DBMS

→ DBMS is a collection of interrelated data and a set of programs for convinient and efficient access of storage data.

Difficulty with File Processing System:

1) Data redundancy and incosistency
   - same data copied no. of times / wastage of space
   - If one copy is updated, all the copies are to be updated simultaneously, otherwise it leads to incosistency

2) Difficulty in accessing data:
→ to meet the changed requirements, new programs may have to be written.

3) Isolation of data
→ data spreads over multiple files and different files with different formats. So, to get complete information, all are to be gathered

4) Concurrency Control
   Concurrent access
→ simultaneous access to same data by multiple processes.
→ If not controlled → improper updates.

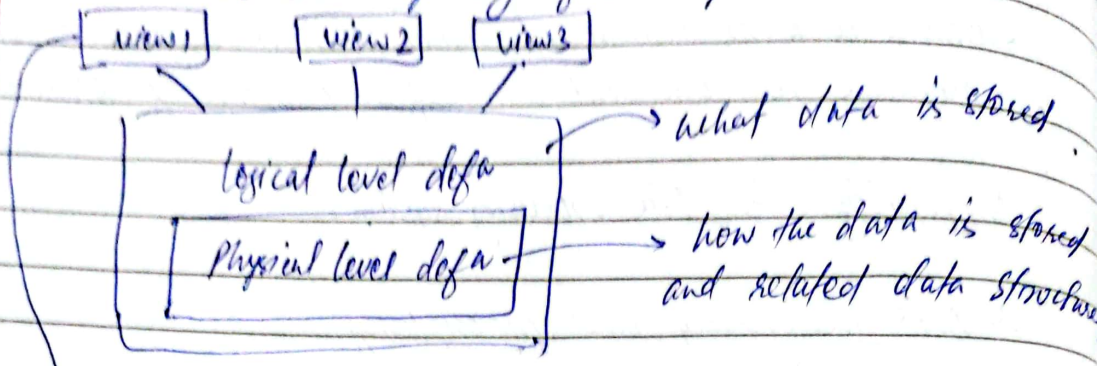
5) security -
   who can do what on which data.

6) Integrity enforcement

→ data must ~~certify cost~~ satisfy certain conditions / constraints

Data Abstraction :

→ To hide the complexity of the representation



Data Independence

→ Change in the physical data does not lead to change in application program.

→ physical level independence is achieved,

Difficult to achieve

→ Change in logical data does not lead to change in application program. → Logical level Independence.

Database Schema and Instance

Overall structure of database.

→ content of the database at a point of time (temporal aspect)

- Student (roll, name, dob, email, phone). → Changes frequent
- can change, but infrequently

Database Model

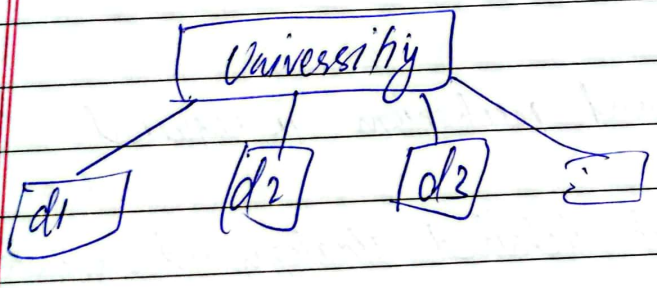
→ Underlying the database, there is a data model.
↳ representation of data and their interrelation, constraints on data, etc.

Record Based Model

- relational model
- network model.
- hierarchial model. → arranged according to hierarchy.

Relational Model → As a part of record, we store the value of a suitable attribute. to maintain relation/associati

Network Model → Instead of keeping the attribute value, pointer to related record is kept (unorganized).



Language

DDL (Data Definition language)
→ to specify/modify the schema of the database.

DML (Data manipulation language)
→ to access/work with the data of the datab

DML
(procedural)
what data to work with
and how to work.
∴ total process

→ non-procedural.
(what is required)

Functional Components of DBMS

1) database manager
core software module.
→ interaction with file manager (part of OS).
→ concurrency control.
→ security
→ integrity.
→ backup and recovery.

2) Query processor
→ non-procedural DML.
Actually, procedural mechanism is required.
→ makes an efficient detailed strategy and execute it

3) DDL compiler.
↓ translates DDL statements into meta data, which is stored into data dictionary. (part of database).

4) DML pre-compiler
Application programs.
→ DML statements for every interaction with database non-procedural.
→ DML to equivalent High level

→ proccedural features are also required.
Control struct/loop variables etc.

→ application is written in some HLL (host language).

DML pre-compiles, converts DML statements into equivalent host language.

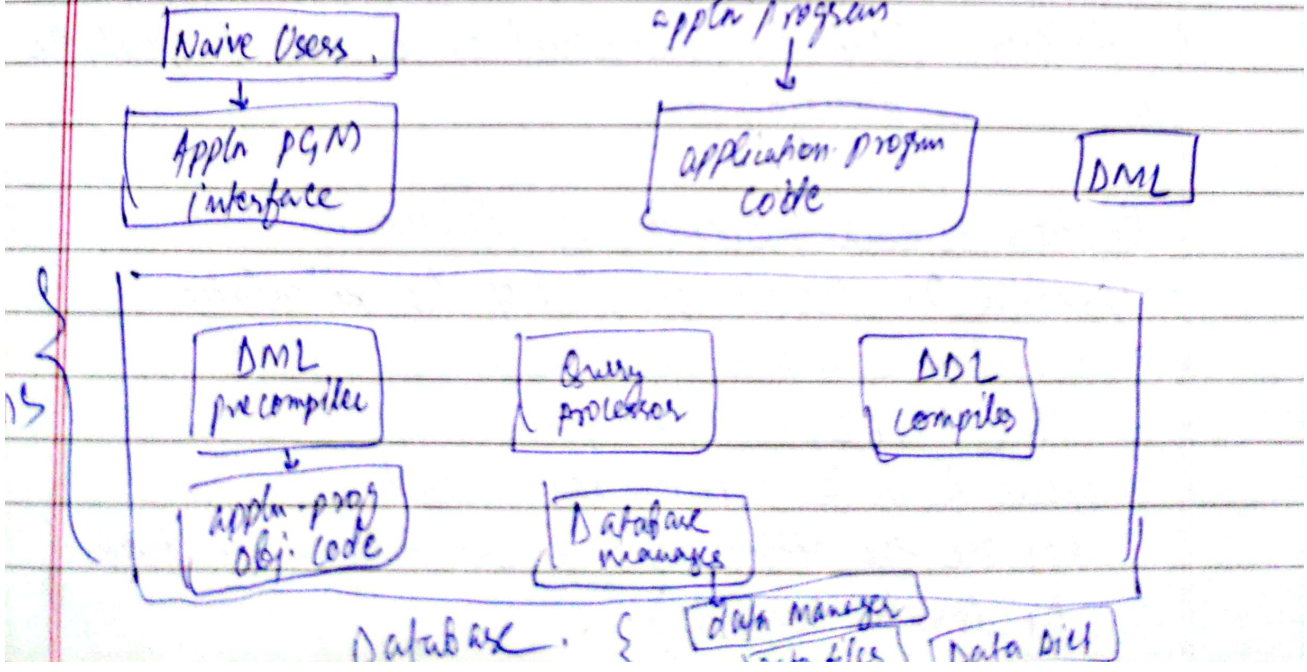
Inside that, use DML to interact with database. (embedded DML/SQL)

Users

- DBA (database Address)
  - → to specify/modify overall schema of database.
  - → to grant/revoke access for DBMS to users.

→ Application programmers.
- sophisticated users ⇒ DML

- Naive Users



Relational Model

→ Relational database is a collection of relations.

Informally, a relation is a set of values → table / file

Roll Name DOB Email

1
2
3

set of related values that represent an entity.

Each row of the table
→ tuple
→ describes an entity

Relation
Tuple
Attribute
→ domain

Attribute → column Header.

Table Name and column Headings
Relation Name — Attribute.
→ They help to interpret the data

Domain of an attribute.
→ a set of atomic values.
↳ cannot be divided further.
↳ application dependent.

→ Every attribute has a domain and it can take values from the domain. (Data type and format can be used to specify the domain).

Relation Schema / Intension of a Relation.
→ specifies the overall structure of the relation.
→ A relation schema is defined as.
R(A1, A2, ... An).
where R is the relation name, and
Ai's are the attributes. Ai can take values from the domain Dom(Ai).

→ Degree of relation is the no. of attributes in the relation.

→ Name of the attribute A_i describes the role played by $Dom(A_i)$ and the relation R.

Roll (int(3))	Score (int(3))
101	798
304	567

} Domain same but role depends on heading.

Relation / Relation State / Extension of a relation

$r(R)$ is the relational state of n-degree relation $R(A_1, A_2 \ldots A_n)$

Time varying

$r(R)$ is set of n-tuples $\{t_1, t_2, \ldots t_m\}$

where t_i is of the form $\langle v_1, v_2, v_3 \ldots v_n \rangle$

v_j is the value of A_j and comes from $Dom(A_j)$ or NULL.

Absence of any value.

Mathematically, a relation (state of relation) is n-degree relation on $Dom(A1), Dom(A2) \ldots Dom(An)$.

$$r(R) \subseteq Dom(A1) \times Dom(A2) \times \ldots Dom(An)$$

$$|r(R)| \leq |Dom(A1)| \times |Dom(A2)| \times \ldots |Dom(An)|$$

$t[A_2]$

value of attribute 2 in that tuple.

Characteristics of a relation:

→ No two tuples are same.

$r(R) = \{ t_1, t_2 \dots t_n \}$

→ tuples are unordered.

Relational model does not depend on ordering of the tuple.

→ Attributes in a tuple may or may not be ordered.

$R(A_1, A_2, A_3 \dots A_n)$

$r(R) = \{ t_1, t_2, t_3 \dots t_m \}$

ordered ← $t_i = \langle v_1, v_2 \dots v_n \rangle$

$v_i \in Dom(A_i)$.

unordered ← $t_i = \{ \langle A_i, v_i \rangle \}$, set of attribute-value pairs.

Ex - { <Roll, 10>, <dob, ...>, <Name, ...> }

$r(R)$ is a set of mappings $\{t_i\}$ where each mapping t_i maps from R to D.

$D = Dom(A1) \cup Dom(A2) \dots \cup Dom(An)$

→ If ordered → ordering of attributes in the schema is followed.

normally followed

* → Each attribute is single valued and atomic.

for a tuple, an attribute can have only value.

→ The value comes from the domain, or it may be null.

Atomic → It cannot be divided further (depends on appln requirement).

→ Interpretation.
→ a relation is a type ~~default~~ declaration about an entity / relationship

Constraints
→ Data in relation must satisfy the constraints.
→ model level constraints. (that comes along with the model).

→ Schema level constraints.
→ the constraints that can be specified at the time of schema defn.

→ Application Level Constraints / semantic Constraints.

→ Data dependency / functional dependency constraints.
Designing the database to judge the goodness of the design.

→ Domain Constraints
→ Each attribute has a domain when schema is defined, for the attributes, domain has to be specified.

```
CREATE TABLE STUDENT
  ( ROLL NUMBER (3,0)),
   ( NAME CHAR (25)),
        !
```

Primary Key Constraints

→ Subset of attributes, $\forall$ SK of R. ($\forall$ SK $\subseteq$ R) is called superkey of R, if for any two tuples, t_1 and t_2

$$t_1[SK] \neq t_2[SK]$$

→ A subset of superkey of no proper subset is superkey is called candidate key.
→ Minimal superkey is ~~subset~~ candidate key.
STUDENT (Regn No, Roll No, Name).

→ Candidate may consist of multiple attributes.

RESULT (ROLL, SUBCODE, SUBSCORE).
Minimal possible key is combination of (ROLL, SUBCODE).

→ One of the candidate key is chosen by the designer as primary key.

→ alphanumeric is preffered over alphabetic candidate key.
→ Smaller in size is preffered.
→ In the context of appln, which ~~is~~ one ~~use~~ is used for identification in reality.

→ Entity constraint.
→ primary key cannot be null.

→ NOT NULL (can be specified for an attribute, if null value is not allowed for that attribute).

Relation Database
↓ (set of rels and constraints).
schema of a relational database is $\{R_1, R_2 \cdots R_n\}$ where R_is are the schema of individual relations and set of integrity constraints.

State → $\{ r_1(R_1), r_2(R_2) \ldots r_n(R_n) \}$.

