Cost Estimation:

- R.a = x => $X \cong \frac{1}{V(R,a)}$
- R.a < x => $X \cong \frac{x - min(R.a)}{max(R.a) - min(R.a)}$
- R.a > x => $X \cong \frac{max(R.a) - x}{max(R.a) - min(R.a)}$
- x1 < R.a < x2 => $X \cong \frac{x2 - x1}{max(R.a) - min(R.a)}$
- R.a = S.a (equijoin) => $X \cong \frac{1}{max(V(R,a), V(S,a))}$
- cond1 AND cond2 => $X = X_1 * X_2$

☆ 所有來自 R, S, T1, T2 ... 的 Tuple 數

Number of tuples returns after applying condition is T(RS)*X=T(R)*T(S)*X (since total number of tuples after cartesian product of R and S is T(R)*T(S).

In general, number of tuple output from a scan is original_number_of_tuple*X

Scan (read data) on a table R's cost is:

- Full sequential scan: B(R)
- Clustered index scan: X*B(R)
- Unclustered index scan: X*T(R)

Join B(R) is the number of blocks of table R, M is the number of block inside memory, cost is:

- Nested-Loop
  - Block at a time: B(R)+B(R)*B(S)
  - Nested-block-loop: B(R)+B(R)/(M-1)*B(S)
- Hash join (assuming B(R) < M or B(S) < M): B(R) + B(S)
- Sort-merge join (assuming B(R) + B(S) < M): B(R) + B(S)
- Index join:
  - Clustered on S: B(R)+T(R)(B(S)/V(S,a))
  - Unclustered on S: B(R)+T(R)(T(S)/V(S,a))

P.S.: 最前面 in B(R): full sequential scan of R into mem if already in mem, the cost can be removed.

2PL v.s. Strict 2PL

2PL:

- In every transaction, **all lock requests must precede all unlock requests**
- Ensure conflict serializability
- Might not be able to recover (Dirty Read: Read on some write that gets rolled back)

Strict 2PL:

- **Every lock each transaction is held until commit or abort**
- Ensure conflict serializability
- Recoverable as each transaction does not affect others until commit/abort

Conflicts

- Most application concurrency problems are describable by conflicts
- Lost Update → Write-Write (WW) conflict
- Dirty Read → Write-Read (WR) conflict
- Unrepeatable Read → Read-Write (RW) conflict
- Phantom Read → Read, Insert, Read. → Same Read has more rows

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantoms
Read Uncommitted	Allowed	Allowed	Allowed
Read Committed	Not Allowed	Allowed	Allowed
Repeatable Read	Not Allowed	Not Allowed	Allowed
Serializable	Not Allowed	Not Allowed	Not Allowed

← Table level locking

enforced:

Isolation Level	Read Lock Behavior	Conflict Example	Conflict Types
Read Uncommitted	No Read Locks	WR or RW conflict	"Dirty reads"
Read Committed	Short-Duration Read Locks	RWR conflict	"Unrepeatable reads"
Repeatable Read	Long-Duration Read Locks (Strict 2PL) + write locks	Phantom conflict	Phantom problem
Serializable	Predicate Locks / Table locks	Serializable	None

Element Granularity

- A DBMS (and sometimes user) may specify what granularity of elements are locked
  - Dramatically qualifies expected contention
- SQLite → Database locking only → phantom are impossible
- MySQL, SQL Server, Oracle, … → Row locking, table locking
- SQL syntax varies or may not exist explicitly

b) R2(B); R1(A); R3(D); R3(C); R2(D); W2(C); Co2; R3(D); W3(A); Ab3; W1(B); I1(D); R1(C); Co1;

None Read uncommitted Read committed Repeatable read Serializable

Serial schedule:

T2; T3; T1; or T2; T1; T3; or T3; T2; T1;

c) R1(A); R2(B); R3(D); R3(C); R2(D); I1(D); W2(C); Co2; R3(A); W1(B); W3(D); Co3; R1(C); Ab1;sdd

None Read uncommitted Read committed Repeatable read Serializable

Serial schedule:

N/A
This schedule contains R3(A) and W3(D) which were not part of T3.

