* Blocking factor [Bf]: Avg no of records / block.

→ $Bf = \frac{\text{Block Size}}{\text{Record Size}}$ (Block Size > Record Size)

Spanned Organisation	Unspanned Organisation
(1.) A record can be stored in more than 1 block.	Record can be stored in a particular block
(2.) $Bf = \frac{\text{Block Size}}{\text{Record Size}}$	$Bf = \left\lfloor \frac{\text{Block Size}}{\text{Record Size}} \right\rfloor$ floor
(3.) No memory wastage, but block access cost increased	Block access cost reduced but wastage of memory
(4.) Suitable for variable length records	Suitable for fixed length records

* Default is unspanned Organisation

Block factor: max possible records / blocks

→ for unspanned:

$\left\lfloor \frac{B-H}{R} \right\rfloor$ records / block

H = header size.

→ for Spanned:

$= \frac{B-H}{R}$

Ordered file.	Unordered File
1. Searching easy.	Insertion easy.
2. Insertion expensive.	Searching expensive
3. To access any no of blocks = $\lceil \log_2 B \rceil$	Linear search: Avg case = $\frac{B}{2}$ Worst case: B.

One Index Record size = Size of search key + Size of pointer.

data blocks = $\left\lceil \frac{\text{\# records}}{\text{Bf of DB file}} \right\rceil$

Dense Index	Parse Index
* Index entry created for every search key value.	Index entry created for some search key values
* No. of index entries = no of DB records.	No of index entries = no of blocks

→ Bf of Index size

$\left\lfloor \frac{\text{Block size}}{\text{Index record size}} \right\rfloor$

→ Block factor of index = $\left\lfloor \frac{B+H}{key} \right\rfloor$ entries/block.

Single level ordered indices:

1. Primary index (key + ordered file)
2. Clustering index (non key + ordered file)
3. Secondary index (key/non key + ordered/unordered file)

↓ Atmost 1 primary index / relation, and ∑1 Secondary index possible (per)

↓ In a relation either CI or PI but not both

→ Access cost, PI with multilevel index = (k+1) blocks

→ To access a record using index avg no of block access = $\lceil \log_2 B_{index} \rceil + 1$

→ 'n' level in multilevel index, # block access to search a record = n+1

B-Tree

P: block/child/index pointer

Pr: read/record/data pointer

K: Search key.

Order p: max possible child pointers that can be stored in node

→ Every internal node: (min) $B_p = \lceil p/2 \rceil$, min keys $(\lceil p/2 \rceil - 1)$ (except root)

max $B_p = p$, max keys $(p-1)$

→ Root can contain atleast 2 B_p (min 1 key)

→ ~~All leaf at same level~~ Each leaf node should be at same level.

→ Order: p: by default max (p = block pointer)

(~~Balanced tree~~) $= p * B_p + (p-1)(key + R_p) \leq$ Block Size

→ Order: p (key), (key order: p)

$= (p+1)Bp + p(key + Rp) \leq$ Block Size

Order p: Minimum

Height / Level	Min nodes	Min # Bp	Min # keys
$h / h+1$	$2\lceil \frac{p}{2} \rceil^{h-1}$	$2\lceil \frac{p}{2} \rceil^{h}$	$2\lceil \frac{p}{2} \rceil^{h-1} (\lceil p/2 \rceil - 1)$

Order p: Maximum

Height / Level	Max nodes	Max # Bp	Maximum keys
$h / h+1$	p^h	p^{h+1}	$p^h(p-1)$

Total minimum keys: $(1 + 2(\lceil p/2 \rceil - 1) + 2\lceil p/2 \rceil (\lceil p/2 \rceil - 1) + \ldots + 2\lceil p/2 \rceil^{h-1} (\lceil p/2 \rceil - 1))$ → GP formula.

Total maximum keys: $(p-1) + p(p-1) + p^2(p-1) \ldots p^h(p-1)$ → GP series

B^+ Tree.

→ All keys are available at leaf node

→ Internal node: no record pointer.

→ Leaf node contains key, and $Rp + 1$ Block pointer

Order of internal node: $p \times Bp + (p-1)$ key $\leq$ Block Size.

Order of leaf node: $(p-1)(key + Rp) + 1\,Bp \leq$ Block Size

→ Split of internal node is same as in B tree, but split of leaf is different from B^+ tree

→ B^+ tree best suited for sequential access of range of records

Procedural Query language: Relational Algebra

Non Procedural QL: SQL and TRC, and DRC

Relational Algebra

* Query executed on DB tuple by tuple. One tuple at a time.
* Generated distinct tuples.
* Takes relational instances as i/p and returns relational instances as o/p.