Integrity Constraints and Foreign Key

Referential Integrity :

Dept (Referenced)

(PK) DCODE	DNAME	...

Refers

Student (Referencing)

ROLL	NAME	--	DCODE (FK)
1			D1
2			D2
⋮			

R1 and R2 are two relations.
where R1 is the referenced relation
and R2 is the referencing relation.

→ PK be the primary key of referenced relation R1
→ A subset of attributes FK in R2 is foreign key in R2 if that ~~refers~~ refers to PK of R1.
→ Domain of PK in R1 is same as that of FK in R2. and for each tuple $t_2 \in r_2(R_2)$ either $t_2[FK]$ is null, or there exists a tuple $t_1 \in r_1(R_1)$ such that

$$t_1[PK] = t_2[FK]$$

Impact of FK on DML Operation :

Operation :	Referencing Rel^n	Referenced Rel^n
Tuple insertion	Value added in FK must must be present in referenced relⁿ. else not allowed.	No Impact.

Operation	Referencing Rln	Referenced Rln
Modifying the tuple.	If FK is ~~changed~~ value is changed ~~(not suggested)~~. then new value must be in referenced rel^n else not allowed.	If PK is getting changed. (not suggested) and old value is ref... By referencing rel... then not allowed
Deleting a tuple	allowed	If PK is value present in referen... rel^n. then not allowed.

Relational Algebra and Relational Calculus

→ To work with relational database.

→ Relational model → Relational algebra, Relational Calculus

Relational algebra → Procedural → Set of operations
Expression: sequence of operations on a rel^n.

Relational Calculus → Non-procedural
Expn: Data that we need.
(No operations).

Relational Algebra

→ provides the foundation to work with relational database.
→ useful for implementing and optimizing the query.
→ some of the concepts of relational algebra is implemented in structured query language (SQL).

DEPT(DCODE, DNAME, ...)

STUDENT(ROLL, NAME, DOB, ..., DCODE)
(FK).

find the students in dept 01(DCODE).

select operation.

$\sigma_{predicate}(R)$

↓ &&
> < = >= <=

$\sigma_{DCODE='01'}(STUDENT)$.

o/p reln:

Schema state → same as I/p reln.

→ set of tuples satisfying the predicate.

Relational Algebra

In an expn, one or more relation as input on those operations are applied. & O/p is another reln.

→ Select operation is commutative.

$\sigma_{SCORE>80}(\sigma_{DCODE='01'}(STUDENT))$

$\sigma_{DCODE='01'}(\sigma_{SCORE>80}(STUDENT))$,

$\sigma_{DCODE='01' \text{ AND } SCORE>80}(STUDENT)$,

$\sigma_{P_1}(\sigma_{P_2}(\sigma_{P_3}(R))) = \sigma_{P_1 \&\& P_2 \&\& P_3}(R)$,

PROJECT OPERATION

$\Pi_{ROLL, NAME}(STUDENT)$

O/p schema consists of only these two attributes.

No. of tuples in o/p => Likely to be same as no. of tuples in I/p rel^n.

$\Pi_{\text{list of attr.}}(R)$ $\quad \Pi_{\text{list2}}(\Pi_{\text{list1}}(R))$

<list1> $\subseteq R$. $\quad$ list2 $\subseteq$ list1.

→ PROJECT is not commutative.

→ If the projected attributes does not have superkey, then values may be repeated, so less no. of tuples.

Ex- $\Pi_{\text{SCORE}}(\text{STUDENT})$.

→ Duplicate eliminating project operation.

→ If project opn. attr. does not contain superkey, after projection, multiple tuples may be repeated, only one is retained and rest are eliminated.

$\Pi_{\text{ROLL,NAME}}(\sigma_{\text{SCORE}>80}(\text{STUDENT}))$.

RENAME OPERATION

R1(A, B, C, D).

$\rho_{R2(X,Y,Z,N)}(R1)$

$\rho_{R2}(R1)$.

$\rho_{X,Y,Z,W}(R1)$.

Cartesian Product

$R1 \times R2$

O/P Reln → Schema of R1 ∪ Schema of R2

R1 → degree n1 R2 → degree n2.

R1 × R2 → degree n1 + n2

R1 → no. of tuples t1, R2 → no. of tuples t2.

R1 × R2 → no. of tuples t1 × t2

STUDENT (ROLL, NAME, ..., DCODE)
DEPT (DCODE, DNAME ...).

STUDENT × DEPT Every ~~one~~ ~~stud~~ tuple of student gets multiplied with every tuple of DEPT.
& Apply filter to retain meaningful tuples.

$\sigma_{STUDENT.DCODE = DEPT.DCODE}(STUDENT \times DEPT)$

Fundamental Operations
- SELECT
- PROJECT
- RENAME
- CARTESIAN PROD
- UNION
- MINUS

Derived Operations
- JOIN.
- Natural Join.
- INTERSECTION.

JOIN Operation

STUDENT ⋈ DEPT
STUDENT.DCODE = DEPT.DCODE

⋈ → Join Condition.

→ If it is not based on the equality of the attributes, then we say θ join.

In equi join common attribute is twice.
To avoid we can use natural join.

STUDENT * DEPT → Natural join

Make pair of student with same score.

$\sigma_{STUDENT.SCORE = T.SCORE \text{ AND } STUDENT.ROLL > T.ROLL}$ (STUDENT × ρ_T(STUDENT))

→ For set operations, two relations must be union compatible.
R($A_1, A_2, \ldots A_n$) and S($B_1, B_2, \ldots B_n$) are union compatible, if
i) both are of same degree
ii) corresponding attributes are with same domain.
Dom(A_i) is same as Dom(B_i) for $1 \le i \le n$.

UNION
R ∪ S ⇒ will have tuples from both relations, and common tuples will appear only once.

→ schema → same as R or S.

$R \cup S \equiv S \cup R$ (commutative).

$R \cup (S \cup T) \equiv (R \cup S) \cup T$ (Associative).

MINUS

$R - S \Rightarrow$ tuples that are present in $r_1(R)$ but not in $r_2(S)$.

$R - S \neq S - R$ (not commutative)

INTERSECTION (can be derived from union and minus).

→ tuples common in both the relations.

$(R \cap S) = (S \cap R)$ (commutative)

$R \cap (S \cap T) \equiv S \cap (R \cap T)$

$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$

ADDITIONAL OPERATIONS:

→ Aggregate function and grouping.

→ Aggregate → Acts on a set of tuple and gives one tuples as output.

RESULT (ROLL, SCODE, SCORE).

P	1	S1	30
	2	S1	32
	⋮		
	1	S2	50

ROLL $\mathcal{F}_{SUM(SCORE)}$ (RESULT)

grouping attr. | Aggreg. fn.

o/p Schema
grouping attr. if any, aggregate fn with attr. on which it works

ROLL	SUM SCORE
1	500
2	704
3	906.

SCODE $\mathcal{F}_{MAXIMUM(SCORE), MINIMUM(SCORE)}$

O/P	SCODE	MAXIMUM SCORE	MINIMUM SCORE
	S1	97	100
	S2	98	:
	:	:	

RESULT (ROLL, DCODE, SCODE, SCORE)

1	D1
2	D2
3	D1
:	
1	D2
2	D2
3	D2
	:

for every dept, find no. of student.

DCODE $\mathcal{F}_{COUNT(ROLL)}$ RESULT

Aggregate of each student.

ROLL, SCODE $\mathcal{F}_{SUM(SCORE)}$

O/P : ROLL DCODE SUM-SCORE

Find rolls & of which sum of score ≥ 500.

$T(ROLL, TOTALSCORE) \leftarrow {}_{ROLL}\mathcal{F}_{SUM(SCORE)}\ RESULT$

$\sigma_{TOTALSCORE \geq 500}\ T$.

DEPT(DCODE, DNAME, ...) STUD(ROLL, NAME, ... DCODE).

$STUD \bowtie_{STUD.DCODE = DEPT.DCODE} DEPT$

→ Let's suppose, for a student, DCODE is yet to be assigned, so its null.

→ The student will not appear in the O/P.

→ We want all the students to appear.

→ We have to use OUTER JOIN.

R1 ⟕ R2

Left Outer join.
→ Retains all the tuples from R1

R1 ⟖ R2

Right outer Join
→ Retains all the tuples from R2

R1 ⟗ R2

Full Outer Join
→ Retains all tuples from both R1 and R2

OUTER UNION

R1 (X, Y) } partially union compatible.
R2 (X, Z)

R1 outer Union R2 | X ∪ Y ∪ Z

$t_1 \in r_1(R1)$
$t_2 \in r_2(R2)$
matched $\Rightarrow t_1[X] = t_2[X]$

If t_1 and t_2 matches $\Rightarrow$ a single tuple in the o/p

for t_1 with no match in $r_2(R2)$, then for it Z is n
for t_2 with no match in $r_1(R1)$, then for it Y is n

→ OUTER UNION, and full outer join
→ same if joining is done based on the equality of all common attributes.

DIVISION

$T = R \div S$
schema of $T \Rightarrow Y = Z - X$
$X \subset Z$.

R(Z) ⟶ ROLL, SCODE | Attendance
S(X) ⟶ SCODE | List of subject codes.

$T = R \div S$

↓

Roll of students who have ~~appeared~~ appeared in exam for all subjects.

$T_1 \leftarrow \Pi_x(R)$ → All Roll nos present.

$T_2 \leftarrow \Pi_y((T_1 \times S) - R)$ → Rolls present in at least one subject.

→ All SCORE ROLL combinations.

$T \leftarrow T_1 - T_2$

→ A tuple t in T (O/P) is present if there are set of tuples, t_r in R such that $t[Y] = t_r[Y]$ and t_r exists for each tuple in t_s in S with $t_r[X] = t_s[X]$.

TUPLE RELATIONAL CALCULUS

→ Non-procedural

what is required?

$\{t \mid cond(t)\}$

↙ tuple variable.

$\{t \mid student(t)\}$

t is a tuple variable that ranges over the tuples of student relation. Student is the range relation for the tuple variable t.

$\{t[roll], t[name] \mid student(t)\}$

Quantifiers

Existential Quantifier → $(\exists n)$
Universal Quantifier → $(\forall n)$.

$\{ t[roll], t[name] \mid Student(t) \text{ AND } t[DCODE] = 'D' \}$.

$\{ t[roll], t[name] \mid Student(t) \text{ AND } (\exists n)(DEPT(n)$
$\text{AND } n.DNAME = 'xyz'$
$\text{AND } t.DCODE = n.DCODE) \}$

RESULT(ROLL, SCODE, SCORE)
SUBJECT(SCODE, SNAME, TYPE) → Theory or sessional
T, S

Find the roll and name of ~~student~~ students with SCORE in all the papers ≥ 50.

$\{ st[roll], st[name] \mid Student(st) \text{ AND } (\forall s)(\text{NOT } SUBJECT(s)$
$\text{OR NOT}(s[type] = 'T') \text{ OR}$
$(\exists r)(RESULT(r) \text{ AND}$
$r.ROLL = st.roll$
$\text{AND } r.SCODE = st.scode$
$\text{AND } r.SCORE \geq 50)) \}$.

! true for all other types of universal rels.

$(\forall x)(P(x)) \Rightarrow (\exists x)(P(x))$ ← implies.

$NOT((\exists x)(P(x))) \Rightarrow NOT((\forall x)(P(x)))$

Safe expression
→ finite no. of tuples in the O/P.

ENTITY - RELATIONSHIP DIAGRAMS

→ At conceptual level, data requirement is represented by ER Diagram.
→ Shows the entities involved in the system and relationship among them. It provides the description of the entity types and constraints of relationship.

→ Entity is a real life object or concept.
→ Each entity is distinguishable from others.
→ An entity is described by a set of attribute-value pairs.

Entity Set - Collection of similar types of entities.
Entity Type - Describes the structure of an entity set.
STUDENT (Roll, Name, DOB - - -).

→ An attribute → domain.
→ An attribute is a mapping fn. from entity set to domain.

Attribute
simple and composite.
Atomic/Indivisible — Collections of simple attr.
Divisible.
(Domain specific).

Attribute

- Single Valued OR Multivalued.
  (For a type, attr. can have only one value).
  In entity,
  A relationship

- Stored Vs Derived.

Stored	Derived
Value can be known only by storing it	Value can be obtained from other stored attr.
Ex- DOB	Ex- Age.

