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 / waste 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.

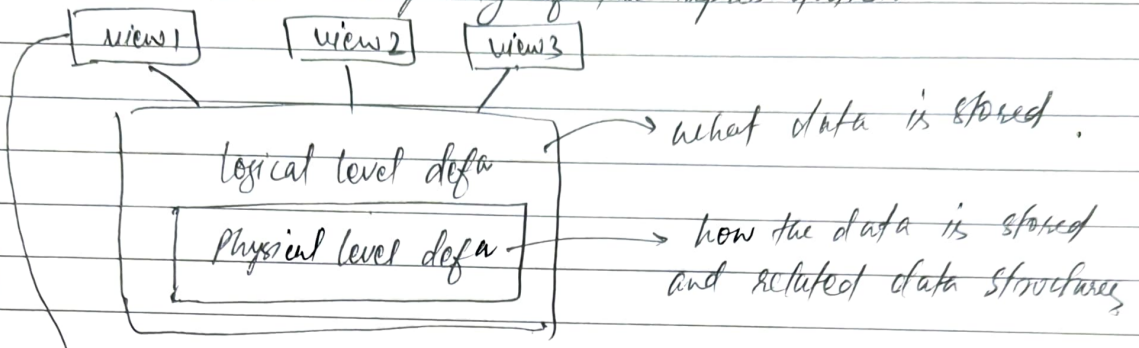
4) Concurrency control
   concurrent access
→ simultaneous access to same data by multiple processes.
→ If not controlled → improper updates.

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

6) Integrity enforcement
→ data must ~~certify~~ satisfy certain conditions / constraints.

Data Abstraction :

→ To hide the complexity of the representation



→ part of the database accessible to a user in the corresponding view.

Data Independence

→ Change in the physical defn does not lead to change in application programs.

→ physical level independence is achieved.

difficult to achieve

→ Change in logical defn does not lead to change in application programs. → Logical level Independence.

Database Schema and Instance

Schema → Overall structure of database.

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

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

Database Model

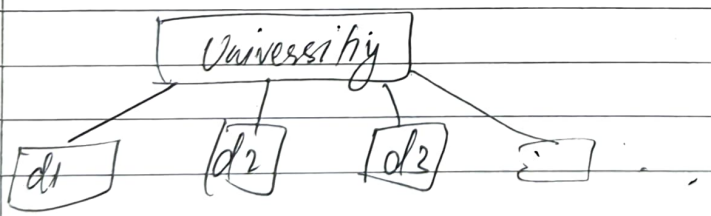
→ Underlying the database, there is a data model.
↳ representation of data and their interrelation, constraints on data, etc.

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/association.

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 database.

DML

(procedural)
what data to ~~work~~ 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.
~~And~~ 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.

→ procedural features are also required.
control struct/loop variables etc.

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

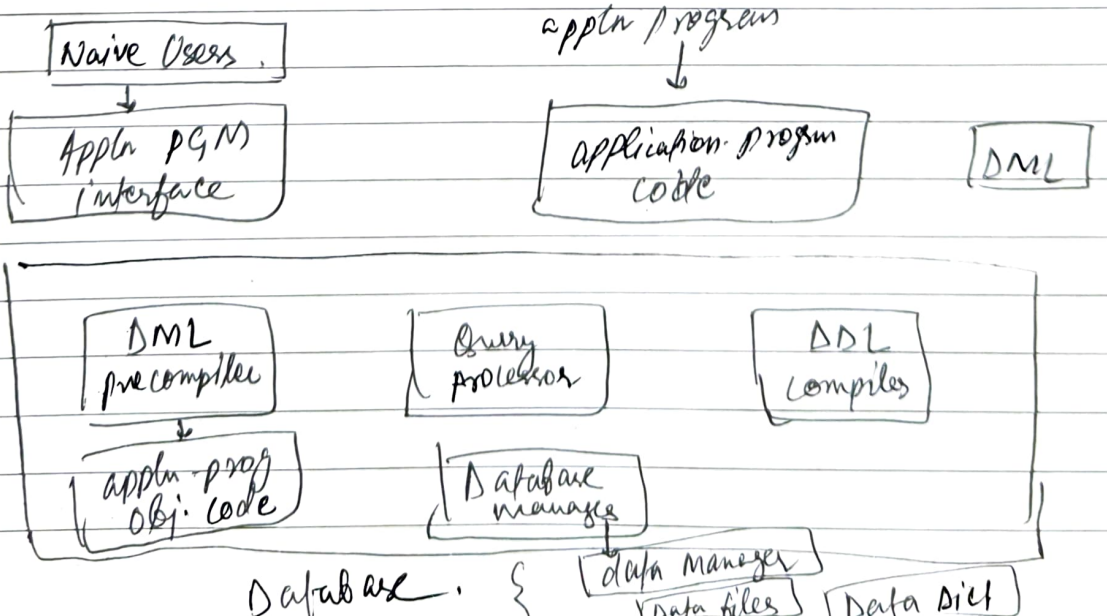
DML pre-compiles, converts DML statements into equivalent host language. ← Inside that, use DML to interact with database. (embedded DML/SQL)

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 of is a set of values → table / file of records.

