DBMS BASICS

*) Disadvs of File System

1) Data redundancy
2) Data inconsistency
3) Diffucalty in accuracy of data
4) Data isolation problem
5) Security problem
6) Atomicity problem
7) Concurrent
8) Integrity problem.

OLAP (Online Analytical problem)	OLTP (Online transaction processing)
→ Historical data	→ Current data
→ Subject oriented	→ Application oriented
→ Decision making	→ Day to day operation
→ TB, PB size	→ MB, GB size
→ CEO, MD, GM uses	→ Clerks, managers uses
→ Read oprtns	→ Read/Write oprts.

Date:

ER DIAGRAM

→ Entity Relationship Diagram.

① Entity: Thing/obj in real world that is distinguishable from other objs based on attributes/values it posseses.

② Types of Entities

→ Tangible: Entity exists physically
→ Intangible: Entity exists logically

③ Entity set: Collection/set of similar entities.

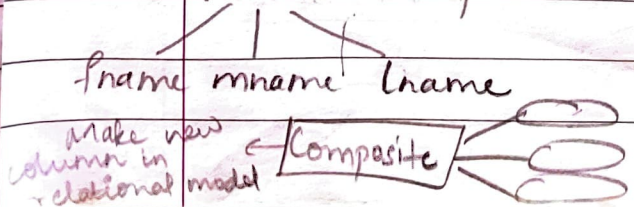
→ Entity sets are represented in the ER diag instead of the indv entity
→ Represented by tables in Relational model

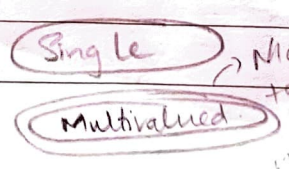
→ Attributes

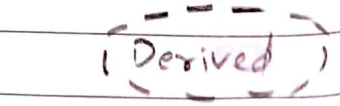
⊛ The units that describe characteristics of entities
- Every attribute there is set of permitted values called domain.

→ Type of Attributes

Simple - composite	Single - multivalued	Stored - derived
- Simple ✗ divided	- Single only one value in 1 instance	- Stored new values
- Composite ✓ divided	- Multivalued >1 value in instance	→ derived we find it Eg:- Age uses stored attributed
Eg:- Roll-no → Simple		
Name → Composite		







→ Relationship in ER diagram

(*) Associatⁿ betwⁿ ≥2 entities of same/diff entity set.

(1) Relationship set: A set of similar type of relationships.

Relationship has 3 components
(i) Name (ii) Degree (iii) Cardinality rate / Participation const

(2) Degree: No: of ~~associat~~ entity set participating in the associatⁿ.

- Min. degree can be unary not necessarily binary
- Most relationships are binary.

(3) Cardinality ratio: Express / No of entities to which other entity can be related by relationship.

1:1, 1:M, M:1, M:M.

→ Participation constants

(*) Specifies whether the existence of an entity depends on it being in a relationship with another entity & its type.

(*) Also denotes max & min no: of relationship instance that each entity must depend on.

- Max. cardinality, Min. cardinality
- Write in brackets (min, max)

→ Participation

(1) Total participatⁿ (═══) Double line.

(2) Partial participatⁿ

→ Functional Dependency

$\alpha \rightarrow \beta$ ⇒ Given the value of 'α' you can SEARCH value of 'β'.

if $t_1[\alpha] = t_2[\alpha]$
then $t_1[\beta] = t_2[\beta]$

Types ($\alpha \rightarrow \beta$)

① Trivial ~~Non~~

$AB \rightarrow A$
if $\beta \in \alpha$

② Non-Trivial
if $\beta \notin \alpha$, $\beta \notin \alpha$
$AB \rightarrow ABC$

→ Attribute Closure / Closure of attribute set

An attribute set 'A' can be defined as a set of attributes which can be functionally determined from it. Denoted by F^+

Eg:- $A \rightarrow B$ $B \rightarrow C$ ⇒ $A \rightarrow C$
$A \rightarrow A$ $A^+ \rightarrow (ABC)$ ∴ $A^+ = (ABC)$

→ Armstrong rules / Axioms

- Axiom is a statement that is taken as true & serves as a premises / starting pt. for further arguments.
- Can be used to find functional dependency.