Transactions copied here for your reference.

T1: R(A), W(B), I(D), R(C)
T2: R(B), R(D), W(C)
T3: R(D), R(C), R(D), W(A)

d) Does there exist a schedule of the above transactions that would result in a deadlock if executed under strict 2PL with **both shared and exclusive table locks**? If so write such a schedule with lock / unlock ops, and explain why the transactions are deadlocked. Otherwise write "No". Use L1(A) to refer to T1 locking table A, and U1(A) for unlocking.
Use S1(A) for grabbing shared lock, and X1(A) for exclusive lock.

S1(A); R1(A); X1(B); W1(B); S3(D); R3(D); S3(C); R3(C); R3(D); X1(D); I1(D); X3(A); W3(A); <deadlock>
T1 and T3 both hold shared locks on A and D but need to grab exclusive locks on the two elements in order to proceed.

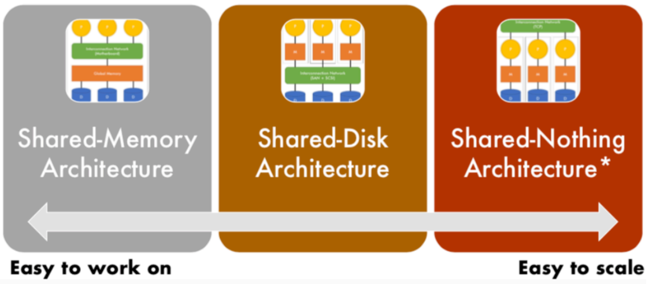
R(A) X(D)
R(D) W(A)

BCNF more important

妈的这感觉
好像冬天，
考试那天穿长袜

[shared-disk / mem / nothing : rarely talked
[parallel query process : map

Recap



- Shared Memory
  Nodes share both RAM and disk
- Shared Disk
  All nodes access the same disks
- **Shared Nothing***: **most scalable solution** and widely used in industry
  Each node has its own RAM and disk

- Inter-Query Parallelism
  Each transaction on a separate node
- Inter-Operator Parallelism
  Each operator on a separate node
- **Intra-Operator Parallelism***
  Each operator processed by multiple nodes

Horizontal Data Partitioning

- Block Partition

  Partition tuples (in arbitrary order) such that each node gets roughly the same amount

- Hash partitioned on attribute A

  Tuple t goes to node i based on i = hash(t.A) mod P

- Range partitioned on attribute A

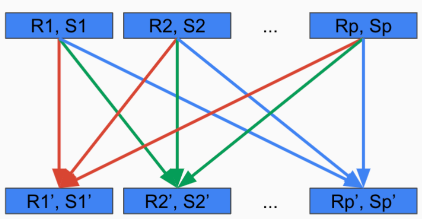
  Partition based on range of A into: -inf < v_1 < v_2 < ... < v_n < inf

Partitioned Hash-Join

We have p machines

We wish to join on some attribute (say R.x and S.y)

Call our hash function h(z)



Node 1

R1

P	X
1	20
2	50
3	30

S1

Q	Y
101	50
102	30

Node 2

R2

P	X
4	20
5	20

S2

Q	Y
201	20
202	50

Join on R.X = S.Y
hash(z): {20}->1, {30, 50}->2

Node 1

P	X
1	20
4	20
5	20

Q	Y
201	20

Node 2

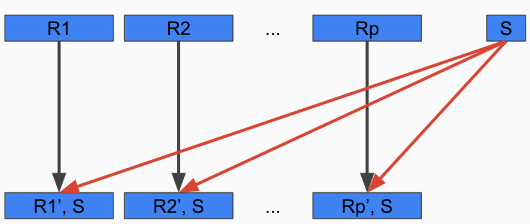
P	X
2	50
3	30

Q	Y
101	50
102	30
202	50

Broadcast Join

We want to think about how to prevent sending all data through the network.

Take advantage of small datasets (meaning the whole dataset can fit into main memory)




```
SELECT R.A
  FROM R, S
 WHERE R.A = S.A AND R.A > 10
 GROUP BY R.A
HAVING MAX(S.B) < 10
```