Key constraints

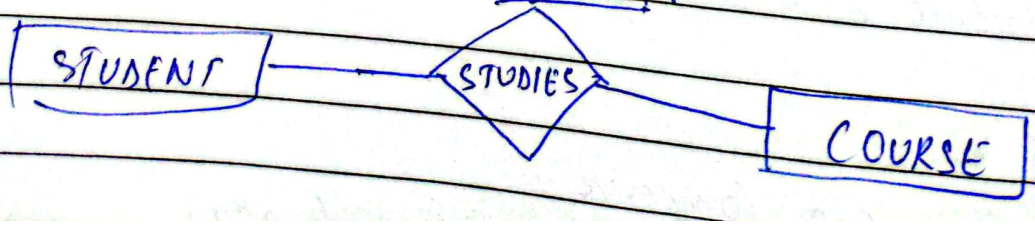
→ Unique value for each entity in the set.

STUDENT (NAME, ROLL, Ph-No, Age, DOB)
(Age → derived.)
(Ph-No can have multiple values.)

Relationship

→ Association between entities.

Relationship Set - Set of similar relationship.



Constraints on Binary Relation

- Mapping Constraint (Structural Constraint)
- Participation Constraint.

$A \longleftrightarrow B$

1:1 => An element of set A can have (one to one) association with at most one element of set B and vice-versa.

A to B

one to many → An element can have association with zero or more no. of elements of B. But an element of B can have association with atmost one element in A.

1 : N

A to B

many to one

N : 1

A to B

many to many

(N : M).

Attribute → Value set is V.

An attribute A of an entity type E is a function from E to P(V) (Power set of V).

$A : E \rightarrow P(V)$

P(V) → Set of all possible subsets of V.

A → composite

A1	A2	...	An
P(A1)	P(A2)		P(An)

Vi is the value set of Ai.

→ Value set (V) = P(V1) × P(V2) × ... P(Vn)

Participation Constraint

→ Also, known as minimum cardinality constraint.

→ Total, partial.

Total participation.

→ An entity type totally participates in a relation, me each entity from corresponding entity set must get associat with at least one entity of the other related type.

→ Total participation is also called existence dependency.

→ Entity type taking part totally in a relation is existentially dependent on the related type.

Relational Model → collection of relations.

↳ Map each entity type into a relation (relation model).

STUDENT (ROLL, REGD-NO, NAME, PH_NO, DOB)

In case there is any composite attribute, replace it by constituent simple attr.

DOB → day, month, year.

In case there is any multivalue attr., then put it remove it and place in a separate rel^n, copy key attr.

PH_LIST (ROLL, PH_NO)
FK & composite key

Roll will be a foreign key also

Mapping 1:1 Binary relation.

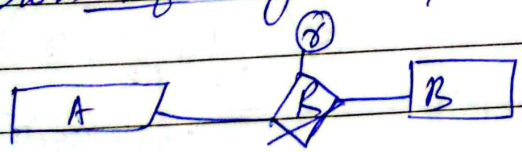
A (a_1, a_2 .. a_n)
B (b_1, b_2, ... b_n, a_1)

Foreign Key Based approach
→ Refer in the entity type that totally participates in the rel^n, copy the key of related entity type and it will be foreign key.

→ The entity type with key of other types will also have the attributes of relationship if any.

Merged relation approach
ARB (a_1, a_2 .. a_n, b_1, b_2 .... b_n, γ).
→ If both are totally participating, else lots of null values.

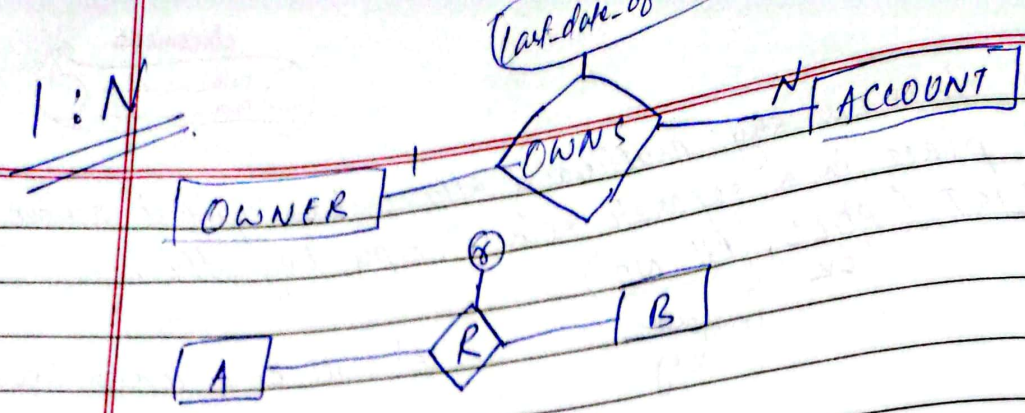
Cross Referencing RELN / RELATIONSHIP RELN / LOOKUP TABLE approach



A (a_1, a_2 .. a_n)
B (b_1, b_2 .. b_n)
R (a_1, b_1, γ)

Separate relation for the relationship attr. will be ⇒ key of both the entity types and its own attr (if any)
→ Each key will be foreign key here, both are individual keys can act as key of this ~~relation~~ rel^n.

1 : N

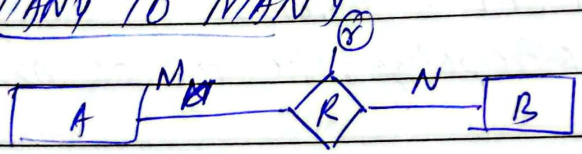


A ($a_1, a_2 \ldots a_n$)
B ($b_1, b_2, \ldots b_m$)

Foreign key Based
→ In many side, copy the key of other entity type as it will be foreign key.
→ Also copy attr. of rela (if any).

Lookup table
R (a_1, b_1, r)
key of many side → key.
copied key of entity type → FK.

MANY TO MANY

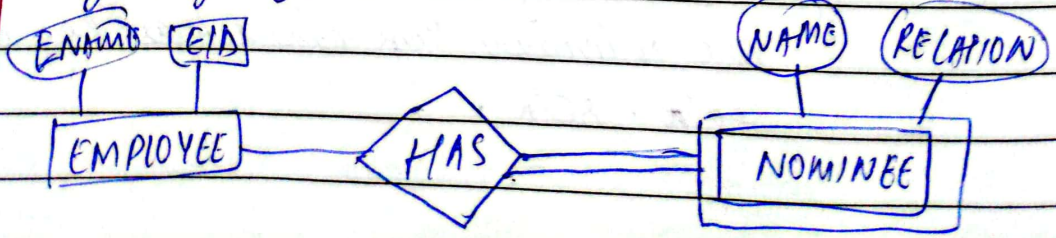


A ($a_1, a_2, \ldots a_m$)
B ($b_1, b_2, \ldots b_n$)

LOOKUP TABLE APPROACH

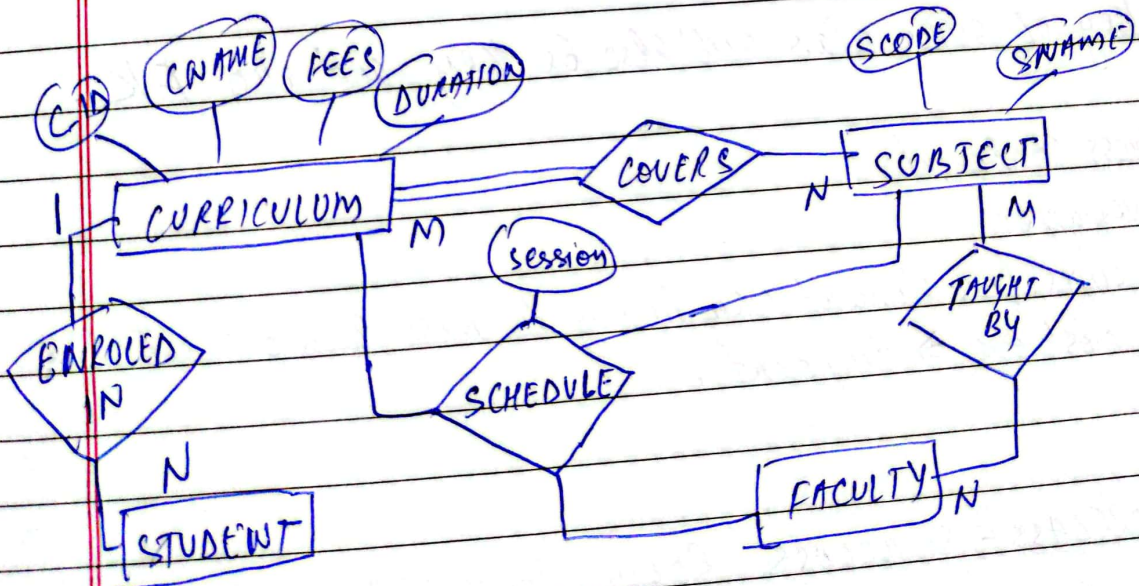
R (a_1, b_1, r)
FK FK
composite key

Weak Entity Type
→ weak entity type is an entity type which does not have any key of its own.



→ Weak entity type totally participates in the relationship with the entity type on which its existence depends.
→ This relationship is called identifying relation.
→ To identify an entity of weak entity type, one must have corresponding entity on which it depends.
→ By traversing the relation one can obtain subset of weak entity set which are related to corresponding determining entity.
→ In this subset, attribute(s) that can be used for individual weak entity is called the partial key.
(Unique value in the subset).

Mapping weak entity type:
A seperate rel^n.
Attr - Its own attr. U key of the entity on which it depends
NOMINEE (ECODE, NAME, RELATION).
(FK)
Key ⇒ Partial key U key of different type.



EER (ENHANCED ER DIAGRAM):

→ Generalisation - Specialisation.

CLASS / SUBCLASS RELN.
SUPERCLASS INHERITANCE

→ Certain attributes may be applicable only for a subset of an entity set.
→ A subset of an entity set take part in an association.

→ In a class - subclass reln. A subclass will have only one parent. Parent can have no. of subclasses.

→ PARTICIPATION CONSTRAINT & DISJOINTNESS CONSTRAINT
↳ Each quantity of superclass set must belong to at least one of the subclasses. Then total participation else partial.

Subclass Attr ⇒ All superclass attr. ∪ its own attr.

→ Key of super class will also be key and also FK.

Disjointness Constraint

Overlapped

→ A superclass element can belong to more than one subclass.
→ at most one ⇒ disjoint.

SUPERCLASS - SUBCLASS RELN. / INHERITANCE

→ In a superclass-subclass reln. there is one superclass.

A superclass element

↓

Belongs to which subclass ?

↓

Predicate or Attribute defined subclasses → Maybe based on the value of a superclass attribute.

Otherwise user defined.

With multiple relations

→ for a superclass there is a reln.

for each ~~reln~~. subclass, one reln.

It will have key of superclass as both key and FK of the subclass.

Additional attr SUB1($\underset{FK}{A1}$, B1, B2, B3).

SUB2($\underset{FK}{A1}$, C1, C2, C3).

→ No relation for superclass.

for each subclass, one reln.

Attr. of superclass ∪ its own attr.

Key of superclass will be the key.

B1(A1, A2, A3, B1, B2, B3)

B2(A1, A2, A3, C1, C2, C3)

With Single Relation

→ attr. of superclass ∪ attr. of subclass1 ∪ attr. of subcl

A (a1, a2, a3, b1, b2, b3, c1, c2, c3 --TYPE)
↳ which subclass

→ key of superclass will be the key.

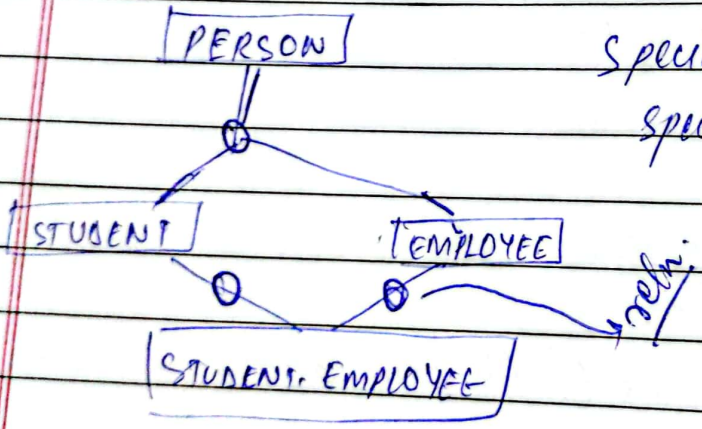
Disjoint → Single type.
Overlapped → Multiple

TYPE → Binary string of length
N ∈ Ti : 2^1, belongs to type

→ If additional attr. are quite large in no., ~~there~~ the large amount of null values will be there.
→ If less no. of additional attr. then it ~~can~~ may be used.