⊛ Primary rules (RAT)

1) Reflexivity : if $Y \in X$ then $X \rightarrow Y$
2) Augmentation : If $X \rightarrow Y$ then $XZ \rightarrow YZ$
3) Transitivity : If $X \rightarrow Y$ && $Y \rightarrow Z$
then $X \rightarrow Z$

Page No.:
Date:

(*) Secondary rules

1) **Union**: If $X \to Y$ & $X \to Z$ then $X \to YZ$

2) **Decomposition**: If $X \to YZ$ then $X \to Y$ & $X \to Z$

} Apply only on R.H.S

3) **Pseudo transitivity**: If $X \to Y$ & $WY \to Z$ then $WX \to Z$

} Can be proved using augmentation & transitivity

4) **Composition**: If $X \to Y$ & $Z \to W$ then $XZ \to YW$

→ Canonical Form / Irreducible set of f.d

$R(WXYZ)$

$X \to W$

$WZ \to XY$

$Y \to WXZ$

Step 1: decompositn

$X \to W$

$WZ \to X$ x

$WZ \to Y$

$Y \to W$

$Y \to X$

$Y \to Z$

$X^+ = XW$ (With 1st f.d)

$X^+ = X$

$WZ^+ = WZXY$

$WZ^+ = WZYX$

$WZ^+ = WZYX$

$WZ^+ = WZ$

$Y^+ = WXYZ$

$Y^+ = YXZW$

$Y^+ = YZ$

$Y^+ = YXW$

CANONICAL FORM:

$X \to W$	$WZ^+ = WZYX$	$Y \to W$
$WZ \to Y$	$W^+ = W$	$WZ \to Y$
$Y \to X$	$Z^+ = Z$	$Y \to XZ$
$Y \to Z$		

R(ABC)	A → B	A^+ : AB	A → B	AC^+ : AC
A → B	C → B	A^+ = A	C → B	A^+ : AB
C → B	D → A	C^+ = CB	D → A	C^+ : CB
D → ABC	D → B	C^+ : C	D → C	
AC → D	D → C	D^+ : ABCD	AC → D	
	AC → D	D^+ : DBC		
		D^+ = DABC		
		D^+ : DAB		
		AC^+ : ACDB		
		AC^+ : ACB		

CANONICAL COVER:

A → B	D → AC
C → B	AC → D

→ Types of keys

$(Key)^+ = R$

Key is a set of attribute which can uniquely identify a row/tuple in a relation/table.

① Superkey

$(S.K)^+ = R$

② Candidate key : Efficient/minimal version of superkey

③ Primary key : Any one of the candidate key and it unique in a relation.

Normalisation

NEED

(1) Redundancy
→ Unnecessary repetitⁿ of data
→ Insertⁿ, deletⁿ, modificatⁿ anomalies

(2) Data Inconsistency
→ Same data when stored in multiple locations and value at all the locatns isnt the same

(1) FIRST NORMAL FORM

→ Every cell should contain atomic value.
- Trivial NF
- All values in same column should be in same domain

(2) SECOND NORMAL FORM

- Prime attribute: Attributes in the candidate key
- Non-prime attribute: Attributes X in candidate key
- Partial dependency: When non-prime attribute instead of depending on entire C.K depends on the prime attribute

Condⁿˢ: (1) Should be in 1NF
(2) Should X have partial dependency.

