINENUMBER, L_QUANTITY, L_EXTENDEDPRICE ,
 L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
 L_SHIPDATE, L_COMMITDATE,
 L_RECEIPTDATE, L_SHIPINSTRUCT ,
 L_SHIPMODE , L_COMMENT);

region(R_REGIONKEY, R_NAME, R_COMMENT);

nation(N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT,);

supplier(S_SUPPKEY, S_NAME, S_ADDRESS, S_NATIONKEY,
S_PHONE, S_ACCTBAL, S_COMMENT);
– 600000 rows in lineitem, 25 nations, 5 regions, 500 suppliers
```

# Denormalizing -- transactions

```
lineitemdenormalized (L_ORDERKEY, L_PARTKEY ,
 L_SUPPKEY, L_LINENUMBER, L_QUANTITY,
 L_EXTENDEDPRICE ,
 L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
 L_SHIPDATE, L_COMMITDATE,
 L_RECEIPTDATE, L_SHIPINSTRUCT ,
 L_SHIPMODE , L_COMMENT, L_REGIONNAME) ;
```

- 600000 rows in lineitemdenormalized
- Cold Buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.



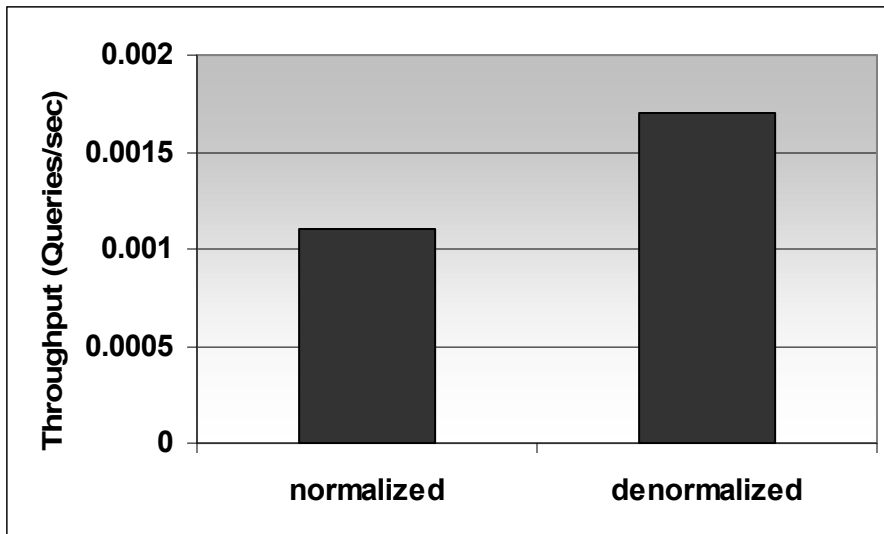
# Queries on Normalized vs. Denormalized Schemas

## Queries:

```
select L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER, L_QUANTITY,
 L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS,
 L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE,
 L_COMMENT, R_NAME
from LINEITEM, REGION, SUPPLIER, NATION
where
L_SUPPKEY = S_SUPPKEY
and S_NATIONKEY = N_NATIONKEY
and N_REGIONKEY = R_REGIONKEY
and R_NAME = 'EUROPE';
```

```
select L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER, L_QUANTITY,
 L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS,
 L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE,
 L_COMMENT, L_REGIONNAME
from LINEITEMDENORMALIZED
where L_REGIONNAME = 'EUROPE';
```

# Denormalization



- TPC-H schema
- Query: find all lineitems whose supplier is in Europe.
- With a normalized schema this query is a 4-way join.
- If we denormalize lineitem and add the name of the region for each lineitem (foreign key denormalization) throughput improves 30%

# Queries

## Settings:

```
employee(ssnum, name, dept, salary, numfriends);
student(ssnum, name, course, grade);
techdept(dept, manager, location);
```

```
clustered index i1 on employee (ssnum);
nonclustered index i2 on employee (name);
nonclustered index i3 on employee (dept);

clustered index i4 on student (ssnum);
nonclustered index i5 on student (name);

clustered index i6 on techdept (dept);
```

- 100000 rows in employee, 100000 students, 10 departments; Cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Queries – View on Join

## View Techlocation:

```
create view techlocation as
select ssnnum, techdept.dept, location
from employee, techdept
where employee.dept = techdept.dept;
```

## Queries:

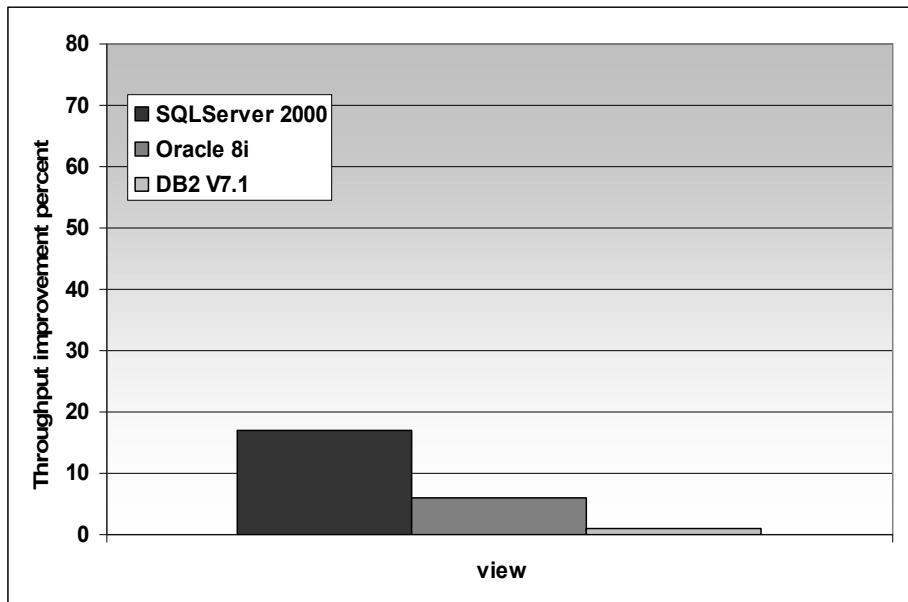
### – Original:

```
select dept from techlocation where ssnnum = ?;
```

### – Rewritten:

```
select dept from employee where ssnnum = ?;
```

# Query Rewriting - Views



- All systems expand the selection on a view into a join
- The difference between a plain selection and a join (on a primary key-foreign key) followed by a projection is greater on SQL Server than on Oracle and DB2 v7.1.

# Queries – Correlated Subqueries

## Queries:

- Original:

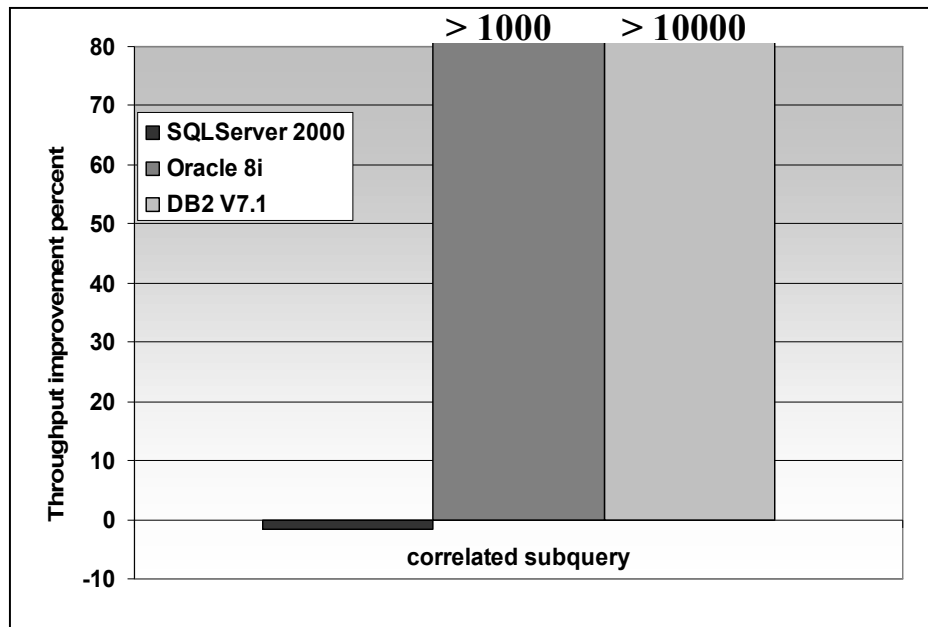
```
select ssnum
from employee e1
where salary =
 (select max(salary)
 from employee e2
 where e2.dept = e1.dept);
```

- Rewritten:

```
select max(salary) as bigsalary, dept
into TEMP
from employee group by dept;

select ssnum
from employee, TEMP
where salary = bigsalary
and employee.dept = temp.dept;
```

# Query Rewriting – Correlated Subqueries



- SQL Server 2000 does a good job at handling the correlated subqueries (a hash join is used as opposed to a nested loop between query blocks)
  - The techniques implemented in SQL Server 2000 are described in “Orthogonal Optimization of Subqueries and Aggregates” by C.Galindo-Legaria and M.Joshi, SIGMOD 2001.

# Aggregate Maintenance -- data

## Settings:

```
orders(ordernum, itemnum, quantity, purchaser, vendor);
create clustered index i_order on orders(itemnum);

store(vendor, name);

item(itemnum, price);
create clustered index i_item on item(itemnum);

vendorOutstanding(vendor, amount);

storeOutstanding(store, amount);
```

- 1000000 orders, 10000 stores, 400000 items; Cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.



# Aggregate Maintenance -- triggers

## Triggers for Aggregate Maintenance

```
create trigger updateVendorOutstanding on orders for insert as
update vendorOutstanding
set amount =
 (select vendorOutstanding.amount+sum(inserted.quantity*item.price)
 from inserted,item
 where inserted.itemnum = item.itemnum
)
where vendor = (select vendor from inserted) ;

create trigger updateStoreOutstanding on orders for insert as
update storeOutstanding
set amount =
 (select storeOutstanding.amount+sum(inserted.quantity*item.price)
 from inserted,item
 where inserted.itemnum = item.itemnum
)
where store = (select store.name from inserted, store
 where inserted.vendor = store.vendor) ;
```

# Aggregate Maintenance -- transactions

## Concurrent Transactions:

- Insertions

```
insert into orders values
(1000350,7825,562,'xxxxxx6944','vendor4');
```

- Queries (first without, then with redundant tables)

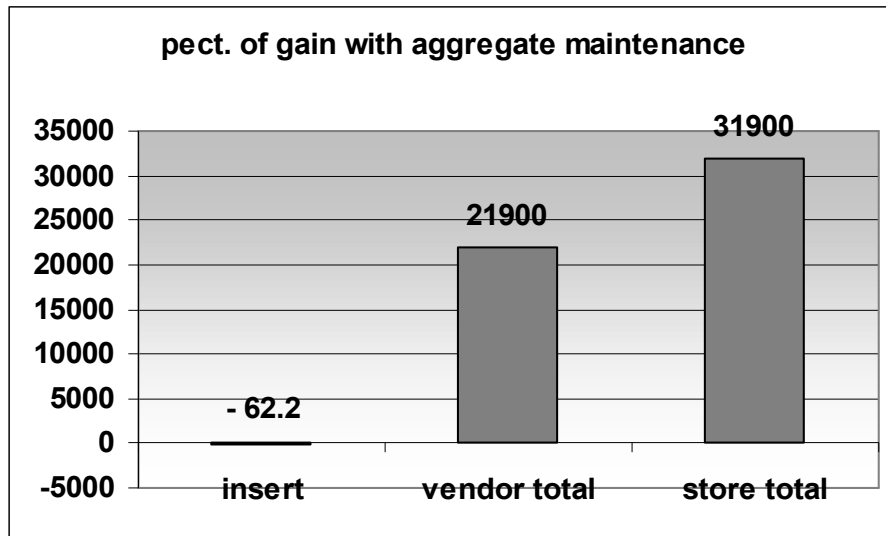
```
select orders.vendor, sum(orders.quantity*item.price)
from orders,item
where orders.itemnum = item.itemnum
group by orders.vendor;
```

```
vs. select * from vendorOutstanding;
```

```
select store.name, sum(orders.quantity*item.price)
from orders,item, store
where orders.itemnum = item.itemnum
and orders.vendor = store.vendor
group by store.name;
```

```
vs. select * from storeOutstanding;
```

# Aggregate Maintenance



- SQLServer 2000 on Windows 2000
- Using triggers for view maintenance
- If queries frequent or important, then aggregate maintenance is good.

# Superlinearity -- data

## Settings:

```
sales(id, itemid, customerid, storeid, amount, quantity);
item (itemid);
customer (customerid);
store (storeid);
```

A sale is successful if all foreign keys are present.

```
successfalsales(id, itemid, customerid, storeid, amount,
quantity);
```

```
unsuccessfalsales(id, itemid, customerid, storeid, amount,
quantity);
```

```
tempsales(id, itemid, customerid, storeid, amount, quantity);
```

# Superlinearity -- indexes

## Settings (non-clustering, dense indexes):

```
index s1 on item(itemid);
```

```
index s2 on customer(customerid);
```

```
index s3 on store(storeid);
```

```
index succ on successfalsales(id);
```

- 1000000 sales, 400000 customers, 40000 items, 1000 stores
- Cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Superlinearity -- queries

## Queries:

- Insert/create indexdelete

```
insert into successfalsales
select sales.id, sales.itemid, sales.customerid,
 sales.storeid, sales.amount, sales.quantity
from sales, item, customer, store
where sales.itemid = item.itemid
and sales.customerid = customer.customerid
and sales.storeid = store.storeid;
```

```
insert into unsuccessfulsales
select * from sales;
go
delete from unsuccessfulsales
where id in (select id from successfalsales)
```

# Superlinearity -- batch queries

## Queries:

### – Small batches

```
DECLARE @Nlow INT;
DECLARE @Nhigh INT;
DECLARE @INCR INT;
set @INCR = 100000
set @NLow = 0
set @Nhigh = @INCR
WHILE (@NLow <= 500000)
BEGIN
 insert into tempsales
 select * from sales
 where id between @NLow and @Nhigh
 set @Nlow = @Nlow + @INCR
 set @Nhigh = @Nhigh + @INCR
 delete from tempsales
 where id in (select id from successfalsales);
 insert into unsuccessfulsales
 select * from tempsales;
 delete from tempsales;
END
```

# Superlinearity -- outer join

## Queries:

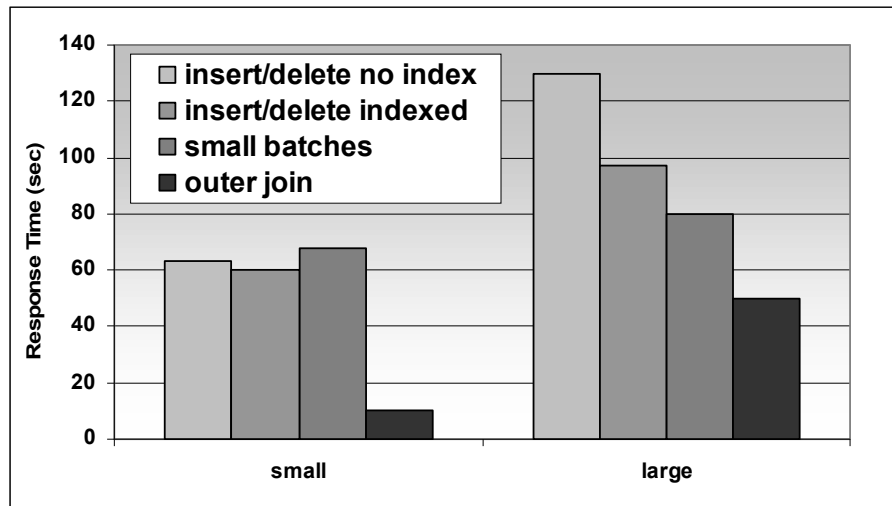
### – outerjoin