MULTIPLE INHERITANCE

→ A subclass is taking part in multiple superclass-subclass relation



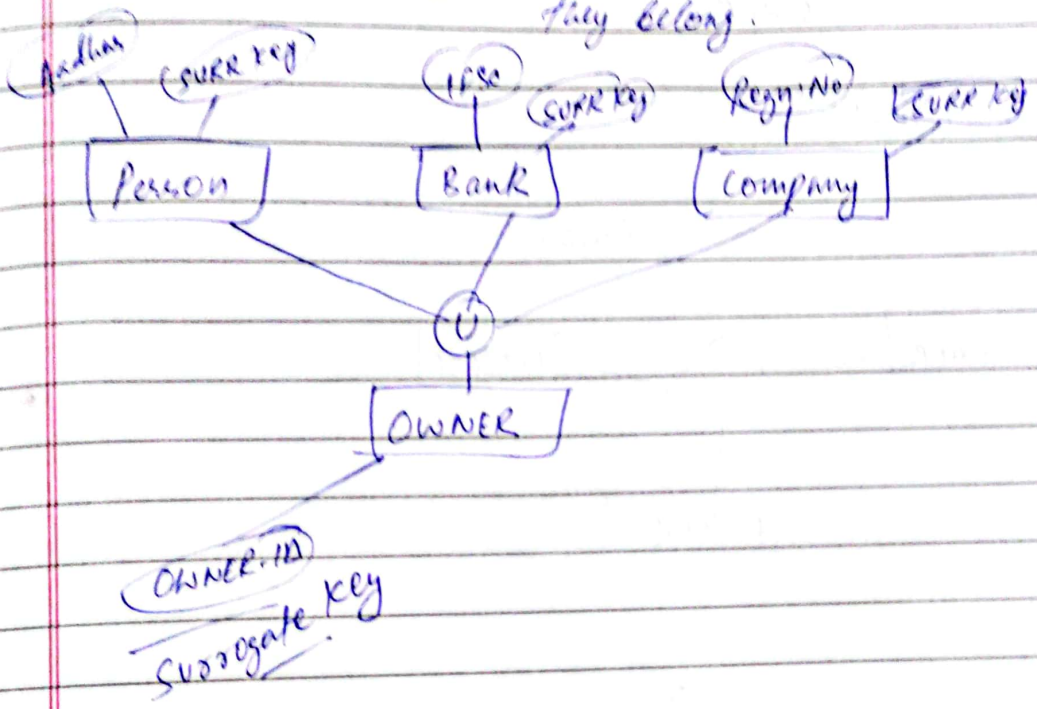
Specialization hierarchy
specialization lattice
↳ when multiple inheritance.

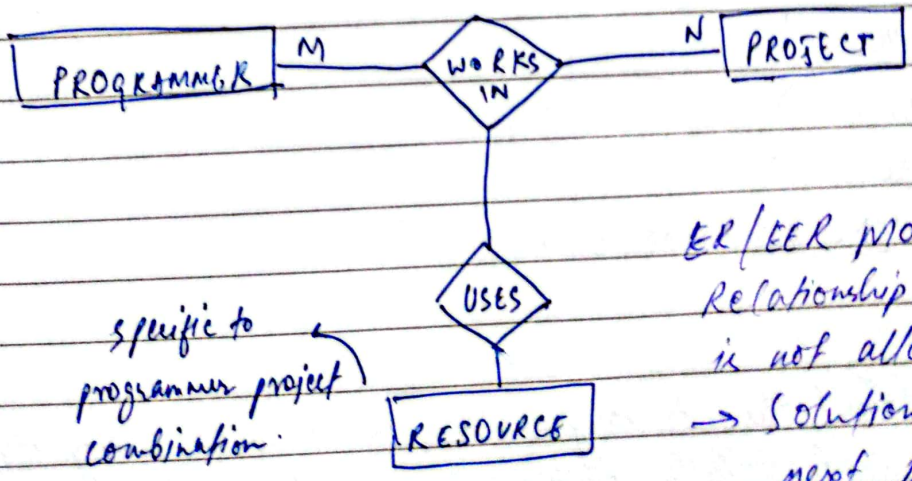
→ In a single association, a subclass will have multiple super classes and that represents different kind of entities
→ Subclass is union/category.

SUBCLASS
↳ Subset of intersection of superclass elements.

Category
↳ Subset of union of superclass elements.

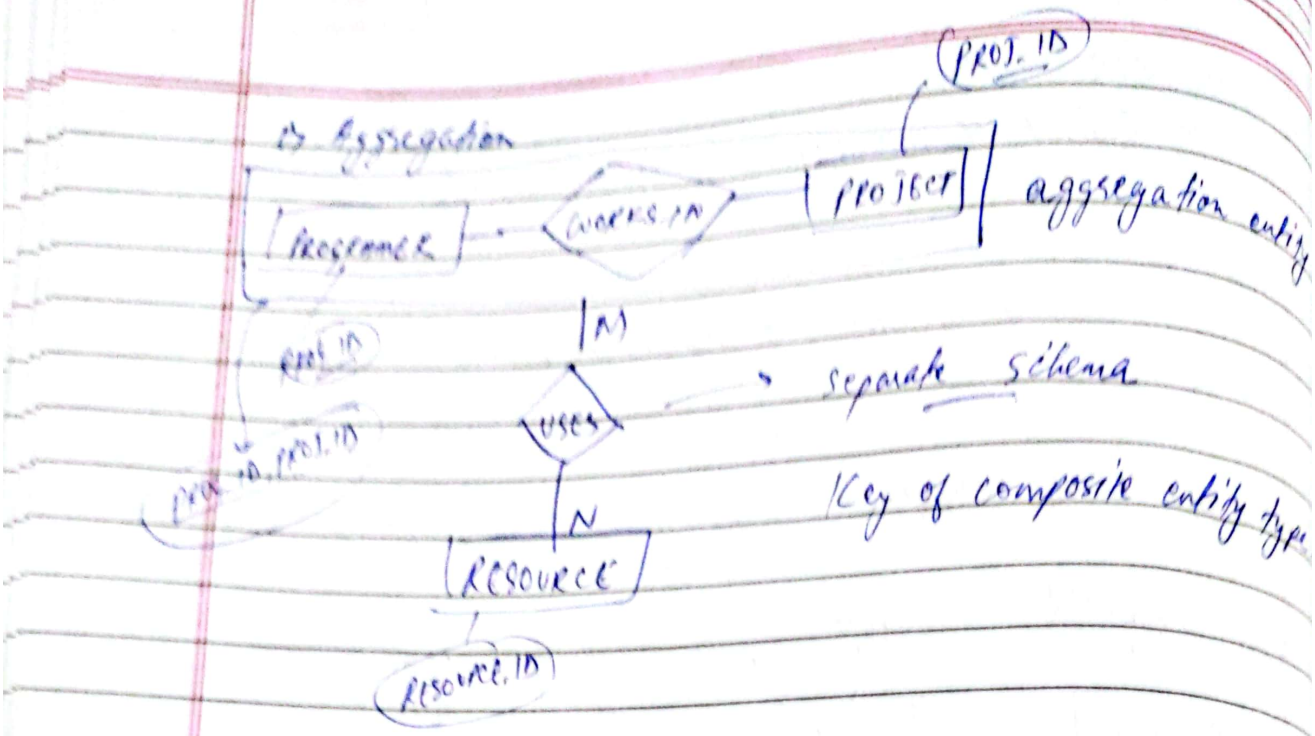
Attribute of category → depends on which superclass elements they belong.



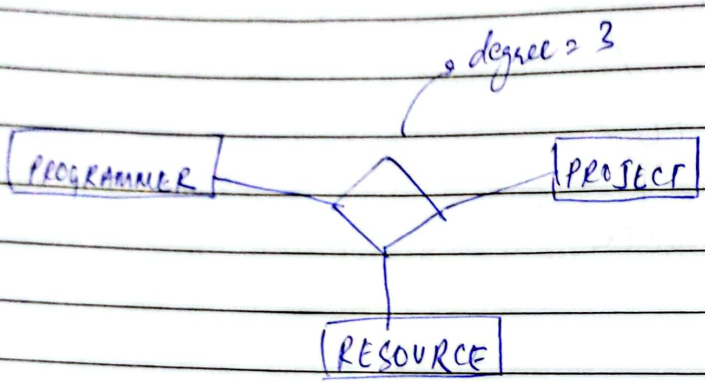


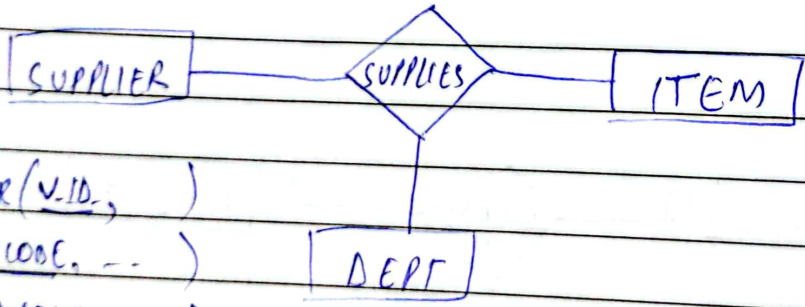
specific to programmer project combination.

ER/EER MODEL
Relationship with relation is not allowed.
→ Solutions in next page.



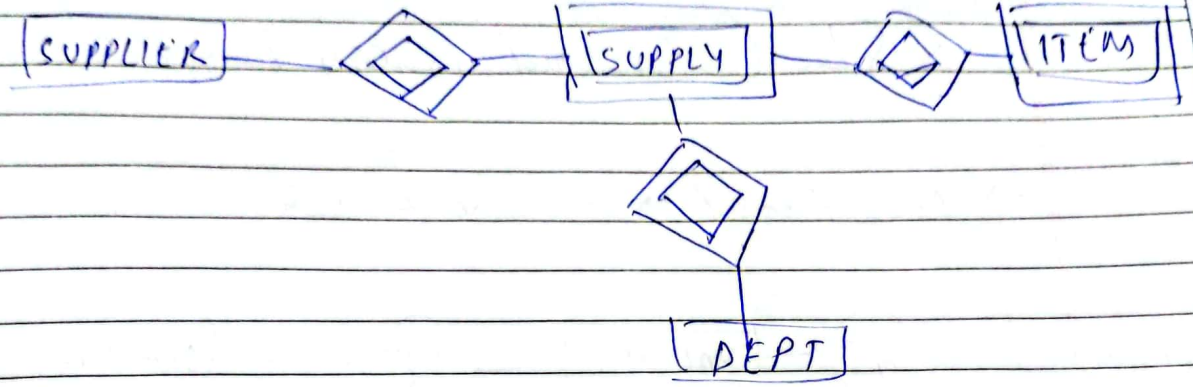
Key of composite entity type

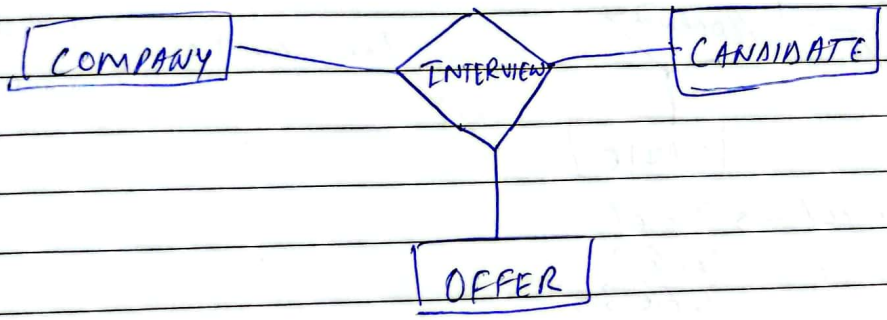


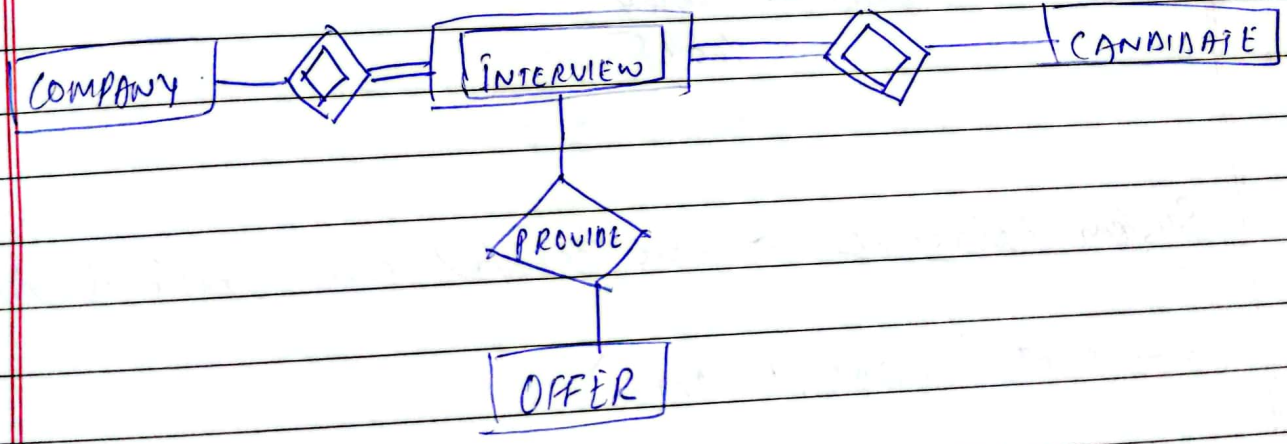


→ SUPPLIER(V.ID.,)
→ ITEM(ICODE, ...)
→ DEPT(DCODE, ...)
→ SUPPLY(VID, ICODE, DCODE, QTY, ...)