Roll	Name	DOB	Email
1	-	-	-
2	-	-	-
3	-	-	-

Set of related values that represent an entity.

Each row of the table → tuple → describes an entity.

Attribute → column Header.

Relation, Tuple, Attribute } → domains.

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.
↳ appln dependent.

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

Relation Schema / Intension of a Relation.
→ specifies the overall structure of the ~~re~~ relation.
→ A relation schema is defined as:
$R(A1, A2, \ldots, An)$.
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 $R(A_1, A_2 \dots A_n)$

↓

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

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

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

Absence of any value.

Time varying

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

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

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

$t[A_2]$

value of attribute 2 in that tuple.

Characteristics of a relation:

→ No two tuples are same.

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

→ tuples are unordered.

Relational model does not depend on ordering of the tuple.

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

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

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

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

$v_i \in Dom(A_i)$,

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

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

→ 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, SK of R. ($SK \subseteq R$)
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 key 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 in~~ 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 rel^n and constraints).
schema of a relational database is $\{R_1, R_2 \dots R_n\}$ where R_is are the schema of individual relations and set of integrity constraints.

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

Integrity Constraints and Foreign Key

Referential Integrity :.

Dept (Referenced)

(PK) DCODE	DNAME	...

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 be present in referenced relⁿ. else not allowed.	No Impact.

Operation	Referencing Rln	Referenced Reln
Modifying the tuple.	If FK is ~~changed~~ value is changed (~~not suggested~~). then new value must be in referenced reln else not allowed.	If PK is getting changed (not suggested) and old value is referenced by referencing reln, then not allowed.
Deleting a tuple	allowed	If PK is value is present in referencing reln. then not allowed.

Relational Algebra and Relational Calculus

→ To work with relational database.

→ Relational model → Relational algebra, Relational Calculus

Procedural	Non-procedural
↳ Set of operations	Expression:
Expression:	↳ Data that we need.
Sequence of operations on a rel^n.	(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 DT (DCODE).

select operation.

$\sigma_{predicate}(R)$

predicate: ↓ && || <, >, =, <=, >=

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

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

O/p relⁿ:
↳ Schema → same as I/p relⁿ.
State → set of tuples satisfying the predicate.

→ select operation is commutative.

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

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

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

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

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

$R1(A,B,C,D)$.

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

$\rho_{R2}(R1)$.

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

Cartesian Product

$R1 \times R2$

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

R1 → degree $n1$ R2 → degree $n2$.

$R1 \times R2$ → degree $n1 + n2$

R1 → no. of tuples t_1, R2 → no. of tuples t_2.

$R1 \times R2$ → no. of tuples $t1 \times t2$

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

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

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

Fundamental Operations	Derived Operations
SELECT	JOIN.
PROJECT	Natural Join.
RENAME	INTERSECTION.
CARTESIAN PROD	
UNION	
MINUS	

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.

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

→ 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.
$\text{DOM}(A_i)$ is same as $\text{DOM}(B_i)$ 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 tuple 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, DCODE $\mathcal{F}_{SUM(SCORE)}$

O/P : ROLL DCODE SUM-SCORE

find rolls of which sum of score > 500.

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

$\sigma_{TOTALSCORE > 500}\, T$.

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

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

→ Lets 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.
$R2(X, Z)$

X → subset of attributes

R1 outer Union R2 | $X \cup Y \cup Z$

$t_1 \in r_1(R1)$
$t_2 \in r_2(R2)$
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 null
for t_2 with no match in $r_1(R1)$, then for it Y is null.

→ 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 $T \Rightarrow Y = Z - X$
$X \subset Z$.

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

$T = R \div S$

↓

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

$T_1 \leftarrow \pi_y(R)$ → All roll nos present.

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

→ All SCOBE 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 ~~students~~ 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.

classmate
Date
Page

$(\forall x)(P(x)) \rightarrow (\exists x)(P(x))$ ← 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 — collection of simple attr.
Divisible.
(Appln. specific).