```
insert into successfalsales
select sales.id, item.itemid, customer.customerid, store.storeid, sales.amount,
 sales.quantity
from
((sales left outer join item on sales.itemid = item.itemid)
left outer join customer on sales.customerid = customer.customerid)
left outer join store on sales.storeid = store.storeid;

insert into unsuccessfulsales
select *
from successfalsales
where itemid is null
or customerid is null
or storeid is null;
go
delete from successfalsales
where itemid is null
or customerid is null
or storeid is null
```



# Circumventing Superlinearity



- SQL Server 2000
- Outer join achieves the best response time.
- Small batches do not help because overhead of crossing the application interface is higher than the benefit of joining with smaller tables.

# Tuning the Application Interface

- 4GL
  - Power++, Visual basic
- Programming language + Call Level Interface
  - ODBC: Open DataBase Connectivity
  - JDBC: Java based API
  - OCI (C++/Oracle), CLI (C++/ DB2), Perl/DBI
- In the following experiments, the client program is located on the database server site. Overhead is due to crossing the application interface.

# Looping can hurt -- data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
L_LINENUMBER, L_QUANTITY, L_EXTENDEDPRICE ,
L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
L_SHIPDATE, L_COMMITDATE,
L_RECEIPTDATE, L_SHIPINSTRUCT ,
L_SHIPMODE , L_COMMENT);
```

- 600 000 rows; warm buffer.
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

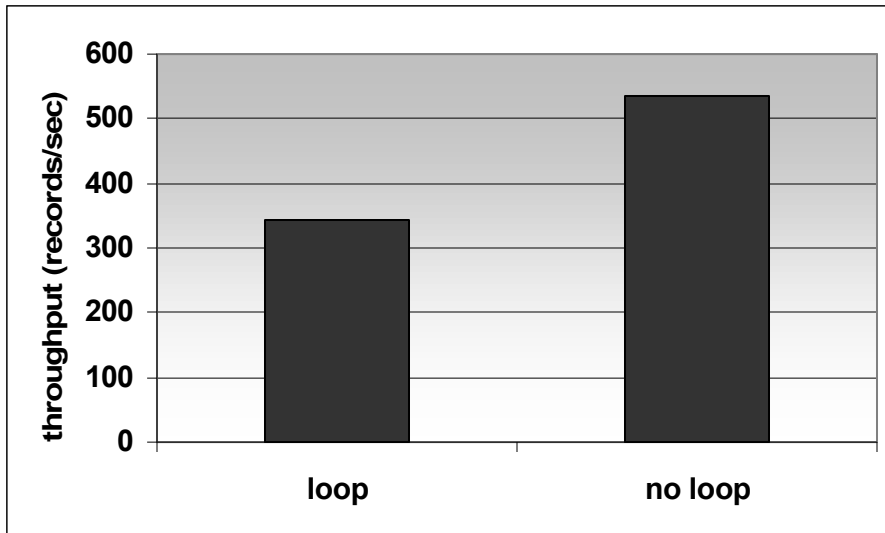
# Looping can hurt -- queries

- Queries:
  - No loop:

```
sqlStmt = "select * from lineitem where l_partkey <= 200;"
odbc->prepareStmt(sqlStmt);
odbc->execPrepared(sqlStmt);
```
  - Loop:

```
sqlStmt = "select * from lineitem where l_partkey = ?;"
odbc->prepareStmt(sqlStmt);
for (int i=1; i<100; i++)
{
 odbc->bindParameter(1, SQL_INTEGER, i);
 odbc->execPrepared(sqlStmt);
}
```

# Looping can Hurt



- SQL Server 2000 on Windows 2000
- Crossing the application interface has a significant impact on performance.
- Why would a programmer use a loop instead of relying on set-oriented operations: object-orientation?

# Cursors are Death -- data

## Settings:

```
employees (ssnum, name, lat, long, hundreds1,
hundreds2) ;
```

- 100000 rows ; Cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Cursors are Death -- queries

## Queries:

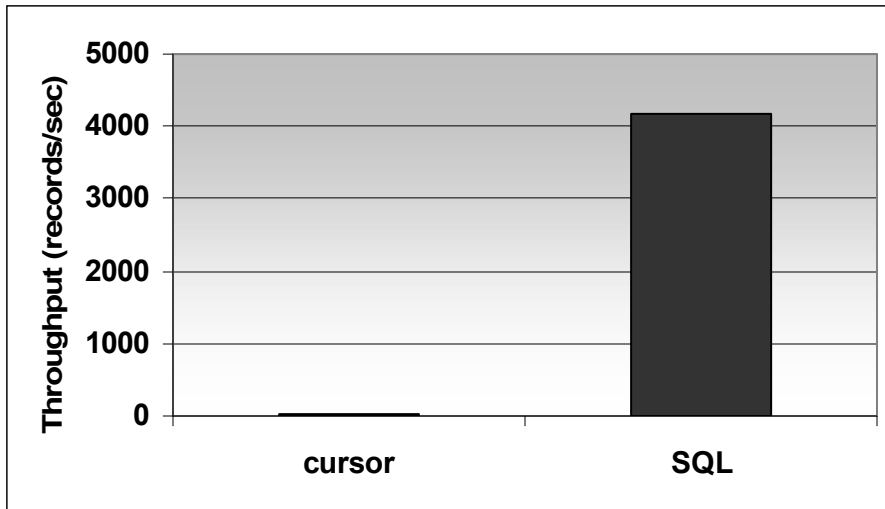
- No cursor

```
select * from employees;
```

- Cursor

```
DECLARE d_cursor CURSOR FOR select * from employees;
OPEN d_cursor
while (@@FETCH_STATUS = 0)
BEGIN
 FETCH NEXT from d_cursor
END
CLOSE d_cursor
go
```

# Cursors are Death



- SQL Server 2000 on Windows 2000
- Response time is a few seconds with a SQL query and more than an hour iterating over a cursor.



# Retrieve Needed Columns Only - data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
 L_LINENUMBER, L_QUANTITY, L_EXTENDEDPRICE ,
 L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
 L_SHIPDATE, L_COMMITDATE,
 L_RECEIPTDATE, L_SHIPINSTRUCT ,
 L_SHIPMODE , L_COMMENT);
```

```
create index i_nc_lineitem on lineitem (l_orderkey,
 l_partkey, l_suppkey, l_shipdate, l_commitdate);
```

- 600 000 rows; warm buffer.
- Lineitem records are ~ 10 bytes long
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Retrieve Needed Columns Only - queries

## Queries:

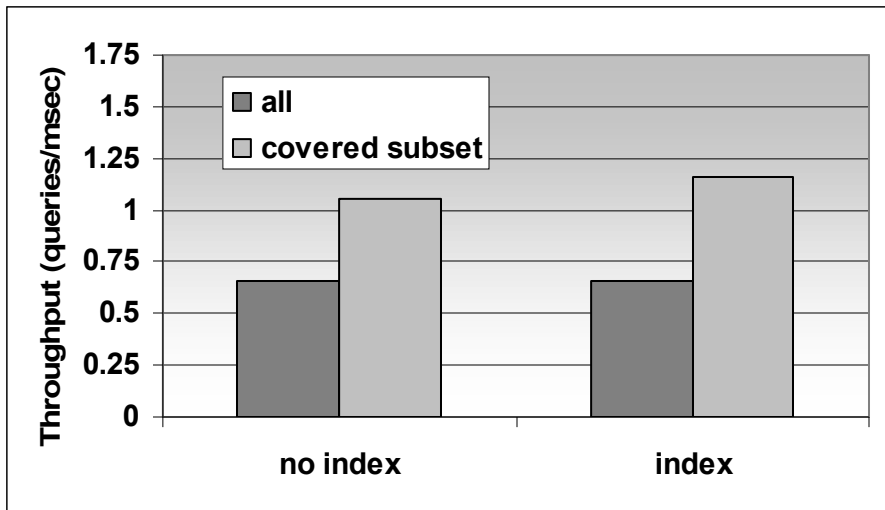
- All

```
Select * from lineitem;
```

- Covered subset

```
Select l_orderkey, l_partkey, l_suppkey, l_shipdate,
 l_commitdate from lineitem;
```

# Retrieve Needed Columns Only



Experiment performed on  
Oracle8iEE on Windows 2000.

- Avoid transferring unnecessary data
- May enable use of a covering index.
- In the experiment the subset contains  $\frac{1}{4}$  of the attributes.
  - Reducing the amount of data that crosses the application interface yields significant performance improvement.

# Bulk Loading Data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY,
L_LINENUMBER, L_QUANTITY, L_EXTENDEDPRICE ,
L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
L_SHIPDATE, L_COMMITDATE,
L_RECEIPTDATE, L_SHIPINSTRUCT ,
L_SHIPMODE , L_COMMENT);
```

- Initially the table is empty; 600 000 rows to be inserted (138Mb)
- Table sits on one disk. No constraint, index is defined.
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Bulk Loading Queries

## Oracle 8i

```
sqlldr directpath=true control=load_lineitem.ctl data=E:\Data\lineitem.tbl
```

```
load data
```

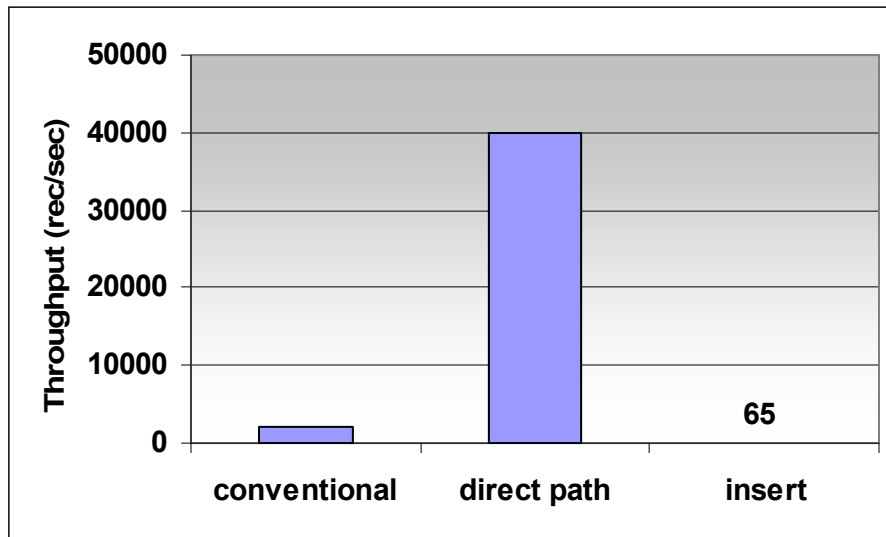
```
infile "lineitem.tbl"
```

```
into table LINEITEM append
```

```
fields terminated by '|'
```

```
(
 L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER,
 L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX,
 L_RETURNFLAG, L_LINESTATUS, L_SHIPDATE DATE "YYYY-MM-DD",
 L_COMMITDATE DATE "YYYY-MM-DD", L_RECEIPTDATE DATE
 "YYYY-MM-DD", L_SHIPINSTRUCT, L_SHIPMODE, L_COMMENT
)
```

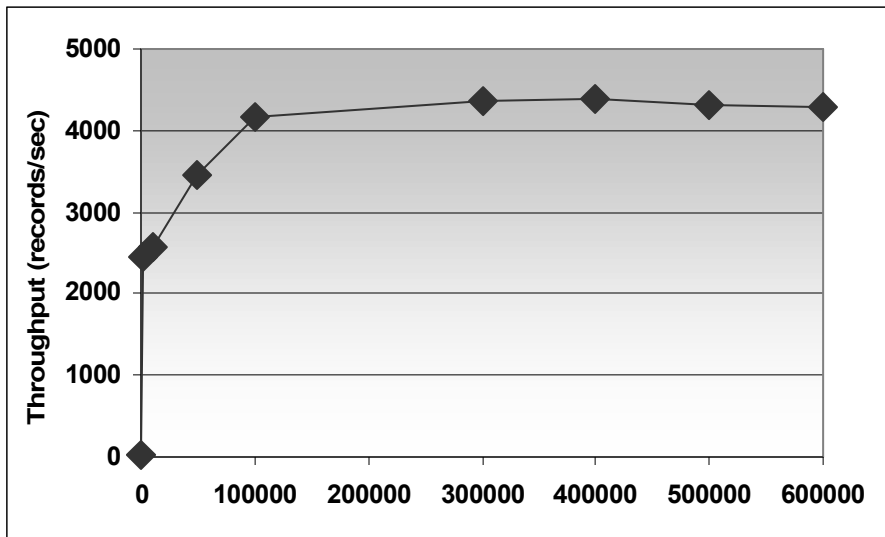
# Direct Path



- Direct path loading bypasses the query engine and the storage manager. It is orders of magnitude faster than for conventional bulk load (commit every 100 records) and inserts (commit for each record).

Experiment performed on Oracle8iEE on Windows 2000.

# Batch Size



Experiment performed on  
SQL Server 2000  
on Windows 2000.

- Throughput increases steadily when the batch size increases to 100000 records. Throughput remains constant afterwards.
- Trade-off between performance and amount of data that has to be reloaded in case of problem.