→ Convert the ternary relⁿ. in a weak entity type and it must totally participate with the related entity types (in original ternary relⁿ.) through identifying relⁿ.

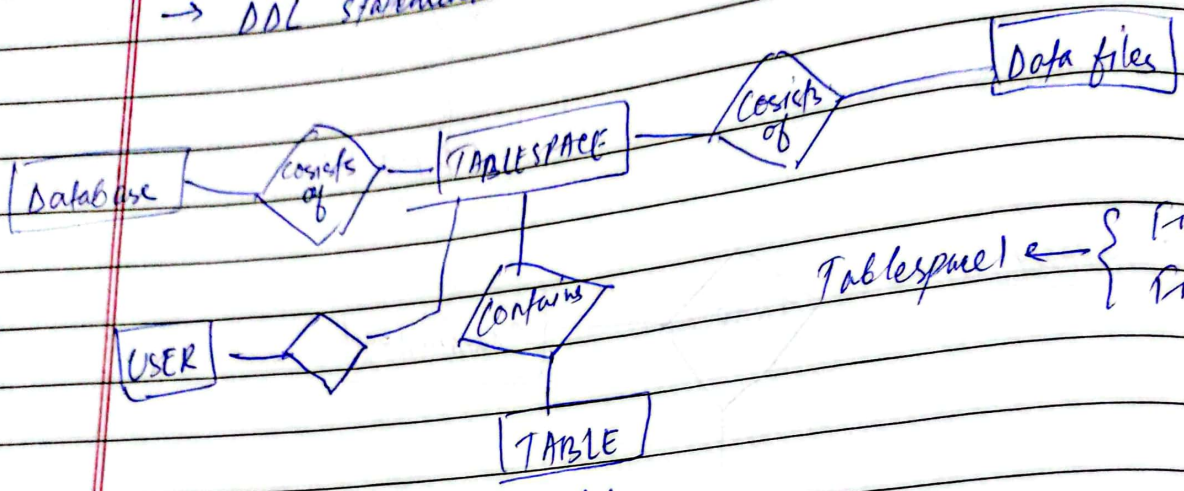

SUPPLIER
SUPPLY
ITEM
DEPT


COMPANY
INTERVIEW
CANDIDATE
OFFER


COMPANY
INTERVIEW
CANDIDATE
PROVIDE
OFFER

SQL (Structured Query Language)

→ DDL statement → to define a rel^n.



Tablespace1 ← { File1, File2

User1 → Tablespace1 → Table1
Table2
Table3.

User2 → Tablespace1 → Table4.
Table5

System tablespace → gets created when Oracle is install

Ex - DEPT (DCODE, DNAME).

```
SQL> CREATE TABLE DEPT
     (DCODE CHAR(5),
      DNAME CHAR(10));
```

→ press Enter.

→ termination of the comman and execute.

SQL BUFFER

stores the last SQL command.

press enter at blank line, terminate the command.

DCODE CHAR(5) PRIMARY KEY CONSTRAINT PK_DEPT

optional.

When violated, gives error with constraint name

CHAR(size).

NUMBER(7, 2)

Total no. of digits

No. of places after decimal.

Eg- STUDENT(ROLL, NAME, REGDNo, DOB, DCODE)

↳ from dept.

SQL> CREATE TABLE STUDENT
(ROLL NUMBER(3,0) PRIMARY KEY,
REGD-NO CHAR(10) UNIQUE,
NAME CHAR(15) NOT NULL,
DOB DATE
DCODE CHAR(5) REFERENCES DEPT(DCODE));

SUBJECT(SCODE, SNAME, ...)
STUDENT(ROLL, ...)
ATTENDANCE(ROLL, SCODE)

SQL> ~~CRE~~ CREATE TABLE ATTENDANCE,

.

At the end we have to specify for composite PK.

PRIMARY KEY (ROLL, SCODE).

RESULT (ROLL, SCODE, SCORE).

SQL> CREATE TABLE RESULT
(ROLL NUMBER(3,0),
SCODE CHAR(5),
SCORE NUMBER(3,0) CHECK(SCORE >= 0)
FOREIGN KEY (ROLL, SCODE) REFERENCES
ATTENDANCE (ROLL, SCODE)

GRADE CHAR(1) CHECK (GRADE IN ('A', 'B', ...

OR

GRADE CHAR(1) CHECK (GRADE BETWEEN 'A' ...

Attr. Type and size DEFAULT value.

DEPT (DCODE, DNAME).
STUDENT(ROLL, REGD.NO, NAME, DOB, DCODE).

SQL> INSERT INTO DEPT VALUES ('01', 'CSE');

>INSERT INTO DEPT(DCODE) VALUES ('02')

SQL PLUS COMMAND > DESC STUDENT.
↳ gives information (attr) for schema.

SQL> SELECT FROM STUDENT
WHERE DCODE = '01'

SQL> UPDATE STUDENT
SET DCODE = '01'
WHERE ROLL = 1 ;

SQL> DELETE
FROM STUDENT
WHERE

STUDENT (ROLL, NAME, DOB, DCODE)
DOB → DD-MON-YYYY
↓
DATE

SQL> INSERT INTO STUDENT VALUES (1, 'ABC', '02-FEB-2010', 'D1');

TO_DATE (date string, format) → 'DD/MM/YYYY'

DD → Numeric day of month MM → Num month.
YYYY → year. HH → hour MI → Minutes.
SS → Seconds

SQL> SELECT ROLL, TO_CHAR (BOB, 'MONTH, YYYY').

SQL> SELECT ROLL, NAME FROM STUDENT
WHERE DCODE = 'D1'

SQL> SELECT * FROM STUDENT WHERE DCODE IS NULL
* → All columns. DCODE IS NOT NULL

Students born after 2004.
SQL> SELECT FROM STUDENT WHERE DOB > '31-DEC-2004'
OR TO_DATE (datestring, format)

SUBJECT (SCODE, SNAME)
STUDENT (ROLL, - -)
RESULT (ROLL, SCODE, SCORE)

Name starts with RA.

SQL> SELECT * FROM STUDENT WHERE NAME LIKE 'RA%';

'RA%' → matches with zero/any no of any chars. → RAM, RAHIM

'RA_' → matches with any single char → RAM

Built in function

SINGLE ROW FUNCTION (Acts on individual row and provides an output)

AGGREGATE FUNCTION. (Acts on a set of rows and for a set of single outcome).

NUMERIC

FLOOR (number)
CEIL (number)
POW (m, n) → m^n
MOD (m, n) → m%n
TRUNCATE (number, upto which place)
TRUNCATE (175.7831, 2) → 175.78
, 0 → 175
, -1 → 170

ROUND (num, upto which place)

ROUND(175.7831, 2) → 175.78
→ 1 → 175.8
0 → 176
-1 → 180

DUAL → SINGLE DUMMY ROW
↳ Can be used for testing functions.

SELECT ROUND(num, dig) FROM DUAL;

STRING
LENGTH(STRING)
ASCII(char).
UPPER(string)
LOWER(")
INITCAP(STRING) First letter of each word capitalized.
RTRIM(STR) Trim empty spaces right.
LTRIM(STR) " " " left.
SUBSTR(string, posn, no. of chars)
m n
starting from mth position n chars will be returned.

Students born in feb.

SQL> SELECT ROLL, NAME
FROM STUDENT
WHERE SUBSTR(TO_CHAR(DOB, 'DD-MM-YYYY'), 4, 2)
= '02'

INSTR(string, substring, start pos, which occurance)
↳ By default → 1
Look for a substring in a string.

MONTHS_BETWEEN(date1, date2)
ADD_MONTHS(date1, n) → months added to date
LAST_DAY(date)
↳ Last day of the month in the date.

SQL> SELECT * FROM RESULT
ORDER BY ROLL;

ROLL
1
1
1
2
2
2

ORDER BY SCODE

ROLL	SCODE
1	S1
2	S1
3	S1
1	S2
2	S2

SQL> SELECT * FROM ~~STUDENT~~ RESULT
ORDER BY ROLL, SCODE

Roll	scode
1	S1
1	S2
1	S3
2	S1
2	S2
2	S3

By default it is ascending, for descending
→ ORDER BY X DESC

SUM, AVG } ignores null. count → ignores null.

MAX, MIN } → handle null

VARIANCE

STDDEV.

SELECT COUNT(ROLL) FROM STUDENT).

COUNT(DISTINCT & ATTR).

Ex- COUNT(DISTINCT DCODE) no. of depts with students in them.

SQL> SELECT AVG(SCORE), ROLL
FROM RESULT
WHERE UPPER(SCODE) = 'S1'

{cannot write this attribute with an aggregate function}

SQL> ORDER BY SUM(SCORE) DESC.

SQL> SELECT ROLL, SUM(SCORE)
FROM RESULT
WHERE SCODE != 'EN5'
GROUP BY ROLL
HAVING SUM(SCORE) > 400.
ORDER BY SUM(SCORE)

To get dept. name for every roll,

```
SELECT ROLL, DNAME
    FROM STUDENT, DEPT
    WHERE STUDENT.DCODE = DEPT.DCODE
```

Subject name and average score.

```
SQL> SELECT SNAME, AVG(SCORE)
        FROM SUBJECTS, RESULT R
        WHERE S.SCODE = R.SCODE
        GROUP BY S.SCODE, SNAME
```

SIMPLE SUBQUERY.

```
SELECT NAME
  FROM STUDENT
  WHERE DCODE =
      (SELECT DCODE
          FROM DEPT
          WHERE UPPER(DNAME) = 'CSE')
```

→ Inner query is executed first, outcome of inner query is used in evaluating the outer query.
→ Type of result returned by inner query must be compatible with the type expected by outer query.
→ Inner query must be return a single value else special measure has to be taken.

```
SQL> SELECT MAX(SCORE)
     FROM RESULT
     WHERE SCORE IS NOT NULL.

SQL> UPDATE RESULT
     SET SCORE = SCORE + 5
     WHERE SCORE IS NOT NULL
     AND SCODE = '51'
```

CORELATED SUBQUERY

→ Get dept with at least 10 students.

```
SQL > SELECT DNAME
      FROM DEPT D
      WHERE 10 <=
            (SELECT COUNT(*)
             FROM STUDENT S
             WHERE S.DCODE = D.DCODE).
```

→ Tuple of outer query is reffered in inner query.
→ Takes a tuple from the relⁿ. of outer query say, candidate tuple.
For the cand. key, inner query is executed. Result is used to decide the action on cand. tuple.
→ Inner query is executed no. of times.

```
SQL> DELETE FROM DEPT D
     WHERE 0 =
           (SELECT COUNT(*)
            FROM STUDENT S
            WHERE D.DCODE = S.DCODE)
```

Efficient approach:

```
SELECT SNAME FROM DEPT D
  WHERE EXISTS
  (SELECT * FROM STUDENT S
    WHERE D.DCODE = S.DCODE);
```

If it returns at least one / one tuple in the O/P, exists ⇒ true.