③ THIRD NORMAL FORM

Condns: ① Be in INF, 2NF
② No partial dependency & transitive dependency

→ (*) For every f.d $\alpha \rightarrow \beta$
① either α is S.K
② ~~β~~ β is a prime attribute
or

④ BCNF

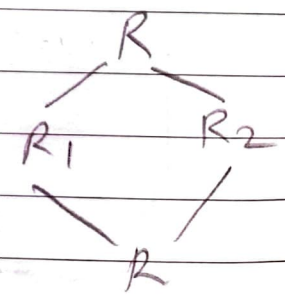
Condns: ① Is INF, 2NF, 3NF
② X partial, transitive dependency

If $\alpha \rightarrow \beta$ (P/np, P) → considered in BCNF
(*) $\alpha \rightarrow \beta$ then, $\alpha \Rightarrow$ S.K (should be) } ③rd condn

From INF → 2NF → 3NF } f.d preservative
While 3NF → BCNF } f.d non-preservative

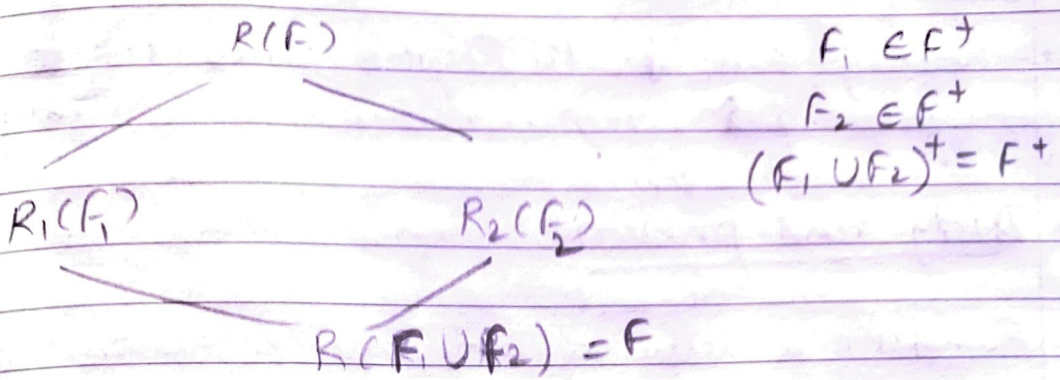
→ Lossless Decomposition / Non-additive decomposition

- This property garantees that the extra
- Child tables should have a common attribute otherwise # it will be lossy.



① attri $(R_1) \cup$ attri (R_2) = attri (R)
② attri $(R_1) \cap$ attri $(R_2) \neq \phi$
③ attri $(R_1) \cap$ attri $(R_2) \rightarrow$ attri (R_1)
attri $(R_1) \cap$ attri $(R_2) \rightarrow$ attri (R_2)