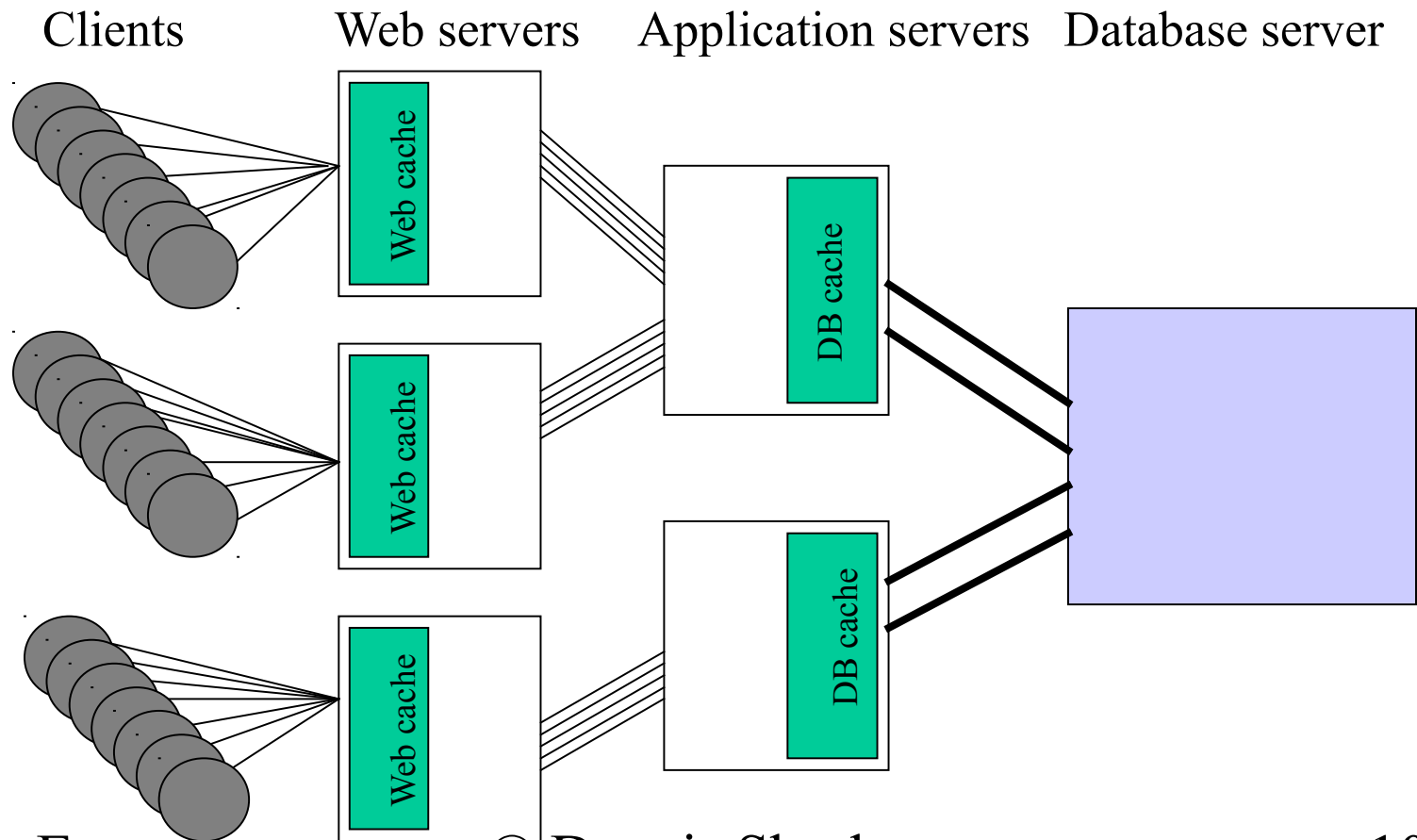
# Tuning E-Commerce Applications

Database-backed web-sites:

- Online shops
- Shop comparison portals
- MS TerraServer



# E-commerce Application Architecture



# E-commerce Application Workload

- Touristic searching (frequent, cached)
  - Access the top few pages. Pages may be personalized. Data may be out-of-date.
- Category searching (frequent, partly cached and need for timeliness guarantees)
  - Down some hierarchy, e.g., men's clothing.
- Keyword searching (frequent, uncached, need for timeliness guarantees)
- Shopping cart interactions (rare, but transactional)
- Electronic purchasing (rare, but transactional)

# Design Issues

- Need to keep historic information
  - Electronic payment acknowledgements get lost.
- Preparation for variable load
  - Regular patterns of web site accesses during the day, and within a week.
- Possibility of disconnections
  - State information transmitted to the client (cookies)
- Special consideration for low bandwidth
- Schema evolution
  - Representing e-commerce data as attribute-value pairs (IBM Websphere)

# Caching

- Web cache:
  - Static web pages
  - Caching fragments of dynamically created web pages
- Database cache (Oracle9iAS, TimesTen's FrontTier)
  - Materialized views to represent cached data.
  - Queries are executed either using the database cache or the database server. Updates are propagated to keep the cache(s) consistent.
  - Note to vendors: It would be good to have queries distributed between cache and server.

# Ecommerce -- setting

## Settings:

**shoppingcart**( shopperid, itemid, price, qty);

- 500000 rows; warm buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Ecommerce -- transactions

## Concurrent Transactions:

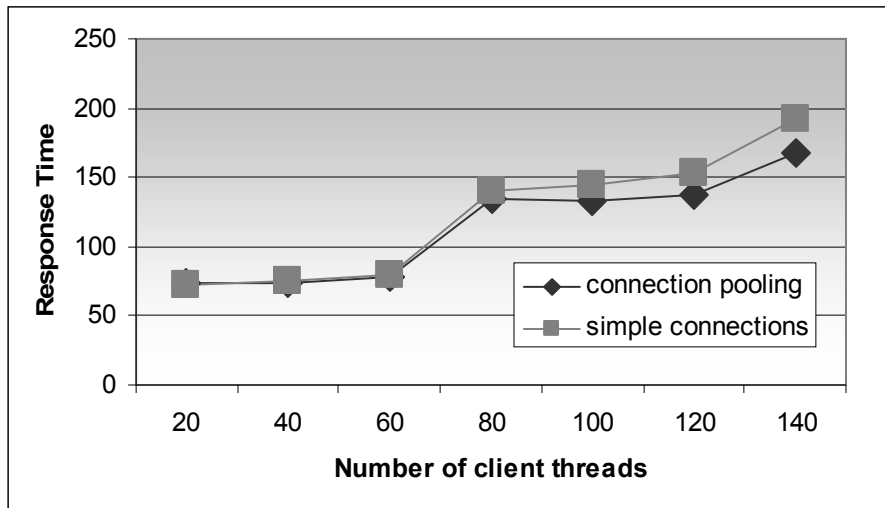
### – Mix

```
insert into shoppingcart values (107999,914,870,214);
update shoppingcart set Qty = 10 where shopperid =
 95047 and itemid = 88636;
delete from shoppingcart where shopperid = 86123 and
 itemid = 8321;
select shopperid, itemid, qty, price from shoppingcart
 where shopperid = ?;
```

### – Queries Only

```
select shopperid, itemid, qty, price from shoppingcart
 where shopperid = ?;
```

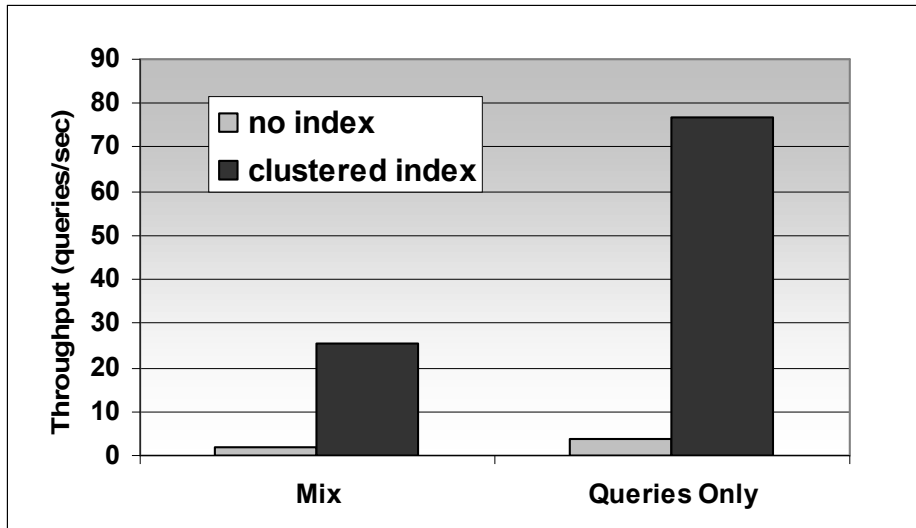
# Connection Pooling (no refusals)



- Each thread establishes a connection and performs 5 insert statements.
- If a connection cannot be established the thread waits 15 secs before trying again.
- The number of connection is limited to 60 on the database server.
- Using connection pooling, the requests are queued and serviced when possible. There are no refused connections.

Experiment performed on  
Oracle8i on Windows 2000

# Indexing

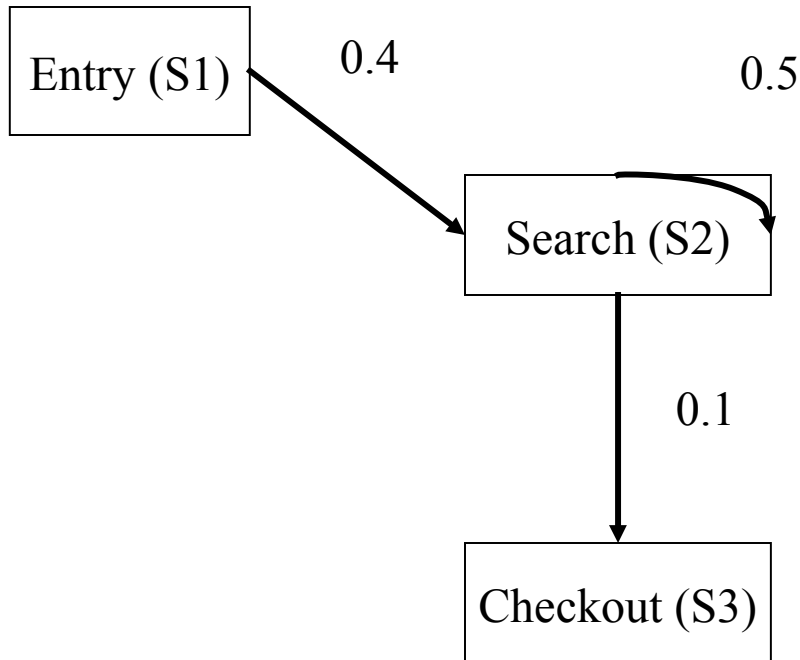


- Using a clustered index on shopperid in the shopping cart provides:
  - Query speed-up
  - Update/Deletion speed-up

Experiment performed on  
SQL Server 2000  
on Windows 2000



# Capacity Planning



Getting the demand assumptions right is what makes capacity planning hard

- Arrival Rate
  - $A1$  is given as an assumption
  - $A2 = (0.4 A1) + (0.5 A2)$
  - $A3 = 0.1 A2$
- Service Time (S)
  - $S1, S2, S3$  are measured
- Utilization
  - $U = A \times S$
- Response Time
  - $R = U / (A(1-U)) = S / (1-U)$   
(assuming Poisson arrivals)

# Datawarehouse Tuning

- Aggregate (strategic) targeting:
  - Aggregates flow up from a wide selection of data, and then
  - Targeted decisions flow down
- Examples:
  - Riding the wave of clothing fads
  - Tracking delays for frequent-flyer customers

# Data Warehouse Workload

- Broad
  - **Aggregate queries over ranges of values, e.g., find the total sales by region and quarter.**
- Deep
  - **Queries that require precise individualized information, e.g., which frequent flyers have been delayed several times in the last month?**
- Dynamic (vs. Static)
  - **Queries that require up-to-date information, e.g. which nodes have the highest traffic now?**

# Tuning Knobs

- Indexes
- Materialized views
- Approximation

# Bitmaps -- data

## Settings:

```
lineitem (L_ORDERKEY, L_PARTKEY , L_SUPPKEY, L_LINENUMBER,
 L_QUANTITY, L_EXTENDEDPRICE ,
 L_DISCOUNT, L_TAX , L_RETURNFLAG, L_LINESTATUS ,
 L_SHIPDATE, L_COMMITDATE,
 L_RECEIPTDATE, L_SHIPINSTRUCT ,
 L_SHIPMODE , L_COMMENT);
```

```
create bitmap index b_lin_2 on lineitem(l_returnflag);
```

```
create bitmap index b_lin_3 on lineitem(l_linestatus);
```

```
create bitmap index b_lin_4 on lineitem(l_linenum);
```

- 100000 rows ; cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.

# Bitmaps -- queries

## Queries:

- 1 attribute

```
select count(*) from lineitem where l_returnflag =
 'N';
```

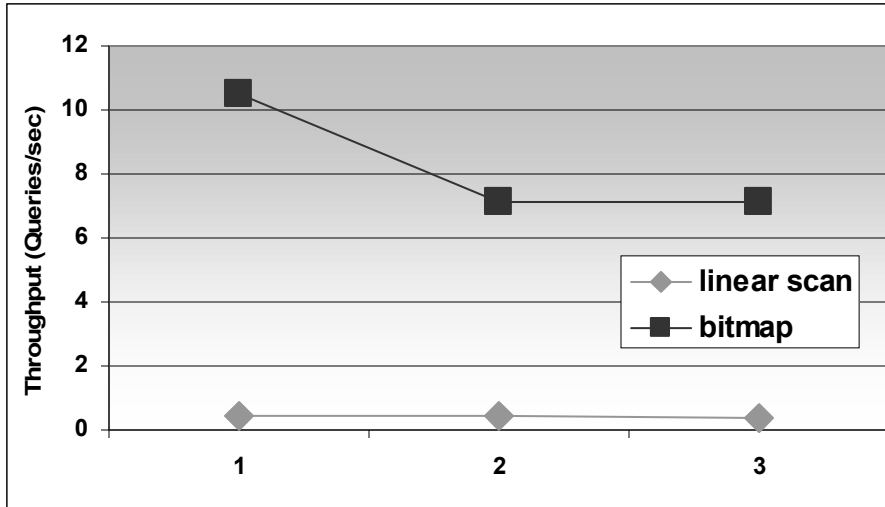
- 2 attributes

```
select count(*) from lineitem where l_returnflag = 'N'
 and l_linenumbers > 3;
```

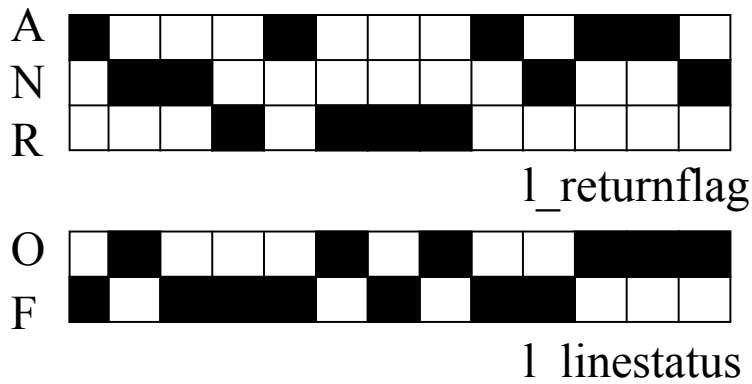
- 3 attributes

```
select count(*) from lineitem where l_returnflag =
 'N' and l_linenumbers > 3 and l_linestatus = 'F';
```

# Bitmaps



- Order of magnitude improvement compared to scan.
- Bitmaps are best suited for multiple conditions on several attributes, each having a low selectivity.