Basic Operators

1. Selection (σ)
2. Projection (π)
3. Cross product ($\times$)
4. Union ($\cup$)
5. Set difference ($-$)
6. Rename (ρ).

Derived Operators

1. Join and its types ($\bowtie$)
2. Division ($/$)
3. Intersection ($\cap$)

* To apply set operators ($\cup, \cap, -$), relations must be Union Compatible:
  - → arity of two relations must be same
  - → range (domain) of attribute must be similar

Cross Product ($\times$)

$R \times S = n_1 \times n_2$ tuple,

$C_1 + C_2$ attribute.

(for cross product, relations not required to be union compatible)

→ Cross Join = Cross product ($\times$)

→ Natural Join = Natural Cross product
- either join-condition is empty
- no common attribute

→ $\frac{\pi_{AB}(R)}{\pi_A(S)} = B$, $\frac{\pi_{AB}(R)}{\pi_B(S)} = A$

Inner Joins

1. Conditional Join ($\bowtie_C$)
2. Equi Join
3. Natural Join

Outer Joins

1. left outer join ($\bowtie$ or $\overset{\circ}{\bowtie}_L$)
2. Right outer join ($\bowtie$ or $\overset{\circ}{\bowtie}_R$)
3. Full outer join ($\bowtie$ or $\overset{\circ}{\bowtie}$)

Structured Query Language

- DDL : Create, Alter, Drop table
- DML : Insert Update, Delete
- DCL : Commit, Abort.
- DQL : Select, From, Where

→ SQL is not case sensitive.

→ Mandatory SQL clause are "SELECT" and "FROM"

→ SQL works on multi set, retains duplicates

SQL Clauses

1. Select
2. From.
3. Where
4. Group By
5. Having
6. Order By, etc

Aggregate functions

1. ~~Count~~ COUNT
2. ~~Sum~~ SUM
3. ~~Avg~~ AVG
4. ~~Max~~ MAX
5. ~~Min~~ MIN

* Aggregate functions Cannot be lowercase

NULL : not existed / unknown

* Arithmetic Operations with "NULL" give NULL
* Aggregate functions ignores NULL.
* Comparision with NULL give value 'unknown'

Set Operations

1. Union & Union all
2. Intersect & intersect all
3. minus & minus all (except, except all)

→ When aggregate Operator and other attribute is used in Select clause, it allowed only if attribute will be in Group by clause

Regular Expressions

- % : Zero or more Characters
- _ : exactly one character.

A% → Start with A

%J → End with J

%I% → Contains I.

'___' → All 3 length

'S___' → Start with S, Exactly 4 char.

'S___%' → Start with S, min 4 char.

Tuple Relational Calculus

Query described in the form of tuples.

Tuple variable: Variable that takes on tuple of a relation schema as value.

Domain Relational Calculus

Query described result is the form of columns (Domain)

Domain variable: Variable that range over the domain of some attribute

Entity Set: Collection of similar entities that share the same properties.

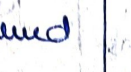
Relationship: associations among several entities.

Attributes: properties used to describe an entity.

Attribute Types are:

1. Simple and Composite
2. Single and multivalued
3. Stored and derived
4. key
5. Complex
6. Descriptive.

Mapping Cardinality Constraints

- One to one ()
- One to many ()
- Many to one ()
- Many to many ()

Referencing relation: Table with foreign key (child table)

Referenced relation: Table referenced by foreign key (parent table)

→ Deletion from 'referenced' relation and insertion into 'referencing' relation may violate foreign key constraint.

→ Foreign key may contain duplicate and null value.

ER to DBMS Conversion

ER	RDBMS
(1.) One : one	2 Table req.
(2) 1 : m.	2 Table req.
(3) m : m	3 Table req
(4) m : 1.	2 Table req.

Partial participation at both side with above mapping are same as above.

- Partial participation (—)
- Total/Full participation (═)
- If Total participation at both side in (1:m, m:1, 1:1, and m:m) then 1 table required.
- Tp at one side in (1:m, 1:1, and m:1) then Only 1 table required
- Tp at any side in m:m then 2 table required

ACID Properties

1. Atomicity
2. Consistency
3. Isolation
4. Durability.

Schedule: Sequence of instructions that specify the chronological order in which instructions of current transactions are executed

Serial schedule: Executes one by one

Non Serial schedule: interleaved executions of two or more transactions.
(may/may not be consistent)

→ for 'n' transactions, n! Serial schedules possible

Conflict Serializable: transforming non serial to serial schedule by swapping non conflicting instructions.