Dependency preserving property



→ Transaction

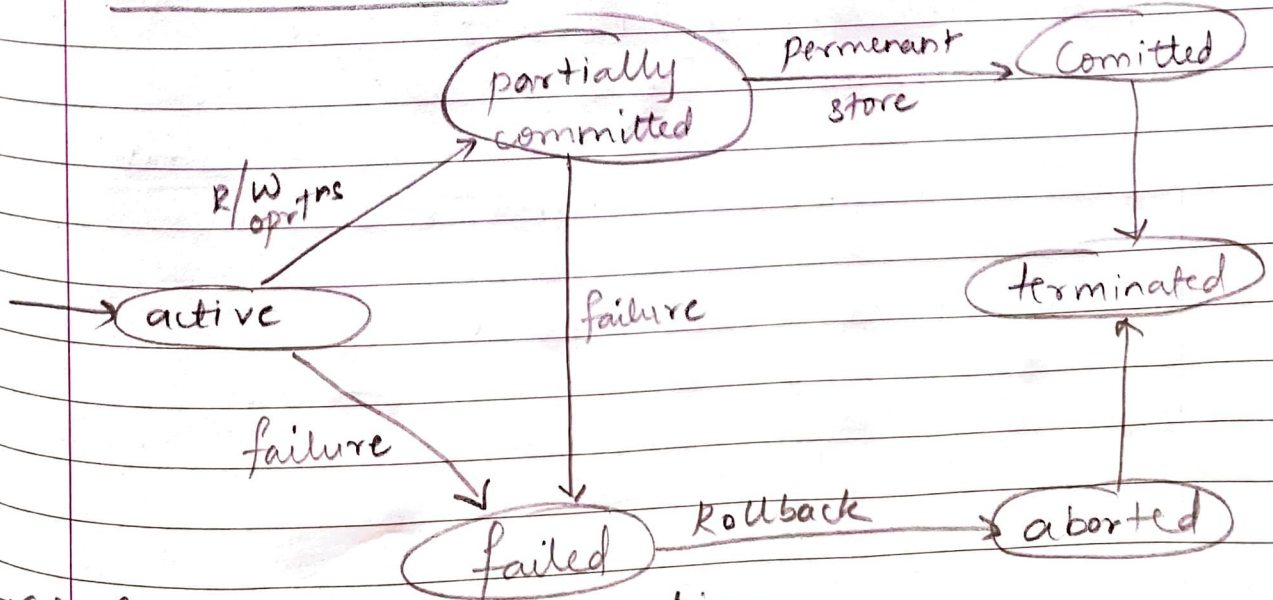
- A set of ~~logical~~ instructions that perform a logical unit of work
- Atomic in nature always.

→ ACID Properties

Ensures the prop

1. Atomicity → Transaction management component ensures atomicity
2. Consistency
3. Isolation → Concurrency control component
4. Durability → Recovery management component

→ Transaction states



Compensating transaction.

Page No.:
Date:

Advs of concurrency

(1) Waiting time ↓ (2) Response time ↓ (3) Resource utilizatn ↑
(4) Effeciency ↑

→ **Dirty read problem**

- Reading a value of an uncommitted transactn.

→ **Unrepeatable read problem**

- When reading a same value ~~tra~~ yields diff results

→ **Phantom read problem**

Deletion of value & then reading that value.

→ **Lost update problem** (write-write conflict)

* **Blind write**: A transaction writes a value without reading it.

→ **Scheduling** → Serial (safe)
→ Non-serial (unsafe)

Ways of finding consistent schedule

(1) Serial ($n_n!$)

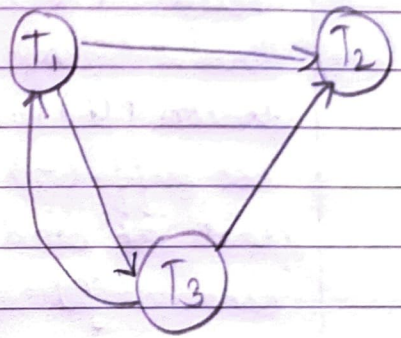
(1) Non-serial $\frac{(n_1 + n_2 - - + n_n)!}{n_1! \times n_2! - - n_n!}$

→ Conflicting instructrs

When they belong to same value and when one of them is write instruction.

→ Conflict serializibility

T_1	T_2	T_3
R(X)		
		R(Z)
		W(Z)
	R(Y)	
R(Y)		
	W(Y)	
		W(X)
	W(Z)	
W(X)		



∵ Cycle exist
∴ T_1, T_2, T_3 are not conflict serializable

→ Cost of finding whether schedule is conflict serializable ⇒ $O(n^2)$

→ RELATIONAL ALGEBRA

- It is one of the two theoretical formal query language associated with relational model
- Takes >1/2 relations and outputs a relation w/o name.