# Multidimensional Indexes -- data

## Settings:

```
create table spatial_facts
(a1 int, a2 int, a3 int, a4 int, a5 int, a6 int, a7
int, a8 int, a9 int, a10 int, geom_a3_a7
mdsys.sdo_geometry);
```

```
create index r_spatialfacts on
spatial_facts(geom_a3_a7) indextype is
mdsys.spatial_index;
```

```
create bitmap index b2_spatialfacts on
spatial_facts(a3,a7);
```

- 500000 rows ; cold buffer
- Dual Pentium II (450MHz, 512Kb), 512 Mb RAM, 3x18Gb drives (10000RPM), Windows 2000.



# Multidimensional Indexes -- queries

## Queries:

### – Point Queries

```
select count(*) from fact where a3 = 694014 and a7 = 928878;
```

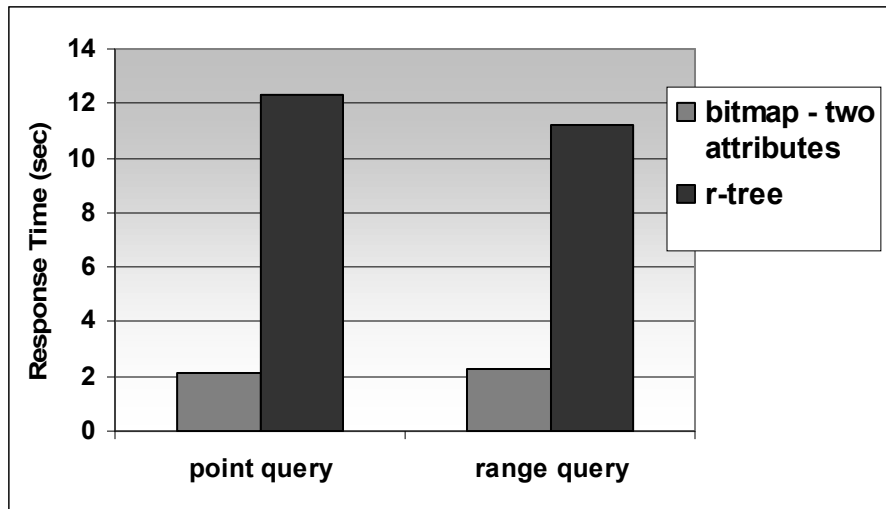
```
select count(*) from spatial_facts where SDO_RELATE(geom_a3_a7,
MDSYS.SDO_GEOMETRY(2001, NULL, MDSYS.SDO_POINT_TYPE(694014, 928878,
NULL), NULL, NULL), 'mask=equal querytype=WINDOW') = 'TRUE';
```

### – Range Queries

```
select count(*) from spatial_facts where SDO_RELATE(geom_a3_a7,
mdsys.sdo_geometry(2003, NULL, NULL,
mdsys.sdo_elem_info_array(1, 1003, 3), mdsys.sdo_ordinate_array(10, 80
0000, 1000000, 1000000)), 'mask=inside querytype=WINDOW') = 'TRUE';
```

```
select count(*) from spatial_facts where a3 > 10 and a3 < 1000000
and a7 > 800000 and a7 < 1000000;
```

# Multidimensional Indexes



- Oracle 8i on Windows 2000
- Spatial Extension:
  - 2-dimensional data
  - Spatial functions used in the query
- R-tree does not perform well because of the overhead of spatial extension.

# Multidimensional Indexes

## R-Tree

SELECT STATEMENT

SORT AGGREGATE

TABLE ACCESS BY INDEX

ROWID SPATIAL\_FACTS

DOMAIN INDEX

R\_SPATIALFACTS

## Bitmaps

SELECT STATEMENT

SORT AGGREGATE

BITMAP CONVERSION

COUNT

BITMAP AND

BITMAP INDEX SINGLE  
VALUE B\_FACT7

BITMAP INDEX SINGLE  
VALUE B\_FACT3

# Materialized Views -- data

## Settings:

```
orders(ordernum, itemnum, quantity, purchaser, vendor);
create clustered index i_order on orders(itemnum);

store(vendor, name);

item(itemnum, price);
create clustered index i_item on item(itemnum);
```

- 1000000 orders, 10000 stores, 400000 items; Cold buffer
- Oracle 9i
- Pentium III (1 GHz, 256 Kb), 1Gb RAM, Adapter 39160 with 2 channels, 3x18Gb drives (10000RPM), Linux Debian 2.4.

# Materialized Views -- data

## Settings:

```
create materialized view vendorOutstanding
build immediate
refresh complete
enable query rewrite
as
select orders.vendor, sum(orders.quantity*item.price)
from orders,item
where orders.itemnum = item.itemnum
group by orders.vendor;
```

# Materialized Views -- transactions

## Concurrent Transactions:

### – Insertions

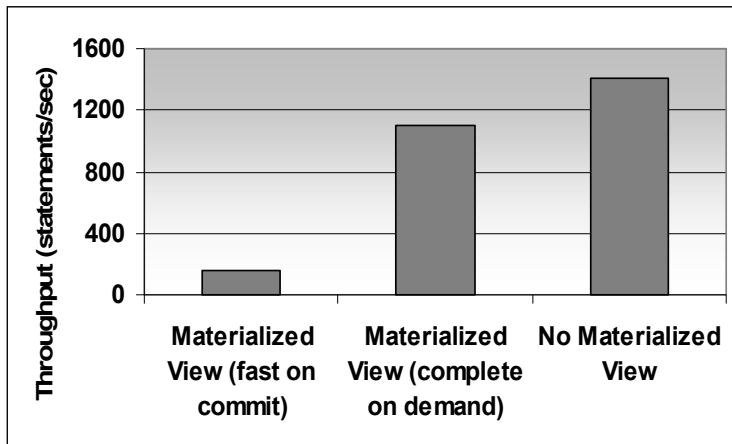
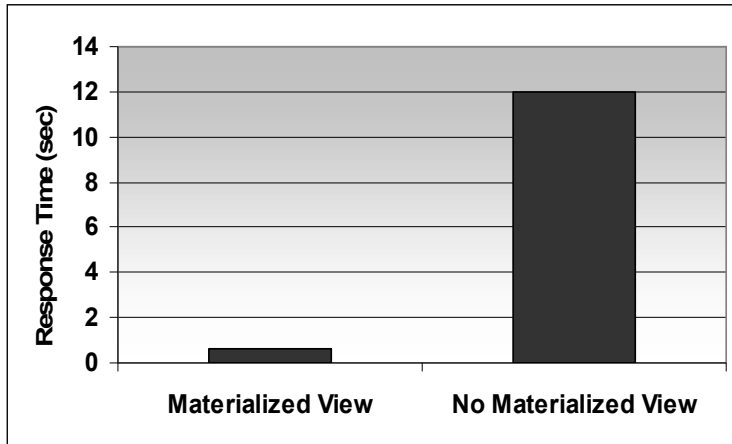
```
insert into orders values
 (1000350,7825,562,'xxxxxx6944','vendor4');
```

### – Queries

```
select orders.vendor, sum(orders.quantity*item.price)
from orders,item
where orders.itemnum = item.itemnum
group by orders.vendor;

select * from vendorOutstanding;
```

# Materialized Views



- Graph:
  - Oracle9i on Linux
  - Total sale by vendor is materialized
- Trade-off between query speed-up and view maintenance:
  - The impact of incremental maintenance on performance is significant.
  - Rebuild maintenance achieves a good throughput.
  - A static data warehouse offers a good trade-off.

# Materialized View Maintenance

- Problem when large number of views to maintain.
- The order in which views are maintained is important:
  - A view can be computed from an existing view instead of being recomputed from the base relations (total per region can be computed from total per nation).
- Let the views and base tables be nodes  $v_i$
- Let there be an edge from  $v_1$  to  $v_2$  if it possible to compute the view  $v_2$  from  $v_1$ . Associate the cost of computing  $v_2$  from  $v_1$  to this edge.
- Compute all pairs shortest path where the start nodes are the set of base tables.
- The result is an acyclic graph  $A$ . Take a topological sort of  $A$  and let that be the order of view construction.



# Approximations -- data

## Settings:

- TPC-H schema
- Approximations

```
insert into approxlineitem
select top 6000 *
 from lineitem
 where l_linenum = 4;
```

```
insert into approxorders
select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE,
 O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY,
 O_COMMENT
 from orders, approxlineitem
 where o_orderkey = l_orderkey;
```

# Approximations -- queries

```
insert into approxsupplier
select distinct S_SUPPKEY,
 S_NAME ,
 S_ADDRESS,
 S_NATIONKEY,
 S_PHONE,
 S_ACCTBAL,
 S_COMMENT
from approxlineitem, supplier
where s_suppkey = l_suppkey;
```

```
insert into approxpart
select distinct P_PARTKEY,
 P_NAME ,
 P_MFGR ,
 P_BRAND ,
 P_TYPE ,
 P_SIZE ,
 P_CONTAINER ,
 P_RETAILPRICE ,
 P_COMMENT
from approxlineitem, part
where p_partkey = l_partkey;
```

```
insert into approxpartsupp
select distinct PS_PARTKEY,
 PS_SUPPKEY,
 PS_AVAILQTY,
 PS_SUPPLYCOST,
 PS_COMMENT
from partsupp, approxpart, approxsupplier
where ps_partkey = p_partkey and
 ps_suppkey = s_suppkey;
```

```
insert into approxcustomer
select distinct C_CUSTKEY,
 C_NAME ,
 C_ADDRESS,
 C_NATIONKEY,
 C_PHONE ,
 C_ACCTBAL,
 C_MKTSEGMENT,
 C_COMMENT
from customer, approxorders
where o_custkey = c_custkey;
insert into approxregion select * from
region;
insert into approxnation select * from
nation;
```

# Approximations -- more queries

## Queries:

- Single table query on lineitem

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
 sum(l_extendedprice) as sum_base_price,
 sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
 sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
 sum_charge,
 avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
 avg(l_discount) as avg_disc, count(*) as count_order
from lineitem
where datediff(day, l_shipdate, '1998-12-01') <= '120'
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

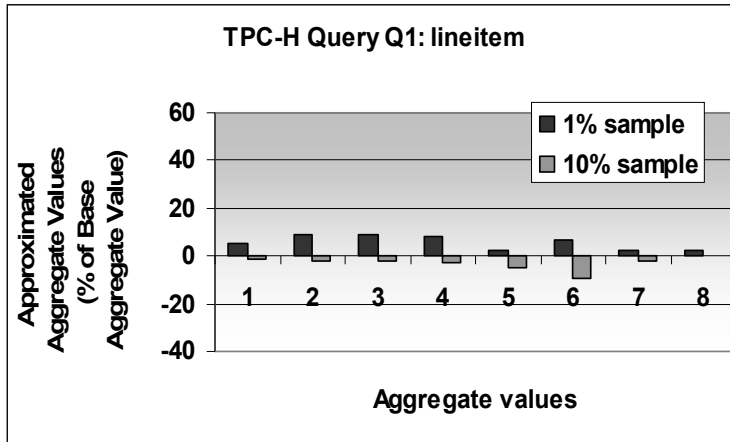
# Approximations -- still more

## Queries:

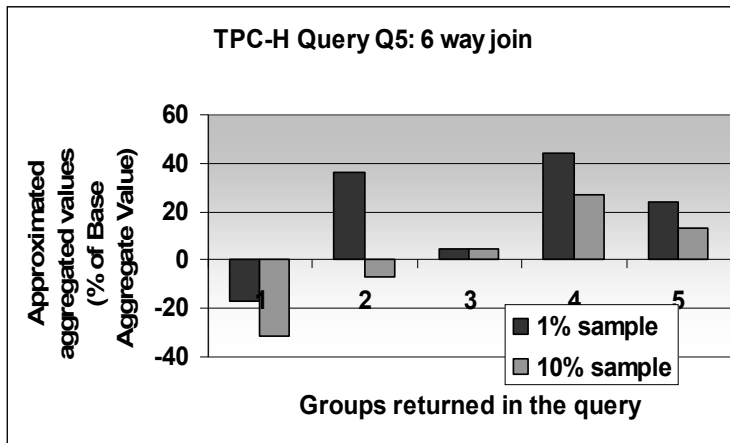
### – 6-way join

```
select n_name, avg(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and l_suppkey = s_suppkey
 and c_nationkey = s_nationkey
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'AFRICA'
 and o_orderdate >= '1993-01-01'
 and datediff(year, o_orderdate, '1993-01-01') < 1
group by n_name
order by revenue desc;
```

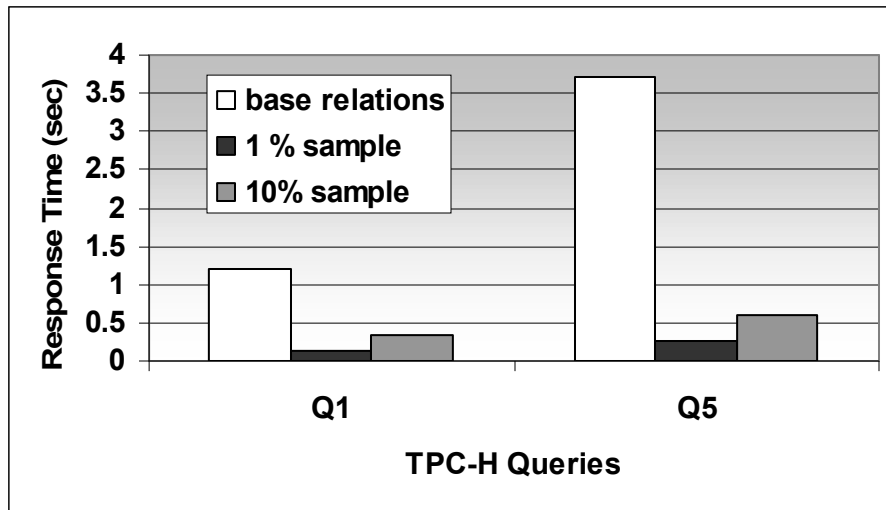
# Approximation accuracy



- Good approximation for query Q1 on lineitem
- The aggregated values obtained on a query with a 6-way join are significantly different from the actual values -- for some applications may still be good enough.



# Approximation Speedup



- Aqua approximation on the TPC-H schema
  - 1% and 10% lineitem sample propagated.
- The query speed-up obtained with approximated relations is significant.