→ if backlog table is present

```
INSERT INTO BACKLOG
    SELECT ROLL, SCODE
        FROM RESULT
        WHERE SCORE
        IS NULL OR SCORE < 40
```

→ Create a backlog table.

```
SQL> CREATE BACKLOG
       AS
     SELECT ROLL, SCODE
       FROM RESULT
       WHERE SCORE < 40
```

```
CREATE BACKLOG (ROLL NO, SC
                )
```

```
ALTER TABLE tablename
ADD (column descp1,
     column descp2 )
MODIFY (column descr1,
        column descp2)
```

DATABASE DESIGN :

ER Model → Mapping to reln.

Formal way of designing.
↓ 2nd stage
NORMALIZATION.
→ Based on the concept of Key ~~constraint~~ and functional dependency.

→ In a relation, we have a set of attributes semantically related, interpretation associated.

Guideline for designing the database:

1) In a schema, we will have only one type of entity or relationship type. It should be easy to interpret.

Update anomalies and redundancy –

Ex- STUD(Roll, Name, dob, addr, courseID, CName, duration, fees).

→ Insert Anomaly.

New course → No student.

For student attr. → allow null values.

If Key is null → not allowed. ∴ redundancy.

No. of students in a course → course info repeated.

→ Deletion anomaly.

Deleting a student tuple and he/she is the only student for a course. So course info is also lost.

Modification anomaly.

If course info changed, it has to be changed for every student taking that ~~cou~~ course.

2) Design must be to minimize/remove redundancy and update anomalies.

NULL VALUE

- If we have a fat relⁿ. (large no. of attr., may be of different entity relationship typ.
  - → subset of attr. may be null for a tuple
  - → Attr. that are not applicable - null.

NULL → wastage of space
→ problem in joining.
→ aggregate problem.

3) Design to avoid at least frequent occurence of null value.

→ If joined based on arbitrary attr. → may give rise to irrelevant tuples
spurious tuples.

4) Joining does not give rise to spurious tuples.
Joining will be based on primary key / foreign key.

Functional Dependency :

→ X and Y are subset of attr. of the schema R.

$X \rightarrow Y$ is a funct. dependency that holds on R provided, for any two tuples t_1 and $t_2 \in R$ if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$

Eg- RESULT(ROLL, score, grade)

SCORE → GRADE
X Y

→ if we know, value of X, we know value of Y.
→ for students having same X, they will have same Y.

→ Functional dependency is specified on schema.
→ At an instance, each tuple in r(R) must satisfy the F.D (func. dependency).

→ Each t ∈ r(R) satisfies each F.D ∈ F then r(R) is a legal/valid state.

→ If X is a candidate key.
X → Y, Y is any subset of R.
i.e X → R.

→ If X → Y
it does not guarantee Y → X.
Eg → SOR SCORE → GRADE
GRADE → SCORE ✗

→ R(A, B, C, D ...)
A → B | A → C.
B → C
Given a set of func. dependencies F specified on relⁿ R we can infer other func. depn.

→ Closure of Func. Depnd. set. (F^+).
→ set of all possible f.d that can be inffered. F and holds on r(R).

INFERENCE RULES :

Armstrong's Axioms:
- IR1 → REFLEXIVITY RULE — if $X \to Y$ if $Y \subseteq X$
- IR2 → Augmentation Rule — if $X \to Y$ then $WX \to WY$
- IR3 → Transitive Rule — if $X \to Y$ and $Y \to Z$ then $X \to Z$

Extension:
- IR4 → Decomposition Rule if $X \to YZ$, then $X \to Y$, $X \to Z$
- IR5 → Union Rule. if $X \to Y$ and $X \to Z$, then $X \to YZ$
- IR6 → Pseudo transitive rule. if $X \to Y$ and $WY \to Z$, then $WX \to Z$.

→ Armstrong axioms are sound and complete.

Given a F.D set F on R, any F.D inferred from F using armstrong's axioms will be valid and holds on R.

complete – By successive application of armstrong's axiom of F till no new inference can be made, all possible F.D on R can be obtained. Closure of F.

Closure of a F.D set

Closure of an attribute (Given a FD set)

↓

set of all attributes which are implied by the attribute.

```
X+ = X
do {
    old X+ = X+
    then for each F.D f in F.D set F
        if LHS of f ⊆ X+
        {
            X+ = X+ ∪ RHS of f
        }
} while (old X+ != X+)
```

COVER OF A FD SET

→ A F.D set F_c covers the F.D set F if each F.D $f \in F$ is in F_c^+

$A \rightarrow C$

F_c	F
$A \rightarrow B$	$A \rightarrow B$
$B \rightarrow C$	$B \rightarrow C$
	$A \rightarrow C$

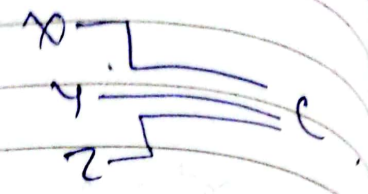
→ Two F.D set F1 and F2 are equivalent if F1 covers F2 and F2 covers F1

i.e. $F1^+ = F2^+$

MINIMAL COVER.
By removing the redundancy in F.D. → F_{MC}

CANONICAL FORM OF A F.D

$XYZ \rightarrow C_1C_2$



$X \rightarrow A_1, A_2, \ldots A_n$.

$\downarrow$

$X \Rightarrow A_1$

$X \rightarrow A_2, \ldots X \rightarrow A_n$.

1) Put F.D's in canonical form.
2) Find and remove redundancy in F.D's.
   a) If there is any redundant attribute on RHS of any FD then those attr. are to be removed.

$X \rightarrow Y$

$(X - \{A\}) \rightarrow Y$ holds

A is redundant.

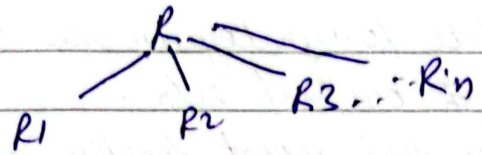
NORMALIZATION
→ The formal approach for designing the database based on the concept of key and functional dependency.

PRIME ATTRIBUTE
→ Attr. which is part of at least one candidate key.

Non-Prime Attr.
→ not part of any candidate key.

Why normalization?
→ to minimize update anamolies, occurence of null values and redundancies.

→ A complex structure will be broken into no. of simple schema/relations.



→ Decomposition must be loss-less
Loss-less-join → while joining the decomposed relations it must not give rise to spurious tuples.

→ Decomposition must be dependency preserving.

N → NORMAL FORM
↳ Highest normal form it satisfies.

Denormalization
→ Instead of highest normal form, designer may restrict the design to a lower form.

1ST NF → Not related with FD (Each attr. must be atomic and single valued)
↓
2ND → BCNF (Boyce Codd N.F)
↓
3RD

NORMALIZATION :

Eg. For every student unique roll no.

STUD(ROLL, NAME, PH_NO, CITY, SSCORE, SNAME, STYPE, FM, PM, SCORE, FAC_ID, FAC_NAME...

1st Normal form

→ Every attribute must be atomic and single valued.

* If composite break it into components.
* If multivalued, remove it from original reln and form a new reln. In the new reln copy primary key that will be foreign key in new reln referring to PK of original reln.
* Related attributes that repeat together, remove to form new reln.

STUD(ROLL, NAME, PH_NO, CITY, DOB)

SUBJECT(ROLL, SCODE, SNAME, STYPE, FM, PM, SCORE, FAC...

F.K

PK

F.Ds

ROLL → NAME, PH_NO, CITY, DOB

SCODE → SNAME, STYPE, FM, PM, FAC_ID

FAC_ID → FACNAME, F_PH

ROLL, SCORE → SCORE

2nd Normal Form

Partial dependency

$X \to Y$ if $A \subset X$ and $X - \{A\} \to Y$ holds.

Then Y is partially dependent on X.

Otherwise, Y is fully functionally dependent on X.

→ A relⁿ. is in 2NF if every non-prime attribute is ~~fully~~ fully functionally dependent on key.

→ If key is simple, then it is in 2NF, after 1NF.

$X \rightarrow$ key.

$X - \{A\} \rightarrow Y$

Violates 2NF

↓ corrective measure

Remove the partially dependent attr. in a new relⁿ, and copy the part of key on which they depend.

will be PK in new relⁿ, FK in original relⁿ.

STUD(ROLL, NAME, PH-NO, CITY, DOB)

SUBJECT(SCODE, SNAME, . . . , FID, FNAME, FPH).

RESULT(ROLL, SCODE, SCORE).

1NF
↓
2NF
↙ ↘
3NF BCNF

→ A relⁿ is in 3NF, if for every FD $X \rightarrow Y$ that holds on the relⁿ, Either of the following is true
→ i) X is a super key.
or Y is prime

~~is~~ X is cand. key OR Y is ~~a~~ prime

X is not cand. and Y is not prime.

X is part of cand (PRIME)

OR X is not part of cand (NON prime)

X is prime → Y is not prime } violation of 2NF

→ To be in 3NF, Rel^n. must be in 2NF and no trans dependency.

X → Y violates 3NF.
Non-prime Non-prime

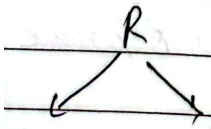
→ From the rel^n, remove Y in a new rel^n.
copy X.
X is PK in new rel^n.
In original form X is FK.

R1 (SCODE, SNAME, STYPE, PM, PM, F_ID)
FK

(F_ID, FAC_NAME, F_PH).
PK.

BCNF → stricter version of 3NF

for every F.D X → Y that holds on rel^n,
X is a super key.



loss-less if
Either R1 ∩ R2 → R1
or R1 ∩ R2 → R2

Dependency Preserving

→ F.D set F is defined on universal ~~set F~~ rela R (all attr)..
R has been decomposed into $R_1, R_2 \dots R_n$.

Let, F_i be the projection of F on R_i
set of F.D in F whose LHS and RHS attributes are in R_i.

Restriction of F on R_i

3NF and BCNF

→ Both remove redundancy.
→ Both are loss less
→ BCNF may not be dependency preserving.

Multivalued Dependency (MVD and 4NF)

$X \twoheadrightarrow Y$ is defined on R
X multidetermines Y.
$X, Y \subset R$
$Z = R - (X \cup Y)$

$X \twoheadrightarrow Y$ if the following situation exists.
→ t_1, t_2, t_3 and t_4 are 4 tuples in $r(R)$ such that
$t_1[X] = t_2[X] = t_3[X] = t_4[X]$ and
$t_1[Y] = t_2[Y]$, $t_3[Y] = t_4[Y]$ and
$t_1[Z] = t_4[Z]$ and $t_2[Z] = t_3[Z]$

P.T.O

$X \Rightarrow x_1$

$Y \Rightarrow y_1, y_2$

for a given X, there are no. of distinct values of Y.

Tuples are to be formed for every distinct value of Z corresponding to each X, Y combination.

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_1
x_1	y_1	z_2
x_1	y_2	z_2

$X \twoheadrightarrow Y$

given X ⇒ set of values of Y which depend only on X.

if $X \twoheadrightarrow Y$
then $X \twoheadrightarrow Z$.

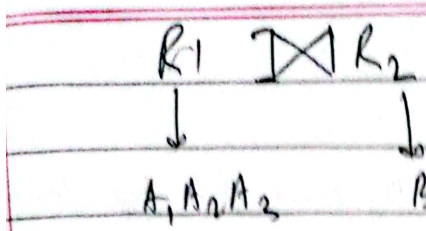
ROLL RHONE SCORE Trivial → $X \cup Y = R$

ROLL $\twoheadrightarrow$ PHONE

ROLL $\twoheadrightarrow$ SCORE

A non-trivial, $X \twoheadrightarrow Y$ if holds on a reln, then it violates 4NF.

→ Remove Y from R, put it into new reln, Copy X. PK = XY




```
SELECT  R1.A1, A2, A3, B1, B2
FROM    R1, R2
WHERE   R1.A1 = R2.A1 (+) .
```

PL/SQL (specific to Oracle).

DML - Non procedural.
Appln → procedural aspects.
→ provides the procedural aspects.

PL/SQL Block

optional | DECLARE
{ vars and constants
}

executable part | BEGIN

END;

→ PL/SQL is sent to a server in a single shot.
Network traffic is reduced.
~~In normal~~ so" If sent seperately, network traffic Yes.

