Relation Algebra

Basic operators:
Projection (π), Selection (σ), Cross-product ($\times$)
Union ($\cup$), Rename (ρ), Set-difference ($-$)

Derived operators:
Join ($\bowtie$), Intersection ($\cap$), Division ($/, \div$)

Projection (π): used to select columns.
Selection (σ): used to select rows.
Cross-product ($\times$): Make combinations of each row in A with every row in B.

Degree $(A \times B) = D(A) + D(B)$
Cardinality $(A \times B) = C(A) \times C(B)$

Set-difference ($-$): $A - B \Rightarrow$ In A but not in B
$A - B \neq B - A$, (It is not commutative)

Union: ① No. of columns must be same.
② Domain of every column must be same.

Degree $(A \cup B) = D(A) = D(B)$
Cardinality $(A \cup B) = C(A) + C(B)$

Division: $A(X, Y) / B(Y)$ = It results x values for that there should be a tuple $\langle x, y \rangle$ for every Y value of relation B. eg. Find Sid of students who are enrolled in all courses.

$$\pi_{sid}(Enrolled) - \left[\pi_{sid}\left(\left(\pi_{sid}(Enrolled) \times \pi_{cid}(Course)\right) - Enrolled\right)\right]$$

Indexing

Indexing is used to Reduce I/O cost i.e reducing no. of blocks moduled to RAM.

Q. Block size = 1000 Bytes (without indexing)
Record size = 250 Bytes
Total records = 10,000 (unordered)
Avg. time complexity to search = ?

No. of records we can put in one block = $\frac{1000\ (BS)}{250\ (RS)} = 4$ records

No. of blocks required = $\frac{10,000}{4} = 2500$

I/O cost; Best case = 1
Worst case = 2500
Avg. case = $\frac{2500}{2} = 1250$

In case of ordered data binary search can be used, then time complexity = $\log_2 2500 = 12$

Q What is avg time complexity to search a record from index table if index table entry = 20 B (key + pointer)
10B 10B

Block size = 1000 Bytes
Index table entry = 20 Bytes
No. of records that we can put in one block of index table = $\frac{1000}{20} = 50$ records

No. of index blocks required (sparse) = $\frac{2500}{50} = 50$

Time complexity (I/O) = $\lceil \log_2 50 \rceil = 6 + \underbrace{1}_{\text{read actual block}} = 7$

If Dense then time, = $\frac{10,000}{50} = 200$ records Time = $\lceil \log_2 200 \rceil + 1 = 8 + 1 = 9$

Unordered data → Dense index → record for each entry
Ordered data → Sparse index → record for each block.

Types of Indexes

1) Primary
2) Clustered
3) Secondary

	Key	non-key	
Ordered file	Primary Index (sparse)	Clustered Index (sparse)	At most 1
Unordered file	secondary Index	Secondary Index	

Primary Index

- Key should be primary key and file must be ordered.
- Sparse index is generally used i.e one pointer for each block.
- First record of each block is called anchor record.
- No. of entries in index table = no. of blocks in harddisk.
- Search time = $\log_2 N + 1$ where N is no. of blocks in index table.

Clustered Index

- Non key but data is sorted.
- one ~~rent~~ entry in index table for each unique value.
- Sparse
- Block hanker is used in a case when a repeating value spans more than one block.
- Time = $(\log_2 N + 1)$ ~~is~~ → minimum, may exceed because of block hanker.

Secondary Index

- When data is unordered (key or nonkey)
- In a case when column is key (unique) but not ordered, Dense index is created which is sorted, so binary search can be used.
- No. of records in index = no. of records in disk.
- Time $\log_2 N + 1$ where N = no. of ~~reto~~blocks in Index table.

- In a case when column is neither key nor ordered, index is created for each unique value and is sorted.
- In addition a 'block of record pointers' is maintained, which points to all duplicate value records. (Intermediate layer).
- Generally dense because of Intermediate layer.
- Min. time = $(\log_2 N + 1 + 1)$.

Normalization

It is a technique to ~~redundancy~~ remove or reduce redundancy from a table.
Insertion anomaly, deletion anomaly and updation anomaly.

First normal form:
Table should not contain a multivalued attribute.