# Tuning Distributed Applications

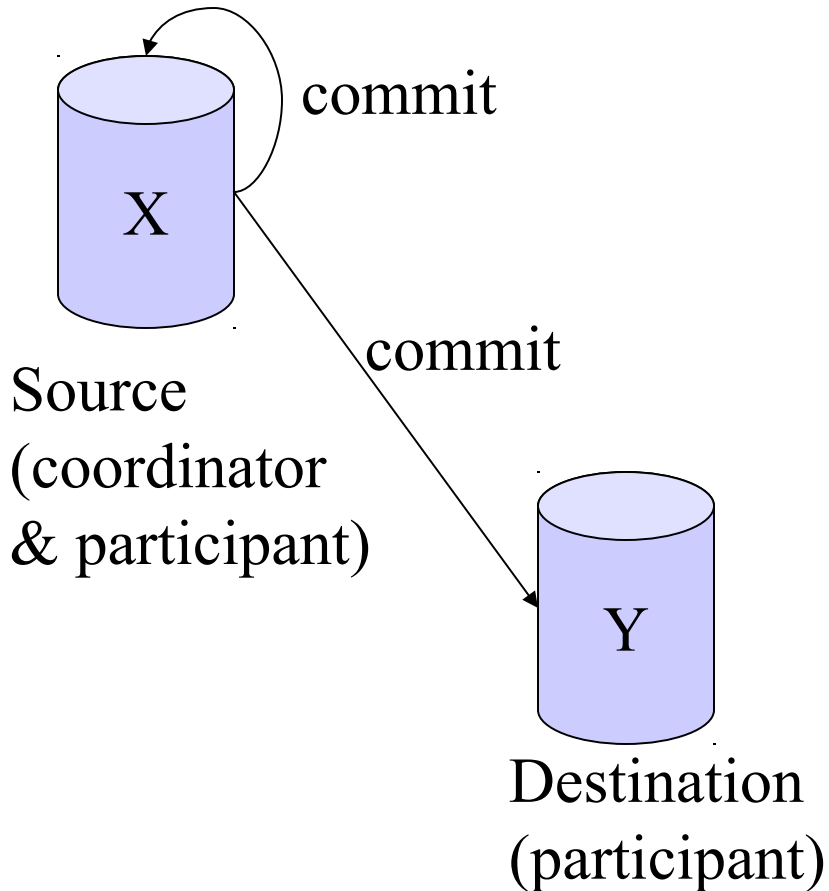
- Queries across multiple databases
  - Federated Datawarehouse
  - IBM's DataJoiner now integrated at DB2 v7.2
    - The source should perform as much work as possible and return as few data as possible for processing at the federated server
- Processing data across multiple databases

# A puzzle

- Two databases X and Y
  - X records inventory data to be used for restocking
  - Y contains sales data about shipments
- You want to improve shipping
  - Certain data of X should be postprocessed on Y shortly after it enters X
  - You are allowed to add/change transactions on X and Y
  - You want to avoid losing data from X and you want to avoid double-processing data on Y, even in the face of failures.

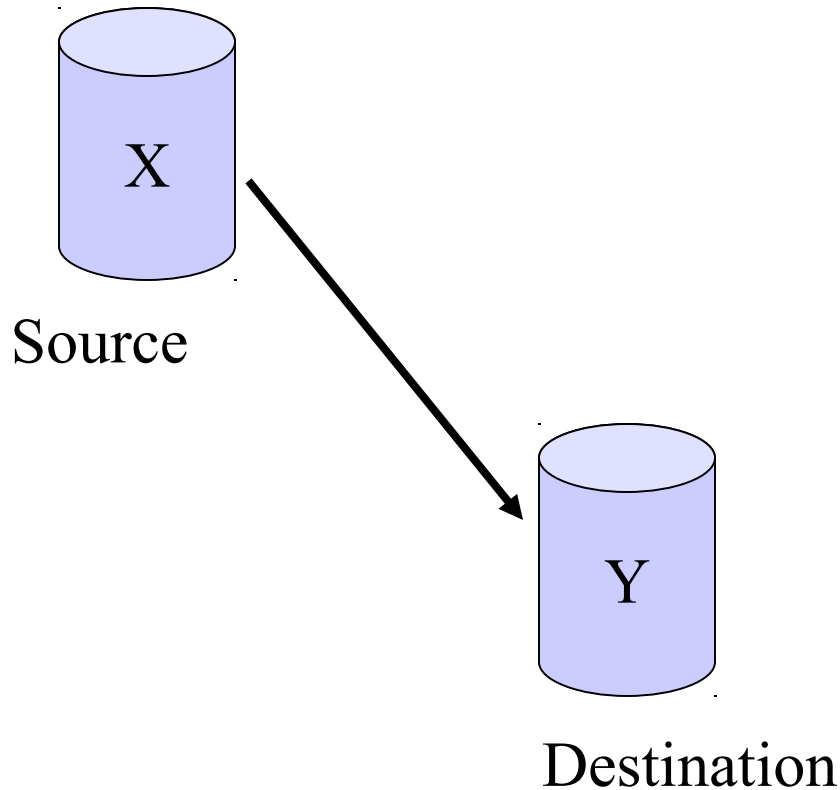


# Two-Phase Commit



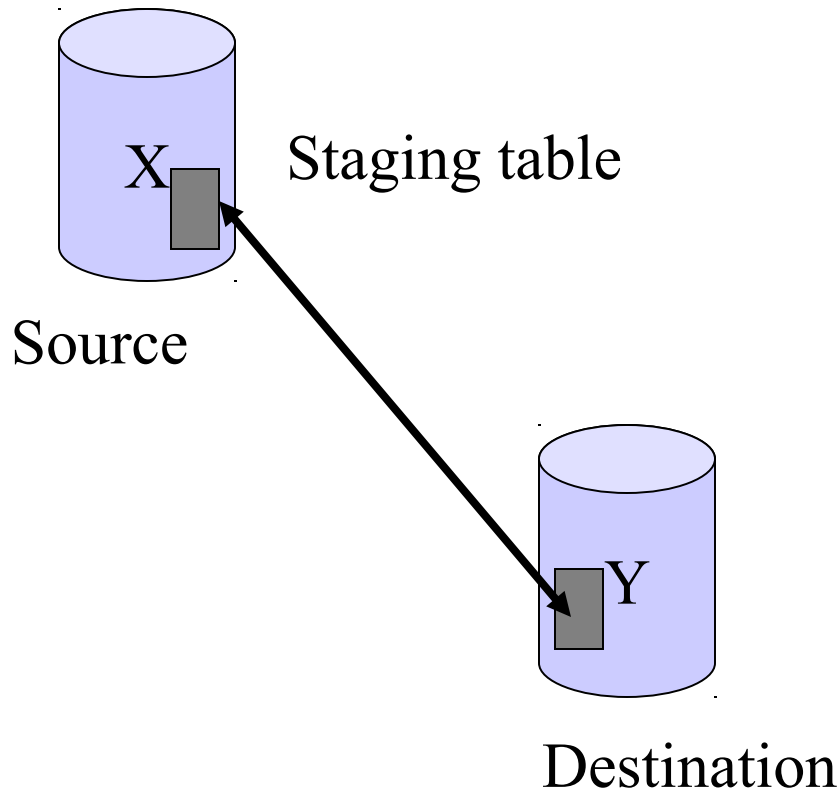
- + Commits are coordinated between the source and the destination
- If one participant fails then blocking can occur
- Not all db systems support prepare-to-commit interface

# Replication Server



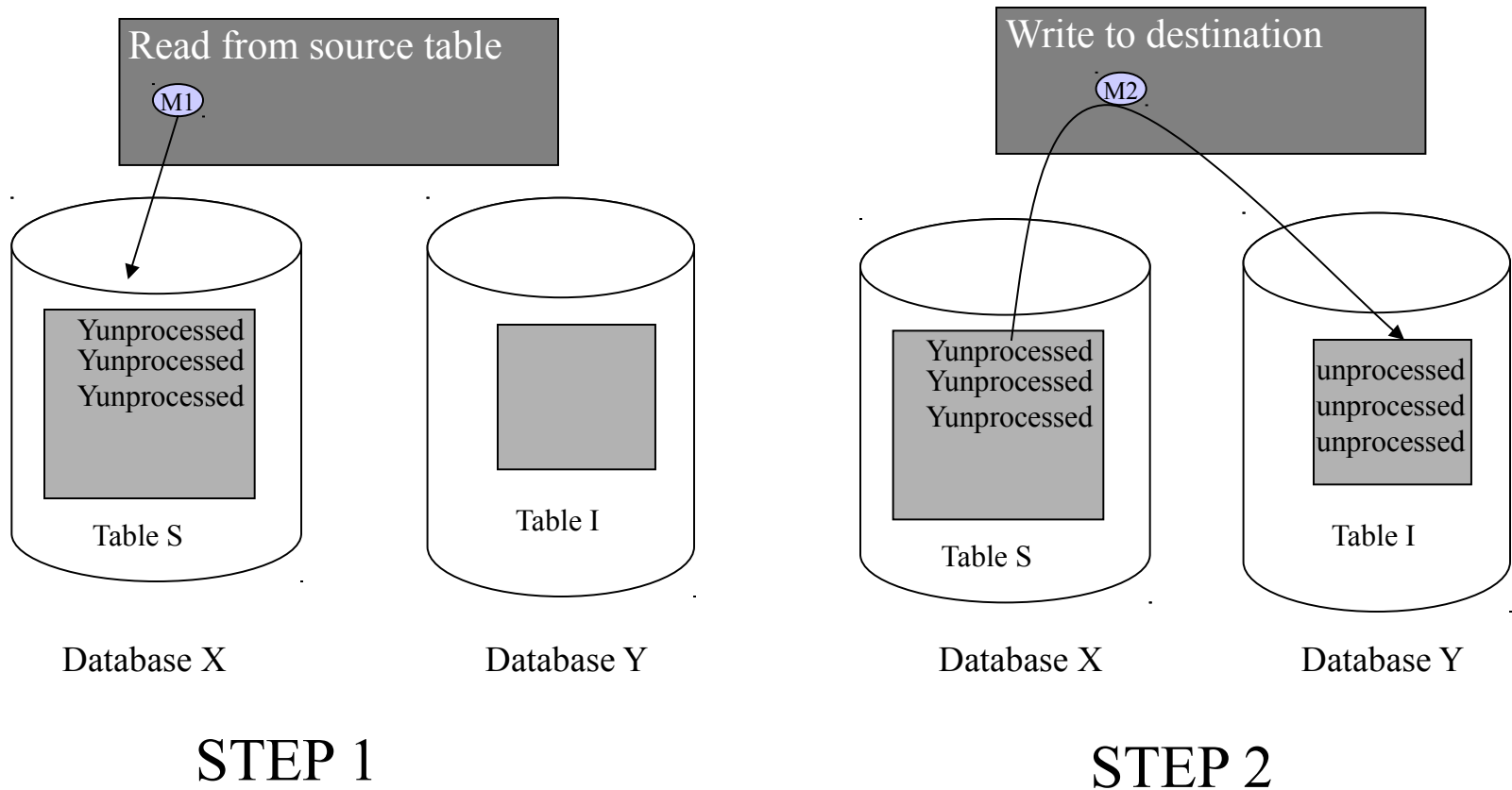
- + Destination within a few seconds of being up-to-date
- + Decision support queries can be asked on destination db
- Administrator is needed when network connection breaks!

# Staging Tables

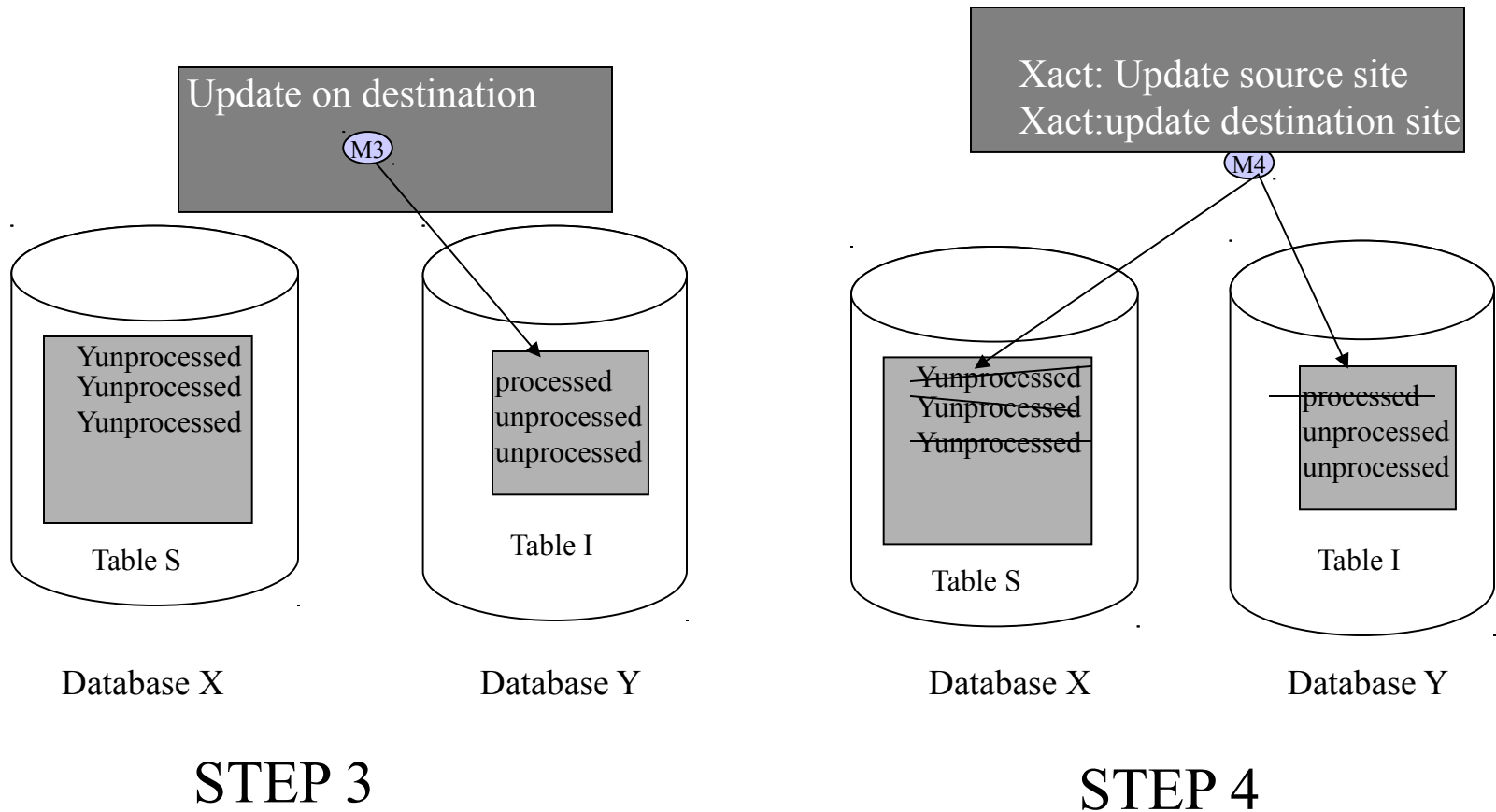


- + No specific mechanism is necessary at source or destination
- Coordination of transactions on X and Y