T_i	T_j
R(A)	W(A)
W(A)	R(A)
W(A)	W(A)

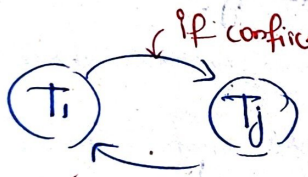
(Swapping not possible)

T_i	T_j
R(A)	R(A)
R(A)	W(B)

(Swapping possible)

Conflicting Instructions

- I_i = read(Q), I_j = write(Q)
- I_i = write(Q), I_j = read(Q)
- I_i = write(Q), I_j = write(Q)



(if conflict arises, we draw an arc)

(Forms, then not conflict Serializable)

Precedence Graph Method → CNC

→ If schedules S_1 and S_2 are conflict equal then their precedence graphs will be same

→ A schedule is Serializable if it either conflict/view serializable or both

→ If a schedule is conflict Serializable then it already view serializable, but vice versa not true.

View Serializability.

* If in schedule S, T_i reads initial value of Q, then in $\bar{S}$ also T_i must read the initial value of Q.
* If in S, T_i executes read(Q), and that value was produced by T_j, then in $\bar{S}$ also transactions must read the value of

Q that was produced by same write(Q) operation of T_j.

↓ The transaction that perform final write(Q) in S_1 must also perform final write(Q) in S_2.

View Equivalence of S_1 and S_2

(1.) Initial reads of S_1 and S_2 should be same.

(2.) Final updation of every data item should be same in S_1 and S_2

(3.) Write-read sequence should be equal.

Problems due to Concurrent Execution:

1.) WR / Uncommitted read / dirty read

2.) R-W / Un-repeatable read problem

3.) W-W / lost update

4.) Phantom tuple problem

→ Non-Serial Schedule = Total concurrent Schedule − Serial schedule ($n!$)

(n = no. of transaction)

Recoverable Schedule: T_j reads a data item previously written by T_i then "commit" of T_i appears before commit of T_j.

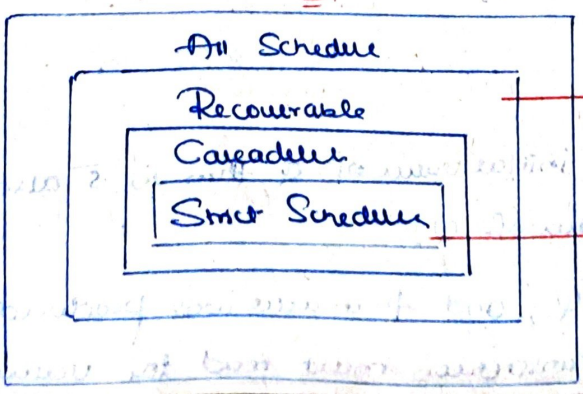
Finding no of Conflict Serializable

1. Write down all operations of following (later) transactions.
2. Start from last operation of first transaction and try to put at correct place such that conflict operations order must be maintained same as transaction order.

eg: $(T_1 \to T_2)$ — T_2 .. — T_2 — ...

(1. 2. 3.)

T_1 - last of (T_1), check for conflict.



Types of Lock

1. Shared lock : only read.
2. Exclusive lock : write / write / read
   read / write.

2 Phase locking Protocol

- Growing phase (Aquire locks)
- Shrinking phase (Release locks)

Lock point : position of last lock operation / first unlock operation

2PL

- Ensures conflict Serializibility.
- Serializability order determined by lock point.
- 2PL not Ensure recoverability.
- 2PL suffers from deadlock (may / may not).
- 2PL suffers from Starvation

Strict 2PL

- 2PL + All exclusive locks taken by the transactions must be held until commit / rollback.
- Ensures : Conflict serializable, recoverable schedule, Cascadless, Strict recoverable
- But suffers from deadlock and Starvation

Rigorous 2PL

- 2PL + All locks (shared & exclusive) must be held by transactions until commit / rollback.
- Suffers from deadlock & Starvation.