Closure method: used to find all primary keys of the table.
R(A B C D E)
FD = {A → B, BC → D, E → C, D → A}

E^+ = EC (E wasn't present on right hand side)
AE^+ = AEBCD — CK = {AE, DE, BE}
DE^+ = DEACB — Prime att. = A E D B
BE^+ = BECDA — Non-Prime att. = C
CE^+ = CE x

Functional Depedency:
X → Y
X determines Y.

[For same value of determining attribute, value of determined attribute should be same]

Reflexivity: If Y is a subset of X then X → Y (Trivial)
Augmentation: If X → Y, then XZ → YZ
Transitive: If X → Y and Y → Z then X → Z
Union: if X → Y and X → Z then X → YZ
Decomposition: If X → YZ then X → Y and X → Z
Pseudotransitivity: If X → Y and WY → Z then WX → Z
Composition: If X → Y and Z → W then XZ → YW

Second normal form

1) Table ora relation must be in 1st normal form.
2) there should be no partial dependency i.e all non prime attribute of a table should be fully functional dependent on candidate key.

R(ABCDEF)
FD {C → F, E → A, EC → D, A → B}
EC^+ = ECDFAB
CK = {EC}
Prime attribute {E, C}
Non-prime attribute {A, B, D, F}
Here F is determined by C i.e a part of primary key.
Hence, this table is not in 2nd normal form.

Partial dependency: L.H.S should be a proper subset of candidate key and R.H.S should be a non prime attribute.

third normal form

1) The relation must be in second normal form
2) there should be no transitive dependency i.e a non prime attribute should no be determined by a non prime attribute.
3) For each FD, L.H.S must be a candidate key or super key or R.H.S is a prime attribute.

R(ABCD)
FD {AB → CD, D → A}
CK AB^+ = ABCD
Since AB is candidate key in (AB → CD) and A is a prime attribute in (D → A), the relation is in third normal form.

BCNF (Boyce Codd Normal Form)

L.H.S of each FD should be a candidate key or a super key.

- Third normal form always ensures 'Dependency preserving decomposition' but not in BCNF.
- Both third and BCNF ensures lossless decomposition.

R(ABCD)

FD{AB → CD , D → A}

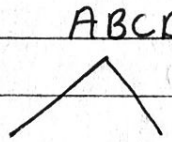
$AB^+ \rightarrow ABCD$

$DB^+ \rightarrow DBAC$

CK = {AB, BD}

PA = {A, B, D}

NPA = {C}



{D → A}

$BD^+ = BDAC$

{BD → C}

Here the relation is in third normal form but not in BCNF because in (D → A) 'D' is not candidate key. Hence we take out D^+ i.e DA from relation.

Also D → A is preserved but AB → CD is not. Hence we have to make new dependency BD → C.

In lossy decomposition, spurious tuples gets generated. To prevent that the common attribute should be a ~~prime~~ candidate key or super key of either R_1 or R_1 or both.

If A is the candidate key: ~~for~~ in R(ABC), then R_1(AB), R_2(AC)

1) $R_1 \cup R_2 \equiv R$

$AB \cup AC \equiv ABC$

$ABC \equiv ACC$

2) $R_1 \cap R_2 \neq \phi$

$AB \cap AC \neq \phi$

$A \neq \phi$

3) R_1 CK or R_2 CK or both.

Equivalence of functional dependency

If X covers Y $X \supseteq Y$

& If Y covers X $Y \supseteq X$

$\Rightarrow X \equiv Y$

$X = \{A \to B, B \to C\}$, $Y = \{A \to B, B \to C, A \to C\}$

If X covers Y

In X

$A^+ = AB$ (Reflexive)

$A^+ = AB$ (Reflexive)

$A^+ = ABC$ (Transitive)

So, A is covering $A \to B$ & $A \to C$

In X

$B^+ = B$ (Reflexive)

$B^+ = BC$ (Reflexive)

So, B is covering $B \to C$

Hence X covers Y

If Y covers X

In Y

$A^+ = ABC$

So A is covering $A \to B$

In Y

$B^+ = BC$

So, B covers $B \to C$ in X

Hence Y covers X.