# Staging Tables



# Staging Tables



# Troubleshooting Techniques (Alberto Lerner)

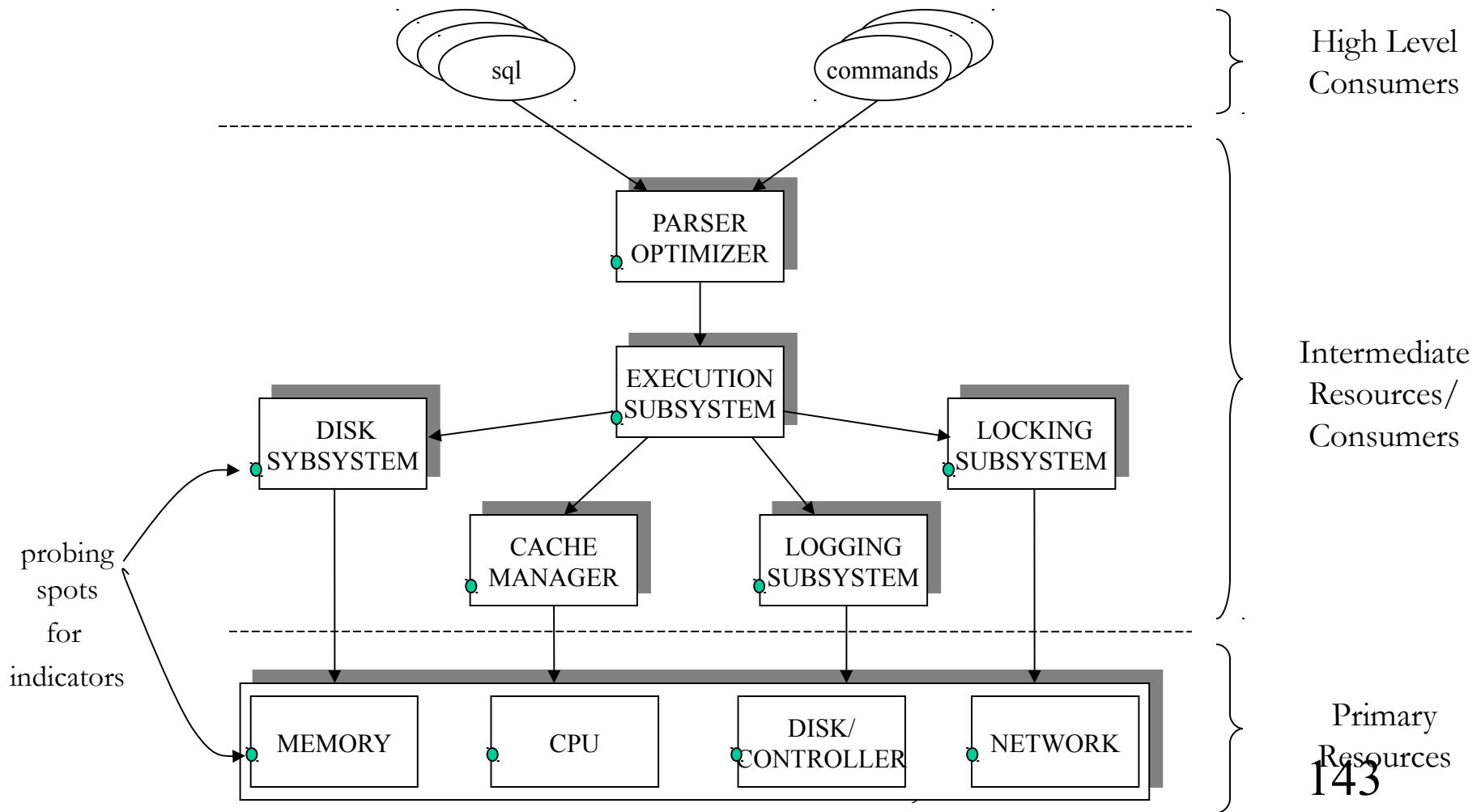
Consumer-producer chain framework

Query plan monitors

Performance monitors

Event monitors

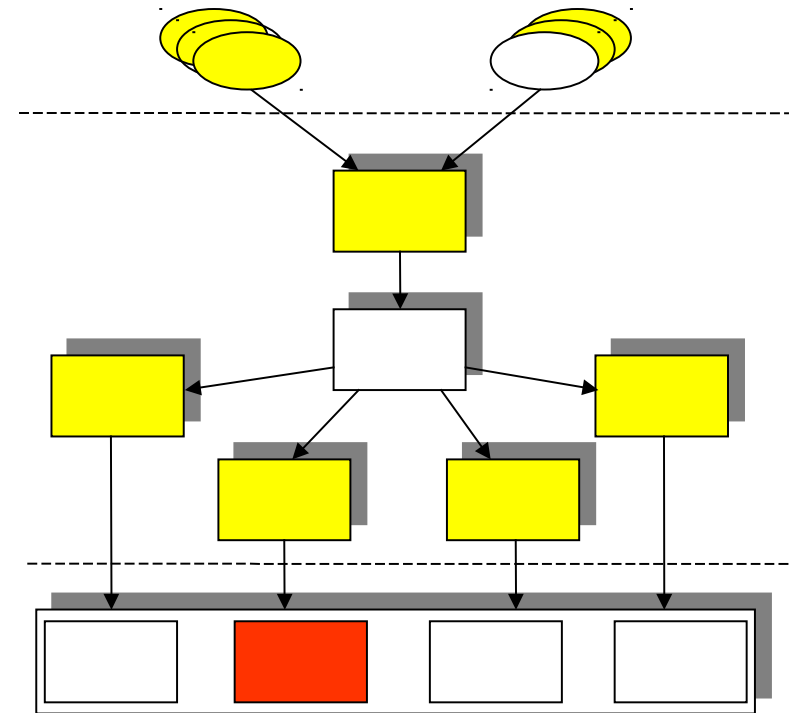
# A Consumer-Producer Chain of a DBMS's Resources



# Recurrent Patterns of Problems

Effects are not always felt first where the cause is!

- An overloading high-level consumer
- A poorly parameterized subsystem
- An overloaded primary resource

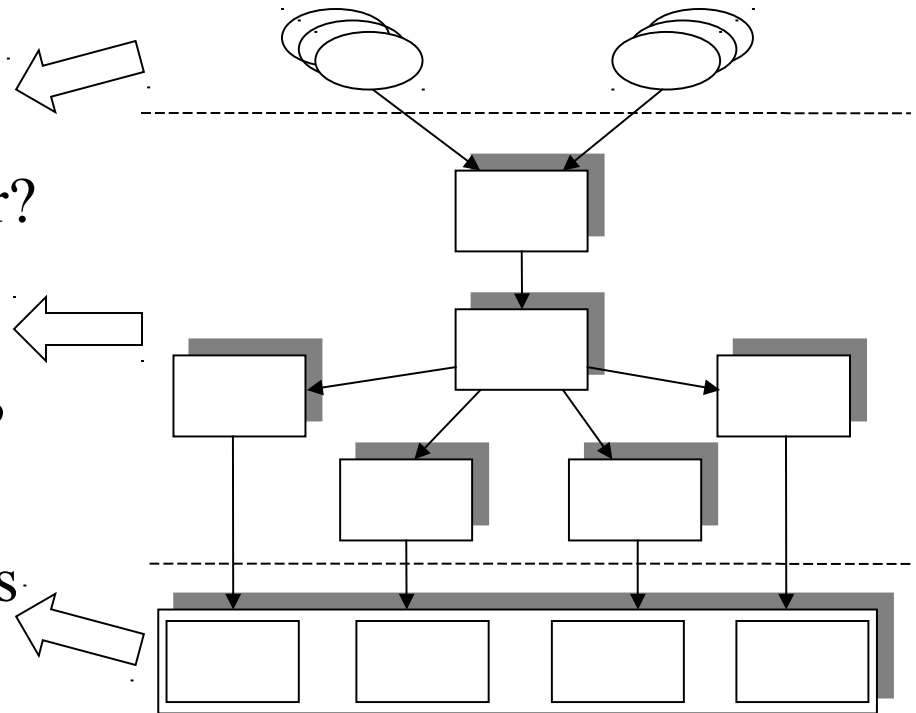




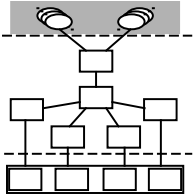
# A Systematic Approach to Monitoring

Extract indicators to answer the following questions

- Question 1: Are critical queries being served in the most efficient manner?
- Question 2: Are subsystems making optimal use of resources?
- Question 3: Are there enough primary resources available?

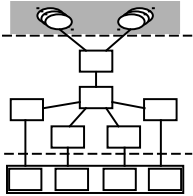


# Investigating High Level Consumers



- Answer question 1:  
“Are critical queries being served in the most efficient manner?”
  1. Identify the critical queries
  2. Analyze their access plans
  3. Profile their execution

# Identifying Critical Queries

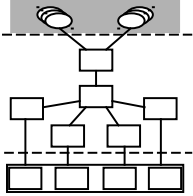


Critical queries are usually those that:

- Take a long time
- Are frequently executed

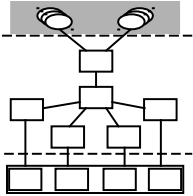
Often, a user complaint will tip us off.

# Event Monitors to Identify Critical Queries



- If no user complains...
- Capture usage measurements at end of each query and then sort by usage  $\times$  time.
- Less overhead than other type of tools because indicators are usually by-product of operations monitored.
- Typical measures include CPU used, IO used, locks obtained etc.

# An example Event Monitor



- CPU indicators sorted by Oracle's Trace Data Viewer
- Similar tools: DB2's Event Monitor and MSSQL's Server Profiler

CPU Consumption - Oracle Trace Data Viewer [Data View]

File View Data View Help

CPU Consumption

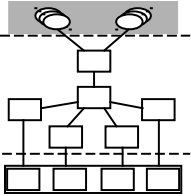
| *Total CPU | Parse CPU | Execute CPU | Fetch CPU | Sorts in Memory | SQL Text                  |
|------------|-----------|-------------|-----------|-----------------|---------------------------|
| 39.000000  | 2.000000  | 37.000000   | 0.000000  | 3               | BEGIN DBMS_APPLICATIONS   |
| 39.000000  | 2.000000  | 37.000000   | 0.000000  | 3               | BEGIN DBMS_APPLICATIONS   |
| 39.000000  | 2.000000  | 37.000000   | 0.000000  | 3               | BEGIN DBMS_APPLICATIONS   |
| 28.000000  | 3.000000  | 0.000000    | 25.000000 | 0               | select table_name, e.* fr |
| 20.000000  | 16.000000 | 4.000000    | 0.000000  | 8               | BEGIN DBMS_OUTPUT         |
| 20.000000  | 16.000000 | 4.000000    | 0.000000  | 8               | BEGIN DBMS_OUTPUT         |
| 20.000000  | 16.000000 | 4.000000    | 0.000000  | 8               | BEGIN DBMS_OUTPUT         |
| 9.000000   | 9.000000  | 0.000000    | 0.000000  | 6               | select * from scott.emp   |
| 3.000000   | 3.000000  | 0.000000    | 0.000000  | 0               | SELECT USER FROM          |
| 3.000000   | 3.000000  | 0.000000    | 0.000000  | 0               | SELECT USER FROM          |

SQL Statement Details

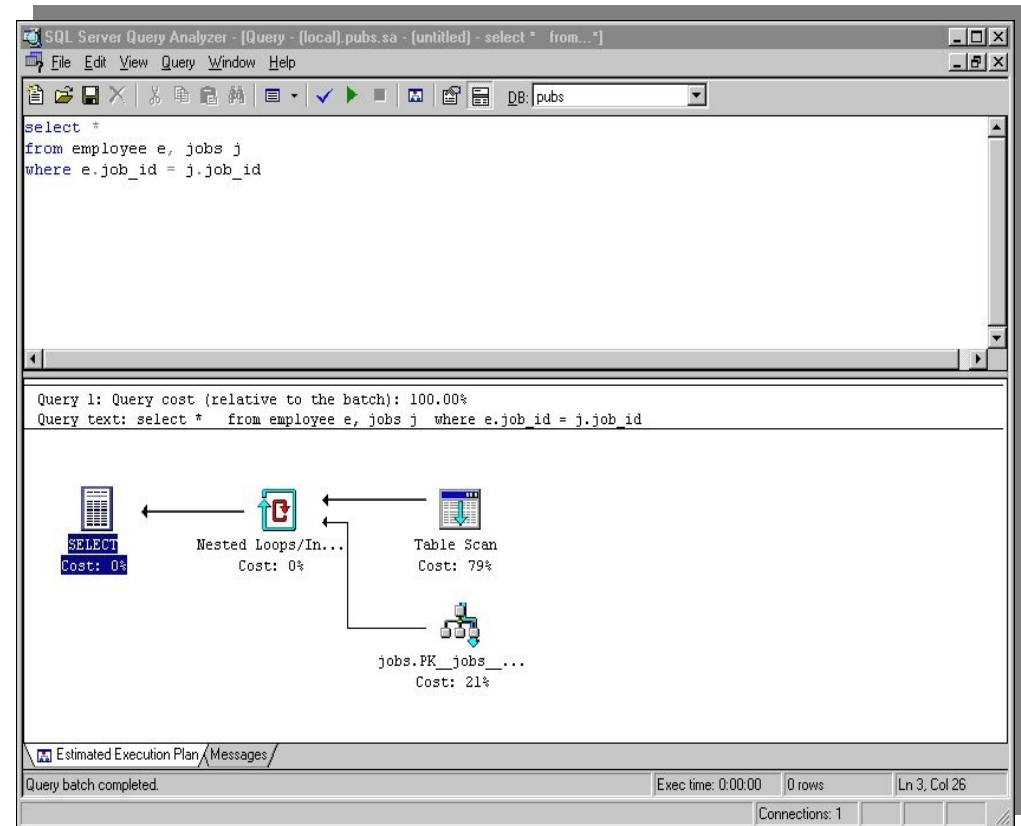
SELECT \*  
FROM scott.emp

For Help, press F1

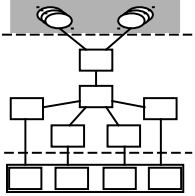
# An example Plan Explainer



- Access plan according to MSSQL's Query Analyzer
- Similar tools: DB2's Visual Explain and Oracle's SQL Analyze Tool



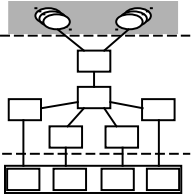
# Finding Strangeness in Access Plans



What to pay attention to in a plan

- Access paths for each table
- Sorts or intermediary results
- Order of operations
- Algorithms used in the operators

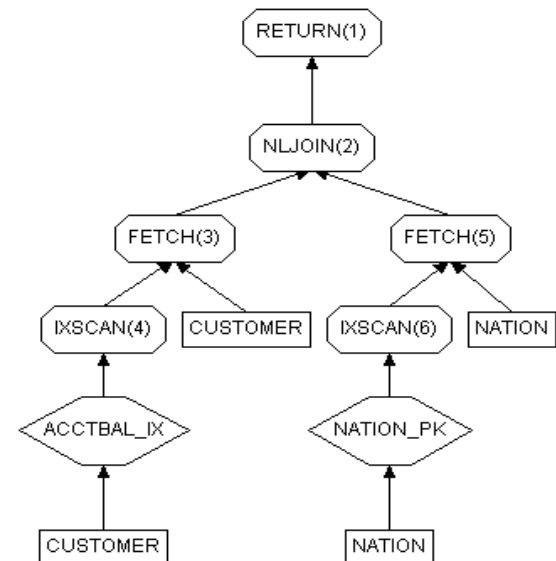
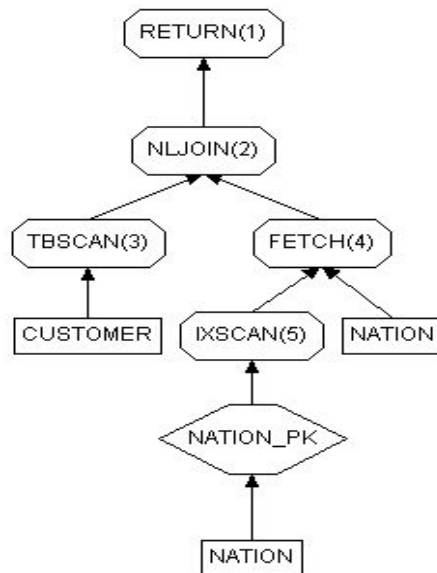
# To Index or not to index?



select c\_name, n\_name from CUSTOMER join NATION  
on c\_nationkey=n\_nationkey where c\_acctbal > 0

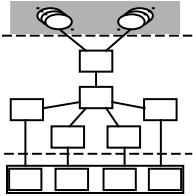
Which plan performs best?