STUDENT (ROLL, NAME, ...)

Dynamic data type declaration
STUD.NAME%TYPE

```
DECLARE
    SNAME CHAR(30);
BEGIN
    SELECT NAME
        FROM STUDENT INTO SNAME
        WHERE ROLL = 1;
    &
    ECHO SNAME;
END;
```

N NUMBER(2) := 5 ; := Assignment

NC CONSTANT NUMBER(5,2) : 3.14

%ROWTYPE → dynamic declaration for holding a row.

```
DECLARE
  REC   EMP%ROWTYPE
  EN    EMP.NAME%TYPE
  EC    EMP.ECODE%TYPE
BEGIN
   SELECT * INTO REC

   EC := '&EC'
```

→ Take input from user

Values can be accessed as
REC.ECODE
REC.NAME

```
old 6 : EC := '&EC'
New 4 : EC := '&E001'
END.
```

```
DECLARE
   DC  EMP.DCODE%TYPE

BEGIN
   DC := '&DC'
   UPDATE EMP
      SET BASIC = BASIC + 1000
      WHERE DCODE = DC
```

Update 0 or any no. of rows.

```
   DELETE FROM EMP
      WHERE . . . .
```

delete 0 or any no. of rows.

```
   C NUMER(5,0)

   SELECT COUNT(*) INTO C
      FROM EMP
```

Always returns something, not exception

```
IF cond. Exp then.
   :
END IF
```

```
IF condn. Exp THEN.
   :
ELSE
   :
END IF
```

```
LOOP
   :
IF cond. Exp then .THEN.
   EXIT
END IF
   :
END LOOP
```

```
WHILE cond exp
LOOP


END LOOP
```

```
SUM := 0          → don't declare or declare as compatible type.
FOR (I) IN 1..10
LOOP
  SUM := SUM + I
END LOOP
```

} I should not be modified inside the loop.

```
FOR (@) I IN REVERSE 1..10
LOOP

END LOOP
```

CURSOR → holding space in memory for a set of rows
↳ To hold number of rows.
Hold employees with BASIC > 50,000

```
DECLARE
   CURSOR C1 IS
      SELECT * FROM EMP
         WHERE BASIC > B;
   R C1%ROWTYPE;
BEGIN
B := &B;
OPEN C1;

FETCH C1 INTO R
```

(cursor name)

B EMP.BASIC %TYPE.

OPEN Cursorname
Query in its defn is executed
Set of returned rows are stored in a cursor.

CURSOR ATTRIBUTES

cursor name %ISOPEN → True if cursor is OPEN
else FALSE.

CURSOR
- EXPLICIT
- ~~IMPLICIT~~ IMPLICIT → automatically created when SELECT, UPDATE, DELETE is executed. Name – SQL.

%FOUND → last fetch was successful.
%NOTFOUND → last fetch was not successful.

%ROWCOUNT → cummulative value of no. of rows fetched so far.

```
FETCH C1 INTO R
WHILE C1%FOUND.
 LOOP
      FETCH C1 INTO R
END LOOP;
CLOSE C1;
```

} access each row one by one.

```
FOR I IN C1
LOOP
   I...
END LOOP
```

} Opens C1, fetches row one by one closes C1

does everything

```
FOR I IN C1
LOOP
    I.NAME.
    I.ECODE..

END LOOP
```

```
FOR I IN C1
LOOP
  IF I.... THEN.
     UPDATE EMP
      SET BASIC = BASIC +1000
      WHERE CURRENT OF C1.
  END IF.

END LOOP.
```

```
SELECT C1
   CURSOR C1 IS
     SELECT *
       FROM EMP
       WHERE ...
       FOR UPDATE OF BASIC
```

EXCEPTIONS

Built in exception.

- CURSOR_ALREADY_OPEN : trying to reopen a cursor which is already open.

INVALID_CURSOR : by trying to close a cursor which is not open.

INVALID_NUMBER } If char to numeric conversion
VALUE_ERROR } fails in SQL statement.

→ Except the case of for invalid_number,
all type of type conversion / constraint violation or truncation of data
⇒ VALUE_ERROR

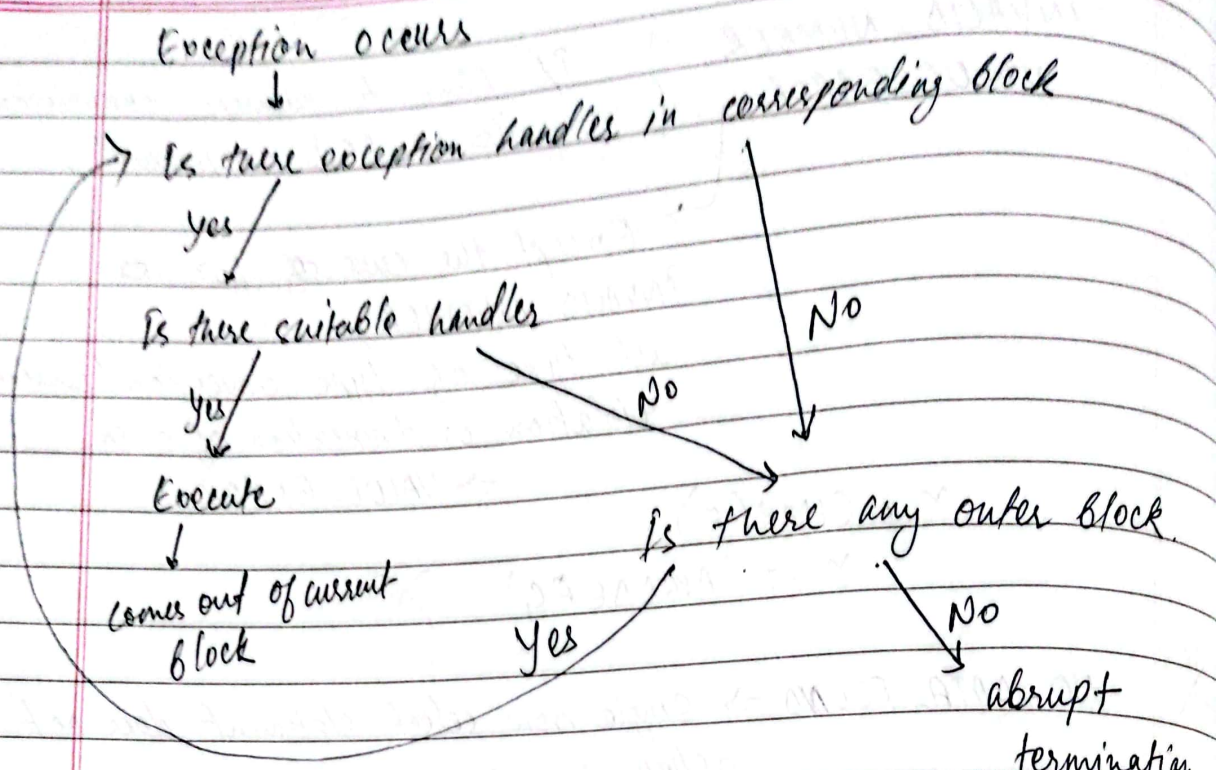
X := CHAR(5)
X := 'ABCDEFG' ✗.

NO_DATA_FOUND ⇒ Single row select statement does not return any row,
{fetched row is null} with cursor if fetch does not bring any data and trying to work with the row.

TOO_MANY_ROWS ⇒ If single row select statement returns multiple rows.

BEGIN
...

DECLARE
...
BEGIN
...
EXCEPTION
WHEN NO_DATA_FOUND THEN
...
WHEN OTHERS THEN.
...
END




```
DECLARE
BEGIN
   ...
   DECLARE
   ...
   BEGIN
   ...
   EXCEPTION
   END;
EXCEPTION
END.
```

```
DECLARE
my_exception EXCEPTION.
BEGIN
   IF ... THEN
   RAISE my_exception
   END IF
EXCEPTION
WHEN my_exception. THE
   ...
END.
```

Continued after 2 pages.

DBMS

TRIGGER

→ Trigger is a subprogram which is stored as a part of database, it is associated with an event and automatically invoked when the event occurs.
(cannot be invoked explicitly).

→ is DML statement on a table.

```
                    Optional
CREATE OR REPLACE TRIGGER triggername
    ← BEFORE / AFTER
    INSERT / DELETE, UPDATE OF COL1, COL2, ...
    ON tablename.
    FOR EACH ROW } Optional

        PL/SQL Block.
```

(BEFORE/AFTER: when to fire this trigger; INSERT/DELETE/UPDATE: event; FOR EACH ROW: statement level or row level)

terminate an event.

RAISE_APPLICATION_ERROR (errorno, msg)

errorno: -20000 to -20000.

PL/SQL Block

```
BEGIN
  IF :NEW.BALANCE < 0 THEN.
      RAISE_APPLICATION_ERROR(- - ).
  END IF
```

:OLD, :NEW } refers to rows.

FILES & INDEXES

Seek Time

→ Read/write ~~the~~ head to be positioned at the desired track. Time acquired to achieve this seek time.

seek time > rotation time.

RPM Revolution/Min.

RPM ↑es Rotational delay ↓es

Block transfer.

t_r bytes/sec

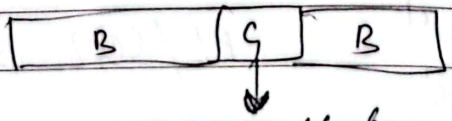
Block size B

$t_r \longrightarrow 1s$

$1 \longrightarrow \frac{1}{t_r}$

$B \longrightarrow \frac{B}{t_r}$

$\rightarrow \frac{B+G}{t_r}$.



Inter block seperation.

To get actual B bytes of data, B+G bytes of data to be read.

File

Collection of records → Collection of related fields.

Records

Fixed length

Variable length.

→ storing records of different ty

→ Records are of same type

Some fields are with multiple values e.g.

→ certain fields have var

→ some fields are optional

fixed length → jumping to a record is easier.
→ modifying a record is easier.
→ wastage of space

How to overcome?

① variable length field → Allot max length to the field.
OR
Store length and then value.

Optional fields → make fields mandatory.

multiple value fields → max no. of fields.

Record seperator

→ field seperator (to deal with variable length fields).
→ <FieldName, Value> (to deal with optional values).
↳ field Name seperator.

→ To deal with multi-valued fields, we need value seperators.

* → seperators must not appear as part of data.

→ file header contains information related to seperators.

Spanned Organisation / Unspanned Organisation
if a record is allowed to cross the block boundary.
↳ A record cannot cross block boundary.

Block Size : B byte
Record size : R byte.
if $R > B$ ⇒ spanned organisation.

Unspanned org → residual part is wasted can be used for linking the next block.

Fixed length record
↓
Unspanned org.

Blocking factor (bfr)
→ No. of records/block.
bfr = $\lfloor$ Block size / Record size $\rfloor$

Variable length record
No. of records/block varies.

bfr is defined as average no. of records/block.

A file → b no. of blocks.
r → total no. of records.

$b = \lceil r/bfr \rceil$

Allocation of Blocks

Contiguous allocation → advantage → sequential access of all data is faster.
disadvantage → expansion of file is problem.

Linked allocation → A block keeps the link for the next one.
→ growth of file is easier
→ accessing all blocks, sequentially, is time consuming

Cluster allocation → collection of contiguous blocks. Clusters are linked.

Indexed allocation → A table holding the 8 block addresses.

FILE

Unordered / Heap
storage of records does not follow any ordering.
→ stored in order of their insertion

Ordered.
Records are in order based on certain fields.

Unordered file.
Adv → insertion (addition of rec) is easier.

→ Retrieval → sequential
worst case → → no. of disk access.

When deleting, logical deletion is preffered, deleting every time or relocating records is difficult.

Ordering field { key field → ordering key
Non-key → ordering non-key.

Search based on ordering field (Binary Search) → faster.
else sequential.

Internal HASHING

in primary memory.
↓
table/block

Hash key
| on which hash fn is applied.
↓ index into table.

index computes
should cover the
range of table,
less collision.

→ Chaining can be used to avoid collisions.

External Hashing

hash (hash key field)
↓
Bucket No.
(collection of no. of 'B' Blocks)

→ collision problem is less as the Bucket can hold no. of records
No. of hash collision <= Bucket capacity.

Indexing.

To make the search with the file faster.

Overhead

Data file
Index file.