Conservative 2PL : Each transaction acquire all the locks in the beginning of execution and release after commit

- Ensures : Conflict serializability, recoverability, cascadless, Strict recoverable, no deadlock.
- But suffers from Starvation

Time Stamp Protocol

- Unique time stamp value assigned to each transaction when they arrive in the system.
- Based on time stamp determines Serializability order
- Ensure : Conflict Serializable, free from deadlock but not recoverable & Starvation

Thomas Write Rule

* Modified version of time stamping protocol.
* When T_i attempts to write data item Q, if $TS(T_i) < W\text{-}TS(Q)$ then T_i is attempting to write an absolute value of {Q}.
  - Rather than rolling back T_i at the TSP.
* Allows view Serializable schedules that are not confict Serializable.

Strict Time Stamp = TSP + Strict schedule

Ordering protocol

↳ **Ensures**: Serializability, recoverability, cascadeless, Strict recoverability.

RDBMS

↓

Relation

↓

Table

↙ ↘

Row. (Tuple/record) Columns (field/attribute)

Arity: no of attributes. (degree) (Columns)

Cardinality: no of tuple/records (rows)

Functional Dependency

X, Y — attribute set of relation R

t_1, t_2 — any two tuple.

⇒ If $t_1.x = t_2.x$ then $t_1.y = t_2.y$ (must be same)

Trivial FD (Always valid)

$X \to Y$, $X \supseteq Y$

Non Trivial FD

$X \to Y$, $X \cap Y = \phi$, and $X \to Y$ (must satisfy)

Semi Non Trivial FD

$X \to Y$, $X \not\supseteq Y$ and $X \cap Y \neq \phi$.

Reflexive Rule: $X \supseteq Y$ and $X \to Y$.

Augementation rule: $X \to Y$, then $XZ \to YZ$

Transitive rule: $\{X \to Y, Y \to Z\}$ then $X \to Z$.

Decomposition: $\{X \to YZ\}$, then $X \to Y$, $X \to Z$

Union: $\{X \to Y, X \to Z\}$, then $X \to YZ$

Super key: If 'x' is a attribute set of a relation R, then if $[x]^+$ determined all attributes of R, then x is superkey

Candidate key: minimal of Super key is called candidate key.

Prime attributes: attributes present in any candidate key (not all CKs)

→ If X attribute → key/prime attribute, then multiple candidate keys will be present.

Equality b/w 2 FD set

F[...], G[...]

They are equal if F cover G, and G cover F

F cover G: F covers all FDs of G FD set.

G cover F: G covers all FDs of F FD set.

Finding Minimal Cover

(1.) Split the FD such that RHS contain single attribute.

(2.) Find redundant attribute on LHS and delete them.

(3.) Find redundant FD and delete from FD set.

* Minimal cover can be ≥ 1

→ Maximum no. of candidate keys = $^{n}C_{\lfloor \frac{n}{2} \rfloor}$ (n = no of attributes)

→ If every single attribute is Candidate key, max superkys = $2^n - 1$

Lossless Join Decomposition

If $R_1 \bowtie R_2 \bowtie R_3 \ldots R_n = R$

Binary Method: $R_1 \bowtie R_2$ is lossless, if

* $R_1 \cup R_2 = R$
* Common attribute of R_1 and R_2 is either Superkey of R_1/R_2

Dependency Preserving Decomposition

If $F_1 \cup F_2 \cup F_3 \cdots \cup F_n = F$

then dependency preserving.

Normal forms

1. 1NF
2. 2NF
3. 3NF
4. BCNF

* Every higher normal forms contain lower normal form

Redundancy level: 1NF > 2NF > 3NF > BCNF

First Normal Form: (1NF)

* Relation doesn't contain any multi valued attribute.
* By default RDBMS it is 1NF.

Partial Dependency

$X \to Y$ is partial FD

if. $A \in Y$ and

$(X - A) \to Y$.

Second Normal Form (2NF)

* Every non attribute in R is fully functional dependent on Primary key of R
* Proper subset of candidate key $\not\to$ non key attribute

Third Normal Form (3NF)

* Every $X \to Y$ non trivial FD must satisfy:

$X \to Y$,

$\Rightarrow$ X: super key $\cup$ Y: prime attribute

BCNF

Every non trivial FD must satisfy: $X \to Y$

X : Super key.

* If 2 attributes in a relation then always — BCNF

	1NF	2NF	3NF	BCNF
0-% Redundancy	x	x	x	✓ (suffers from multivalued attributes)
Lossless join	✓	✓	✓	✓
Dependency Preserving	✓	✓	✓	(may/may not)