(nation\_pk is an non-clustered index over n\_nationkey,  
and similarly for acctbal\_ix over c\_acctbal)





# Non-clustering indexes can be trouble

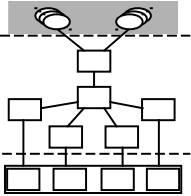


For a low selectivity predicate, each access to the index generates a random access to the table – possibly duplicate! It ends up that the number of pages read from the table is greater than its size, i.e., a table scan is way better

|                      | Table Scan    | Index Scan    |
|----------------------|---------------|---------------|
| CPU time             | 5 sec         | 76 sec        |
| data logical reads   | 143,075 pages | 272,618 pages |
| data physical reads  | 6,777 pages   | 131,425 pages |
| index logical reads  | 136,319 pages | 273,173 pages |
| index physical reads | 7 pages       | 552 pages     |

# An example

## Performance Monitor (query level)



- Details of buffer and CPU consumption for a query according to DB2's Benchmark tool
- Similar tools: MSSQL's SET STATISTICS switch and Oracle's SQL Analyze Tool

```
Statement number: 1
select C_NAME, N_NAME
from DBA.CUSTOMER join DBA.NATION on C_NATIONKEY = N_NATIONKEY
where C_ACCTBAL > 0
```

```
Number of rows retrieved is: 136308
Number of rows sent to output is: 0
Elapsed Time is: 76.349 seconds
```

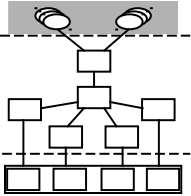
```
...
Buffer pool data logical reads = 272618
Buffer pool data physical reads = 131425
Buffer pool data writes = 0
Buffer pool index logical reads = 273173
Buffer pool index physical reads = 552
Buffer pool index writes = 0
Total buffer pool read time (ms) = 71352
Total buffer pool write time (ms) = 0
```

Summary of Results

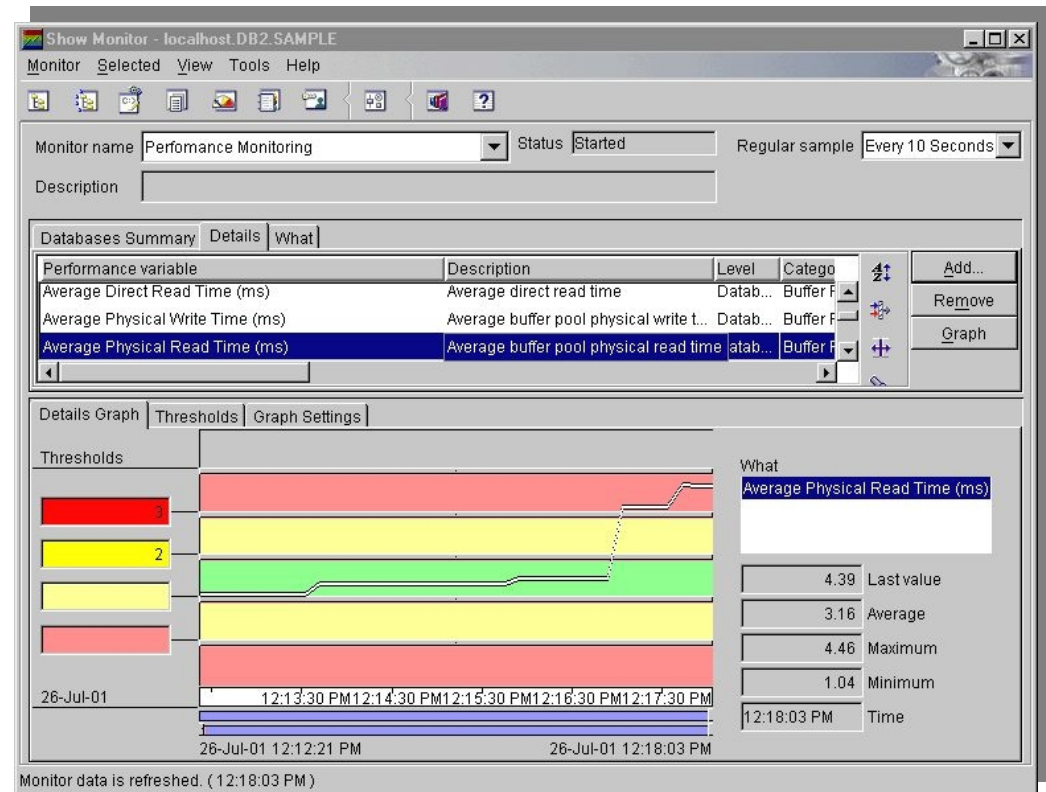
| Statement # | Elapsed Time (s) | Agent CPU Time (s) | Rows Fetched | Rows Printed |
|-------------|------------------|--------------------|--------------|--------------|
| 1           | 76.349           | 6.670              | 136308       | 0            |

# An example

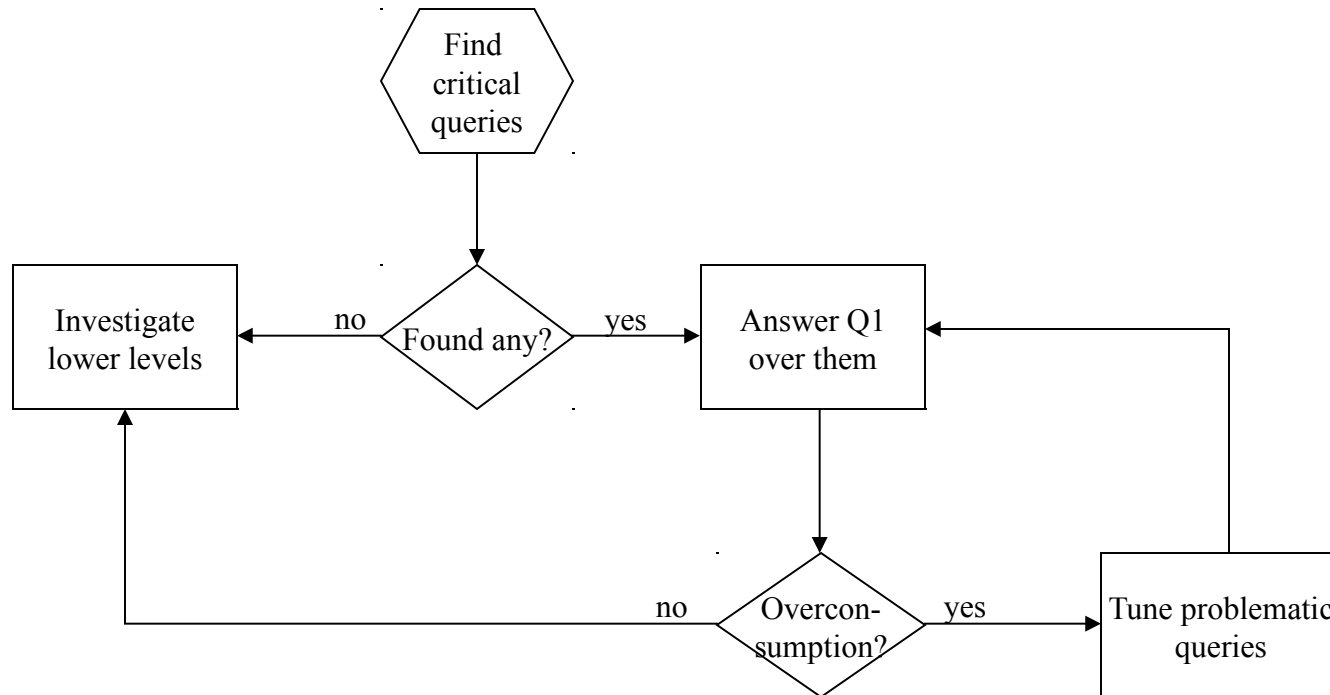
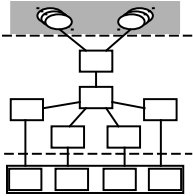
## Performance Monitor (system level)



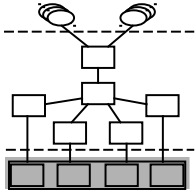
- An IO indicator's consumption evolution (qualitative and quantitative) according to DB2's System Monitor
- Similar tools: Window's Performance Monitor and Oracle's Performance Manager



# Investigating High Level Consumers: Summary



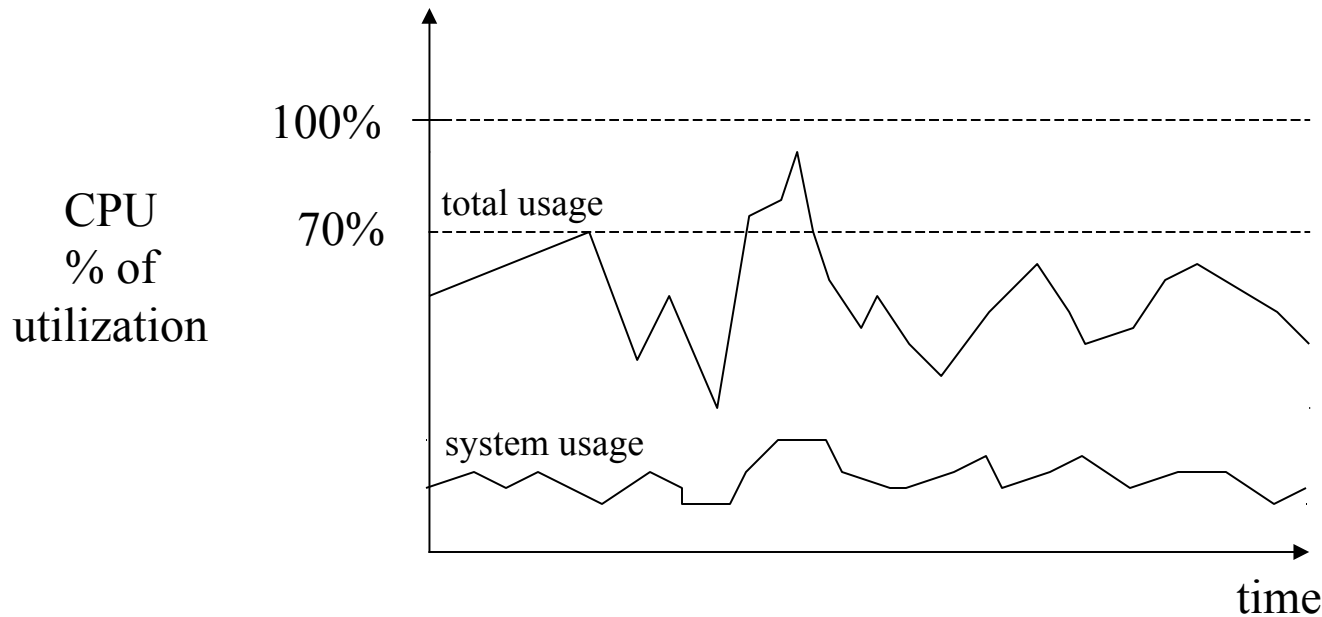
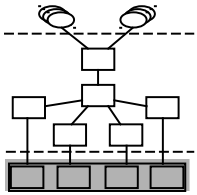
# Investigating Primary Resources



- Answer question 3:

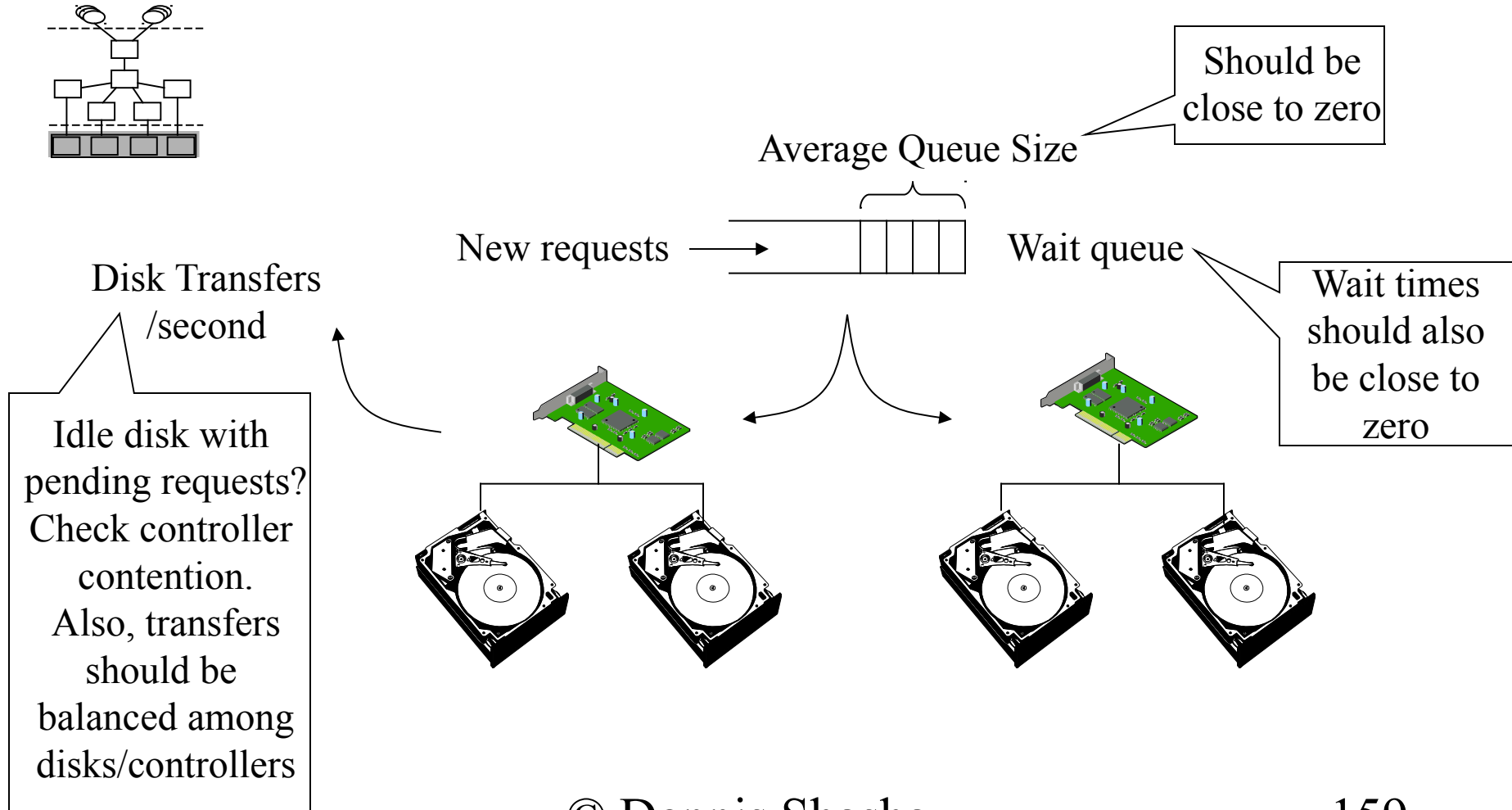
“Are there enough primary resources available for a DBMS to consume?”
- Primary resources are: CPU, disk/controllers, memory, and network
- Analyze specific OS-level indicators to discover bottlenecks.
- A system-level Performance Monitor is the right tool here