INDEX

- Dense Index
  - For every data record there is an entry/rec in index file.
- Sparse Index.
  - No. of index record. < no. of data record.

Index file
→ Ordered file (ordered on index field)
→ Mostly, fixed length record.
Rec has two fields
< Index field value, Record/Block pointer >.

Data File

- Ordered
  - case 1, Indexing Field ≡ Ordering Field.
    - Key → Primary Index
    - Non-key → Clustering Index → sparse Index.
- Unordered.
  - Data file is not ordered on index field.
  - → Secondary Index
    - Key
    - Non-key
  - → can be sparse/dense.

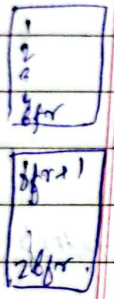
PRIMARY INDEX

→ Data file is ordered on key.
→ Ordering key field is the indexing field.
→ Primary index file is an ordered file of fixed length.
It is ordered on indexing field (same as ordering key)
Records are of the form < index field, record/block pointer >

→ First record of data block → Block anchor / anchoring record

In the index file, no. of records = No. of data blocks.

corresponding to block anchor
there is an entry in index file



Primary index is sparse.
Size of pri. index << size of data file.
↦ No of index record << No of data record.
→ size of index record smaller than that of data record.

Ex. – 50,000 data records.
Size of data record = 50 bytes
Block = 1024 bytes

$$bfr = \left\lfloor \frac{1024}{50} \right\rfloor = \left\lfloor \frac{1024}{50} \right\rfloor \approx 20$$

$$\left\lceil \frac{50000}{20} \right\rceil = 2500 \text{ data blocks.}$$

Binary Search on data file

$$\lceil \log_2 2500 \rceil = 12 \text{ block access.}$$

Pri Index Primary Index

No. of entries = No. of datablocks.
= 2500

Size of index record = 10 bytes.

$bfr = \lfloor \frac{1024}{10} \rfloor \approx 100.$

No. of index block = $\lceil \frac{2500}{100} \rceil$

$= 25.$

$\lceil \log_2 25 \rceil + 1 = 6$ Block accesses.

→ If insertion or deletion of data records leads to change in block anchor, then primary index file which is also an ordered file gets affected.

Cluster Index

→ data file is ordered on a non-key field.
→ indexing is to be done on that ordering field (Non-key).
Eg- order by dept.
→ An ordered file of fixed length records.
→ Ordering non-key ~~fifo~~ field is the index field.
Record format < Index field value, Block pointer >
→ sparse.
→ For each distinct value of ordering (indexing) to non-key field, there is an entry
(distinct value, block in which it first appears).

Secondary Index

Index field:

Data file is not ordered on index field → Secondary Index.

Index field
- Key
- Non-key.

⇒ Data file is not ordered on indexing field and it is a key.
- → Dense
  - → for each data record, there is an entry in index file.
  - fixed length < Index field, Block/rec pointer >.
  - Index file is ordered on index field.

⇒ Data file in not ordered on index field and its a non-key.
- → Secondary index is a non-key field.
- Any index → ordered on index field.

Sparse.

< Index field, Set of block/rec pointer >
- for each distinct value of index value
- variable length record.

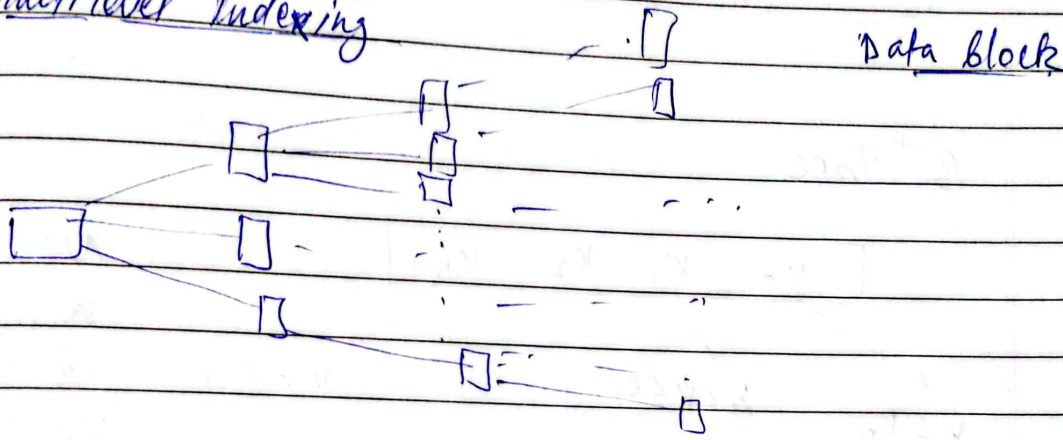
dense.

< Index value, record pointer >

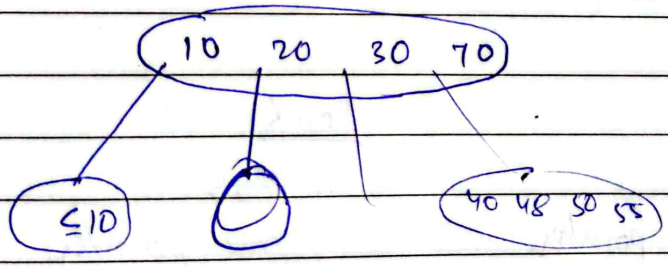
Sparse and fixed length entries
→ additional level of indirection is introduced

Index field value — Pointer to a block of record pointer.

Multilevel Indexing

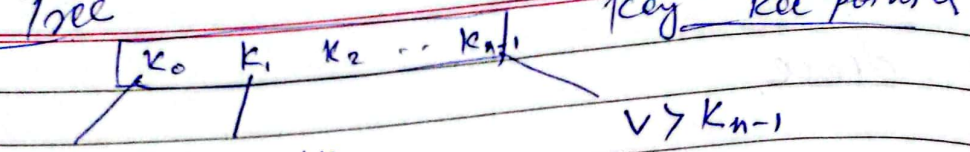


Binary tree and B^+ tree → Balanced multiway search tree



In BST, each node is mapped to one block.
In Balanced MST, each node has more values hence no. of nodes decrease, hence no. of blocks decrease.

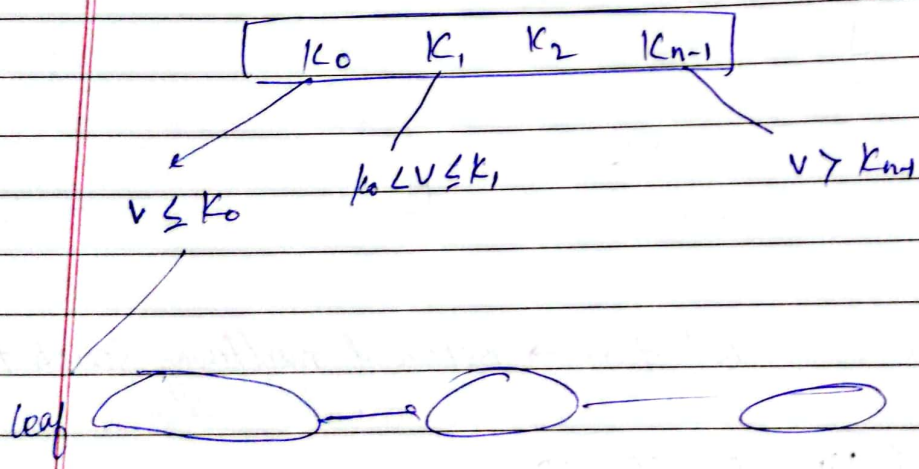
B Tree



ith subtree
$K_{i-1} < V < K_i$

n keys
↓
n data rec/block pointer
→ (n+1) subtree pointers

B^+ Tree



No. of blocks is and depth is likely to be less w.r.t that of B tree.

Non-leaf
n keys
n+1 child pointer

leaf
n keys
n rec pointer
+1 pointer to next leaf.

SECURITY

DBA (Database Administrator) creates user.

CREATE USER userid
IDENTIFIED BY pwd.
DEFAULT TABLESPACE tsname (if not assigned, system tablespace)

→ If a user has access to no. of tablespaces, when he creates a new table, it will either go to a default tablespace or the tablespace can be specified with. →

~~TABLESPACE~~

CREATE TABLE ..
(

) TABLESPACE tsname;

ALTER USER Userid,
IDENTIFIED By pwd,
DEFAULT TABLESPACE ..;

what a User can do?

System level privileges
(Granted By DBA).

- connect (connect to database)
- RESOURCE (can create own object)

Object level privileges
(table, view, index ..)
(Granted by owner of the object).

GRANT priv1, priv2 ...
TO userid1, userid2 ...

REVOKE priv1, priv2 ...
FROM userid1, userid2 ...

→ with GRANT OPTION, a user with privileges can grant them to a different user.

DBA
GRANT RESOURCE (10 M) → K/M
integer
ON TABLESPACE tsname.
TO userid1, userid2.

VIEW
logical table

CREATE VIEW viewname
AS
SELECT * FROM STUDENT
WHERE BCODE='CSE';

VIEW
→ SINGLE TABLE
→ MULTITABLE (SELECT).

INSERT INTO MYVIEW VALUES
SELECT *
FROM MYVIEW.

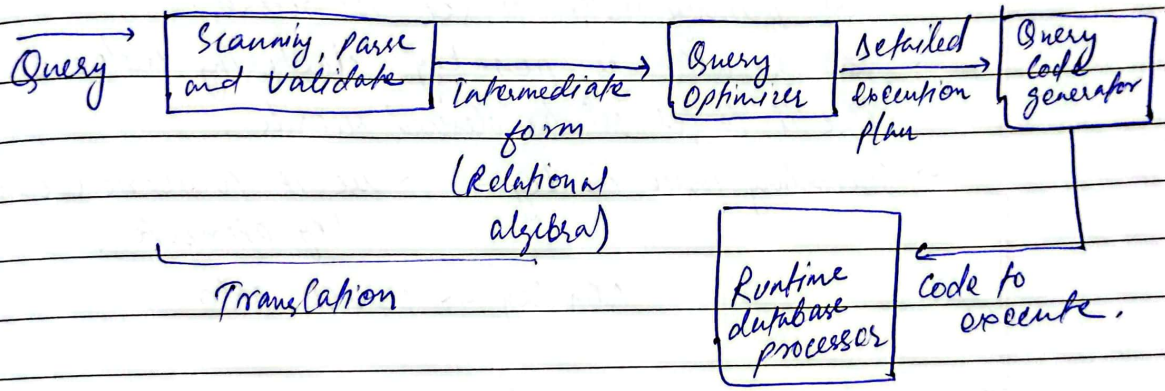
DROP VIEW viewname.

Query Processing

DML Statement to DBMS → Query Optimization.
Non-procedural
↓
How to execute it?

→ Given a non-procedural query, making a detailed and efficient but equivalent representation is the task of query processor. Formulating the efficient strategy.
↳ Query optimization.

2 Steps → 1) Translation
2) Optimization.



Translation - scan the query, parses the same to obtain tokens, validates the rela. and attributes, checks the syntax.
↳ Data dictionary.
View name → replaced by view defn.
Generates the equivalent expression in intermediate forms.

Optimization (2 phases)
1) Expression level Optimization
Relational algebra exp → Transform into efficient but equivalent forms.
→ reduce disk block access.

→ Apply select operation as early as possible (reduce no. of tuples to work with at an early stage).
→ Take the projection as early as possible of attr. required in final output and those needed for intermediate processing.
→ When doing joins ($a_1 * a_2 * a_3$).
The join which produces less tuples will be done first.

2) Detailed Strategy

Query

* involves a simple ~~cond~~ criteria (based on a single attr.)
→ Linear search.
→ If the data is sorted on the search attr. then binary search
→ Search attr. is a key attr. and primary index exists
use primary index to get the record.
→ Search attr. is non-key, clustering index exists.
Use it to get all the desired records.
→ Secondary index on key ⟶ use it to retrieve based on key attribute.
Non key → search attr.

* Based on complex criteria (involving multiple attributes).
Conjuctive (AND) Disjunctive (OR)

→ For every simple criteria → make the strategy.
→ For a one condn.: indexed search is possible (retrieve the records) → on these verify the other criteria.
→ For few cases - efficient (except linear search) strategy is possible. Apply individually to obtain set of records.
↳ Intersection.
Check remaining criteria.

JOIN

R1 x R2

n1 tuples n2 tuples

worst case

n1 → n1 x n2