Basic / fundamental	Derived
Select (σ)	Natural join ($\bowtie$)
Project (π)	$\bowtie$, $\bowtie$, $\bowtie$
Union ($\cup$)	Intersection ($\cap$)
set diff ($-$)	Division ($\div$)
Cartesian product ($\times$)	
Rename (ρ)	

1) Select (σ): (Selecting rows / horizontal partitioning)

- Unary operator so can take only 1 table at a time
- Fundamental operator
- Syntax $\sigma_{\text{condition/products}}(\text{table})$
- Same func. as 'where' clause in SQL

$\sigma_{\text{condition1} \wedge \text{condition2}}(\text{table})$ } (AND

$\sigma_{\text{cond1}}(\sigma_{\text{cond2}}(\text{table}))$ }

$\sigma_{\text{condition1} \vee \text{condition2}}(\text{table})$ (OR)

2) Project (π): (Vertical partitioning)

- Unary, fundamental
- Equivalent to 'select' clause
- Syntax: $\pi_{\text{column-name}}(\text{table})$

$\pi_{c1,c2 \ldots cn}(\text{table})$

Select * from table

↓

(table)

that's it no operator needed.

- For queries using σ & π always 'σ' is BEFORE ~~sigma~~ 'π'.

3) Cartesian Product ($\times$):

- Binary operation, Fundamental
- $|R_1| = n$ $|R_2| = m$ => $|R_1 \times R_2| = nm$

$R_1(A, B)$ $R_2(C, D)$ => $(R_1 \times R_2)(A, B, C, D)$

- Allows combining two tables.
- Commutative operator, Associative

$R \times Q = Q \times R$ $A \times (B \times C) = (A \times B \times) C$

→ SQL (Structured Query Language)

- Domain specific language.
- Also used in modifying & creation of rdbms.
- SEQUEL → Structured English Query Language
- Data Definition Language (DDL):- provides commands for defining schema.
- Data Manipulation language (DML):- Works on ~~schema~~ schema instance.
- Transaction control
- Embedded SQL & Dynamic SQL
- Basic structure:

Select $A_1, A_2, A_3 \cdots A_n$ from $R_1, R_2, R_3 \cdots R_n$ where P
(column_name) (table_name) (condition)

① SELECT (π)

- Pick columns required in result out of all columns in i/p relation.

Select column from table ;

② Where (σ)

Select column from table where condition ;

For 2 condit^n's,

Select col from table where con1 & and con2 ;

For range queries

select col from table where (con1) b ≥ 100 and b ≤ 1000 (con2)
  ||
select col from table where (con1) b between 100 and 1000;

For between keyword or not between, the boundary value is considered and not respectively.

(3) Set operations (Duplication by default eliminated)

i) Union, intersect, except/minus/set diff.
(∪) (∩) (−)

select col from table1
  Union / minus / intersect
select col' from table2;

(4) Cartesian Product (⋈)

- Combines >=2 relations
- Commutative

select col from table1, table2 where condition and table1.common_col = table2.common_col;

(5) Natural join (⋈, ⋈, ⋈, ⋈)

- Same like (×) but considers only those values where both the relations have same value for the common attribute
- Commutative in nature

→ select col from table1 natural join table2 where condⁿ;

- May lead to data loss if attributes to be matched have diff names but same values within / Same values occur only in one table.

Page No.:
Date:

⑤ Inner Join / Join: (using)

- Works same as (x) if used alone
- If join used with 'using' it provides additional functionality.

  common attributes : (common_col1, common_col2)

  select col from table1 join table2 using (common_col1);

- (On) operator
- In join with 'on', allows us to use a general predicate over the relations being joined.

  select col from t1 join t2 on t1.com_col = t2.com_col
  where condtn ;

- 'on' is conditional operator & can be used instead on 'where'.

  select col from t1 join t2 on t1.com_col = t2.com_col
  and condtn ;

- In case of outer join, 'on' behaves differently than 'where'.

⋈ ⟕ ⟖ ⟗

Page No.:
Date:
YOUVA

(7) Outer Join / Lef / Righ / Complete order join :

- Overcomes data loss caused due to inner join / natural join.
- Fills in 'null' values for non-common values in the 2 relations.

(8) Rename Operator

- Changing names of columns, tables.

select acct_no, balance * 1.08 as compute from Account where balance < 1000