# CPU Consumption Indicators at the OS Level

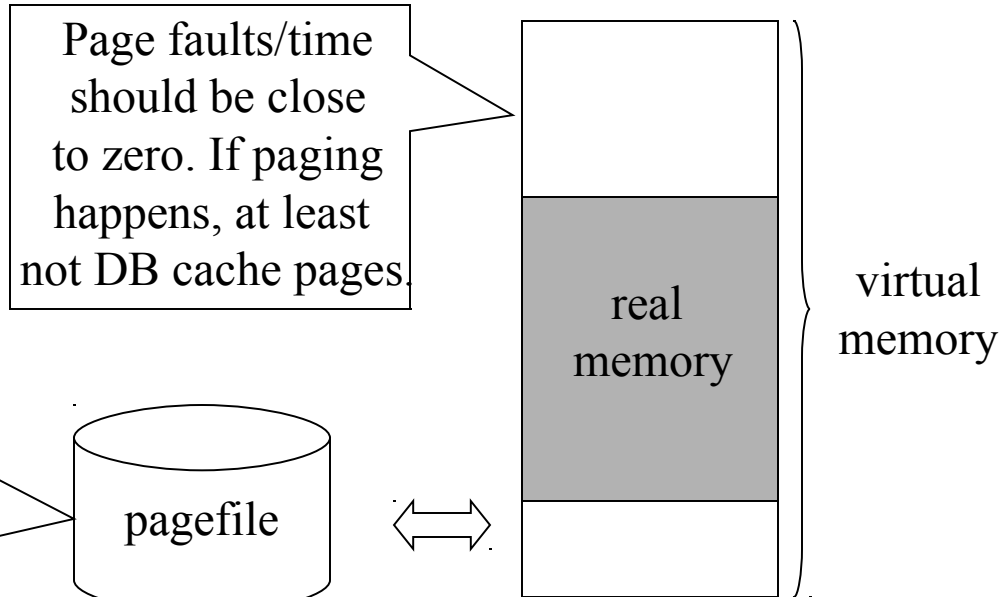
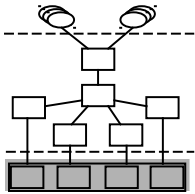


Sustained utilization over 70% should trigger the alert. System utilization shouldn't be more than 40%. DBMS (on a non-dedicated machine) should be getting a decent time share.

# Disk Performance Indicators at the OS Level

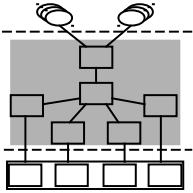


# Memory Consumption Indicators at the OS Level



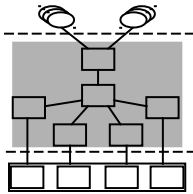


# Investigating Intermediate Resources/Consumers

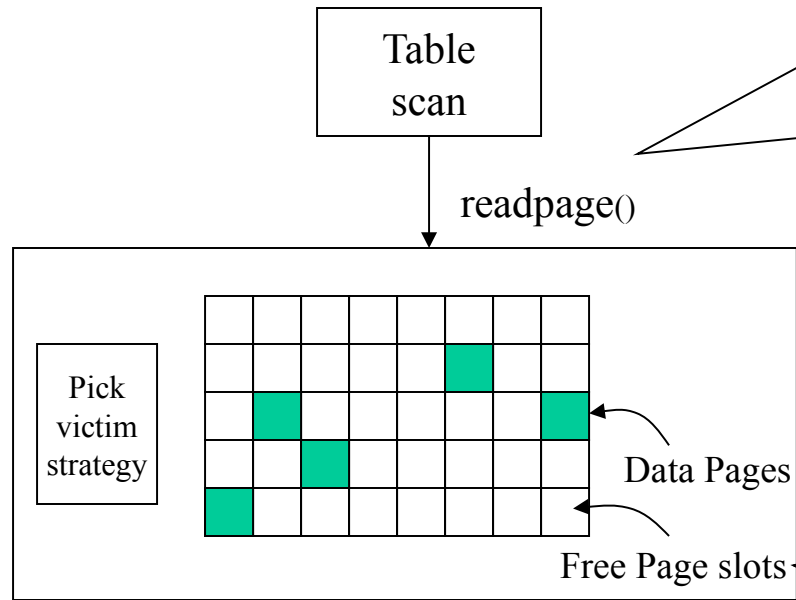


- Answer question 2:  
“Are subsystems making optimal use of resources?”
- Main subsystems: Cache Manager, Disk subsystem, Lock subsystem, and Log/Recovery subsystem
- Similarly to Q3, extract and analyze relevant Performance Indicators

# Cache Manager Performance Indicators



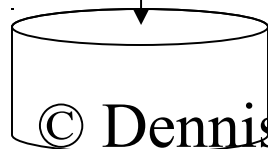
Cache  
Manager



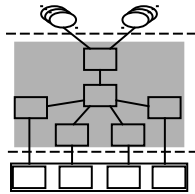
If page is not in the cache, readpage (logical) generate an actual IO (physical). Ratio of readpages that did not generate physical IO should be 90% or more

Pages are regularly saved to disk to make free space. # of free slots should always be  $> 0$

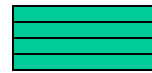
Page reads/  
writes



# Disk Manager Performance Indicators

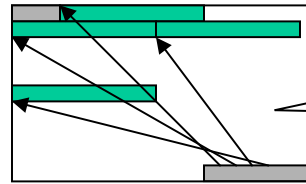


rows



Row displacement: should be kept under 5% of rows

page



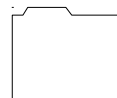
Free space fragmentation: pages with few space should not be in the free list

extent

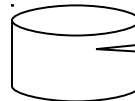


Data fragmentation: ideally files that store DB objects (table, index) should be in one or few (<5) contiguous extents

file



disk



File position: should balance workload evenly among all disks

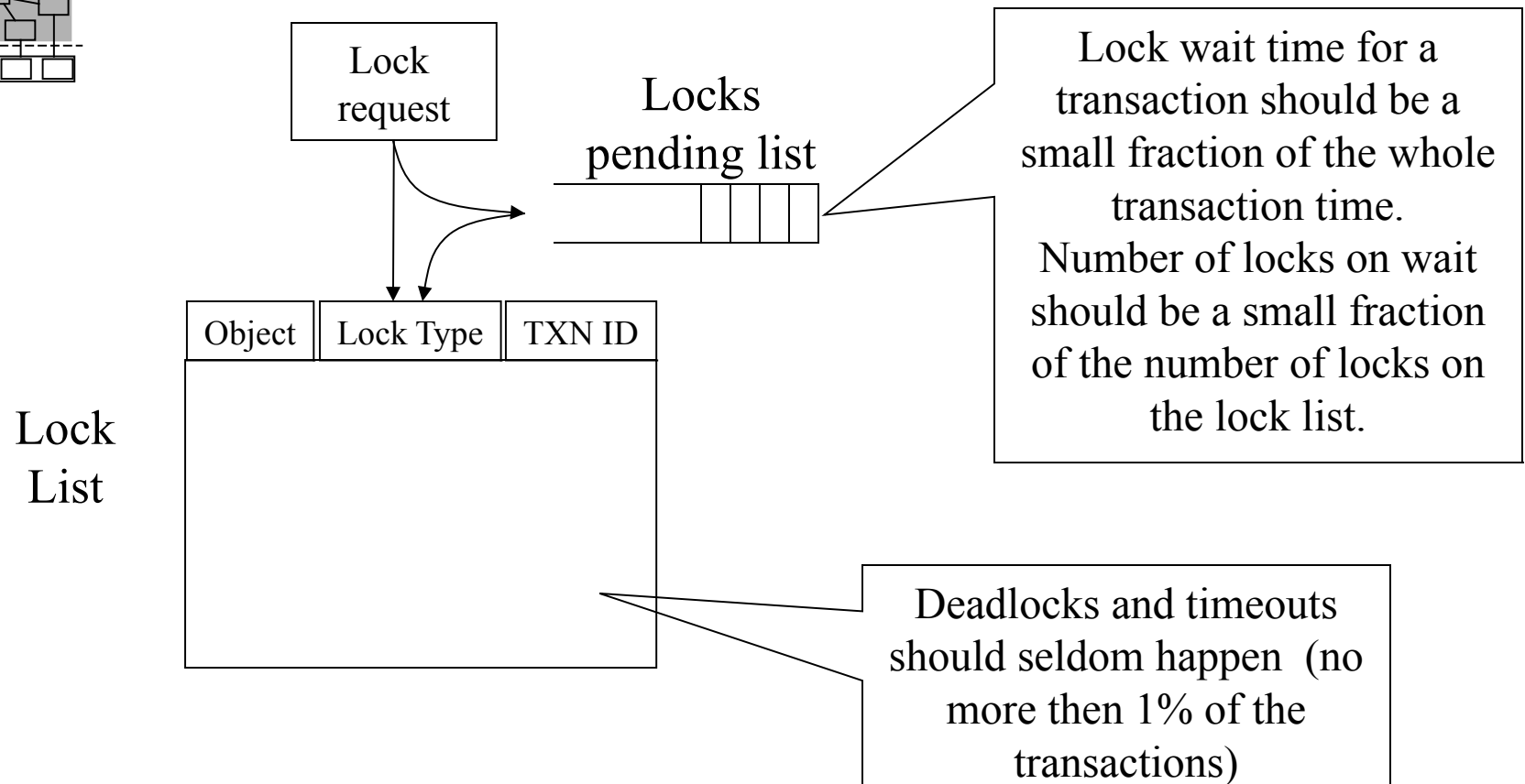
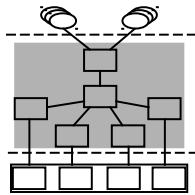
Storage  
Hierarchy  
(simplified)

9 - Troubleshoot

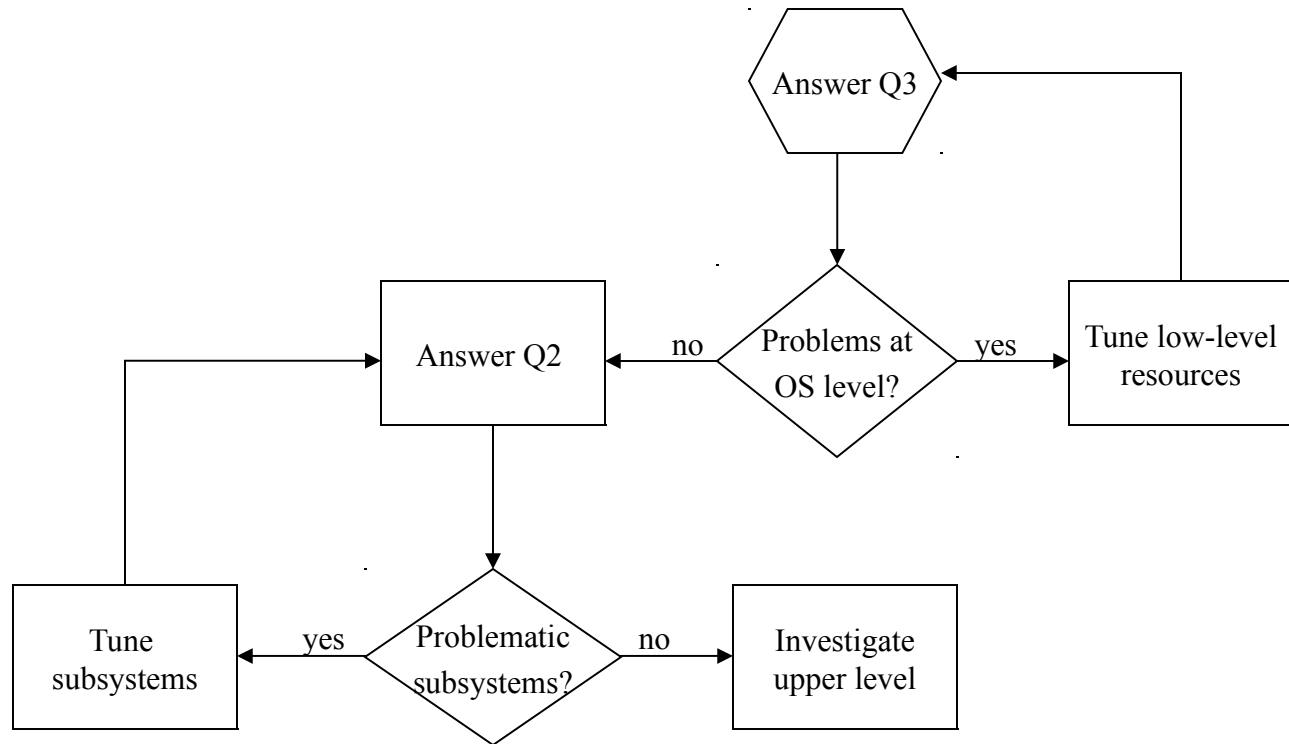
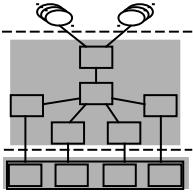
© Dennis Shasha,  
Philippe Bonnet, 2002

163

# Lock Manager Performance Indicators



# Investigating Intermediate and Primary Resources: Summary



# Troubleshooting Techniques

- Monitoring a DBMS's performance should be based on queries and resources.
  - The consumption chain helps distinguish problems' causes from their symptoms
  - Existing tools help extracting relevant performance indicators

# Recall Tuning Principles

- Think globally, fix locally (troubleshoot to see what matters)
- Partitioning breaks bottlenecks (find parallelism in processors, controllers, caches, and disks)
- Start-up costs are high; running costs are low (batch size, cursors)
- Be prepared for trade-offs (unless you can rethink the queries)