$\therefore X \supseteq Y$

$Y \supseteq X$

$\Rightarrow \boxed{X \equiv Y}$

B-Tree

- Block pointer points to other node in tree.
- Data/Record pointer points to actual record stored in harddisk.
- For a tree with order p
  - No. of block pointers = p
  - No. of max. children = p
  - Keys = p-1
  - Record pointers = p-1
- Keys are inserted in sorted order

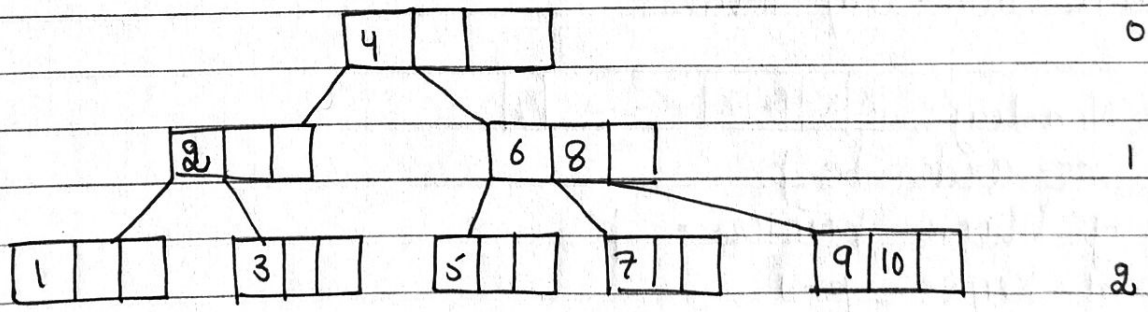
	children	Root	Intermediate Node
Max	p	p	
Min	2	$\lceil p/2 \rceil$	

Q) Insert following keys into B-tree, if order of B-Tree = 4.

~~Asc~~ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Max keys = $p-1 = 3$

Min keys = $\lceil \frac{p}{2} \rceil - 1 = 1$



Q) Consider a B-Tree with key size = 10 bytes, block size 512 Bytes, data pointers of size 8 bytes and block pointer is 5 bytes. Find the order of B-Tree?

Ans. Let ~~there be~~ order of tree be p

no. of block pointers = p

no. of keys and data pointers = p-1

So, $p \times B_p + (p-1)\,key + (p-1)R_p \leq$ Block size

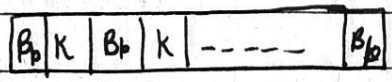
$5p + (p-1)(10+8) \leq 512 \Rightarrow 23p \leq 530$

$p \leq 23.04$

Min: $\lceil \frac{p}{2} \rceil = 12$ i.e 24 B_p or children so $\boxed{p = 23}$ keys = 23-1 = 22.

B-Tree	B^+-Tree
1) Data is stored in leaf as well as internal nodes.	1) Data is stored only in leaf node.
2) Internal nodes have key and record pointers.	2) Record pointers are not there due to which more keys can be stored.
3) Searching is slower and deletion complex.	3) Searching is faster and deletion easy (directly from leaf node).
4) No redundant search key present	4) Redundant keys may present.
5) Leaf nodes are not linked together.	5) Linked together like linked list.
6) Leaf node have Null ~~B_p~~ B_p	~~6) B_p are not present at leaf~~ 6) Only one B_p for linking.

Q Consider a B^+ Tree with key size = 10 bytes, block size = 512 Bytes, data pointers = 8 bytes and block pointer = 5 bytes. What is the order of leaf and non-leaf nodes?

For Non-Leaf 

Let the order be p.

No. of block pointers = p

No. of keys = p - 1

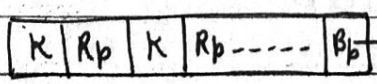
So, (key size) × (p − 1) + p × B_p size ≤ 512

$10(p-1) + 5p \leq 512$

$15p \leq 522$

$p \leq \frac{522}{15}$ 34.8

so $\boxed{p = 34}$

For Leaf 

$x(K + R_p) + B_p \times 1 \leq$ ~~512~~ BS

$x(10 + 8) + 5 \leq 512$

$x \leq 28.16$ or $\boxed{x = 28}$

Let x = order of leaf node i.e max no of (key, R_p) pair