Attribute

- Single Valued OR Multivalued.
  - (For a tuple, attr. can have only one value).
  - for entity,
  - A relationship
- Stored Vs Derived.

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

Key Constraints

→ Unique value for each entity in the set.

STUDENT (NAME, ROLL, Ph-No, Age, DOB)

Ph-No → can have multiple values.

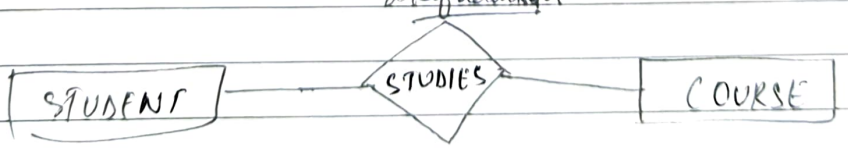
Age → derived.

Relationship

→ Association between entities.

→

Relationship Set - Set of similar relationship.



classmate
Date
Page

Constraints on Binary Relation

- Mapping Constraint
- Participation Constraint.

Structural Constraints

$A \longleftrightarrow B$

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(V2) P(Vn) Vi is the value set of Ai

→ Value set (N) = P(V1) x P(V2) x . . . P(Vn)

Participation Constraint

→ Also, known as minimum cardinality constraint.

→ Total, partial.

Total participation.

→ An entity type totally participates in a relation, mean each entity from corresponding entity set must get associated 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.

1) 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 rela, copy key attr.

PH_LIST (ROLL, PH_NO)
FK & composite key

Roll will be a foreign key also

Mapping 1:1 Binary relation :

A (a1, a2 -- an)
B (b1, b2, ... bn, a1)

Foreign Key based approach
→ Better in the entity type that totally participates in the rela, 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.

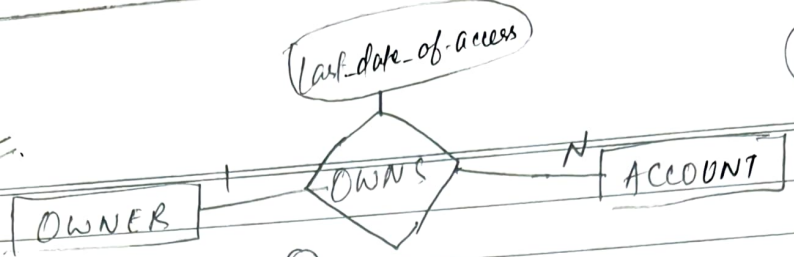
Merged relation approach
ARB (a1, a2 -- an, b1, b2 ... bn, γ).
→ If both are totally participating, else lots of null values.

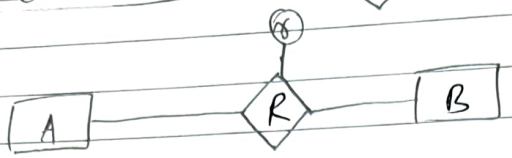
Cross Referencing RELN / RELATIONSHIP RELN / LOOKUP TABLE approach

A (a1, a2 -- an)
B (b1, b2 -- bn)
R (a1, b1, γ)

Seperate 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~~ rela.

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 and it will be foreign key.

→ Also copy attr. of rela (if any).

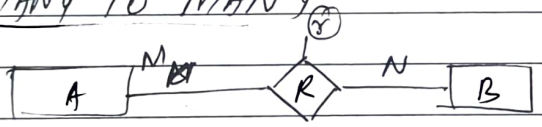
Lookup table

$R(a_1, b_1, \gamma)$

key of many side → key.

copied key of entity type → FK.

MANY TO MANY



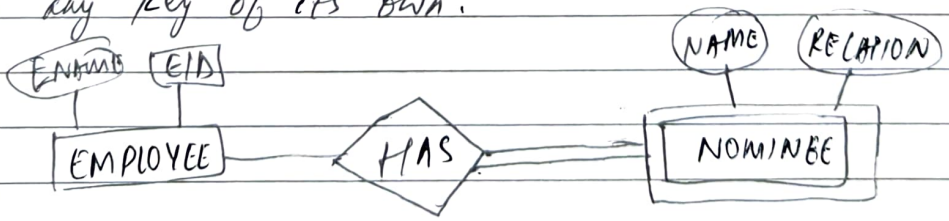
$A(a_1, a_2, \ldots a_n)$
$B(b_1, b_2, \ldots b_n)$

LOOKUP TABLE APPROACH

$R(a_1, b_1, \gamma)$
FK FK
composite key

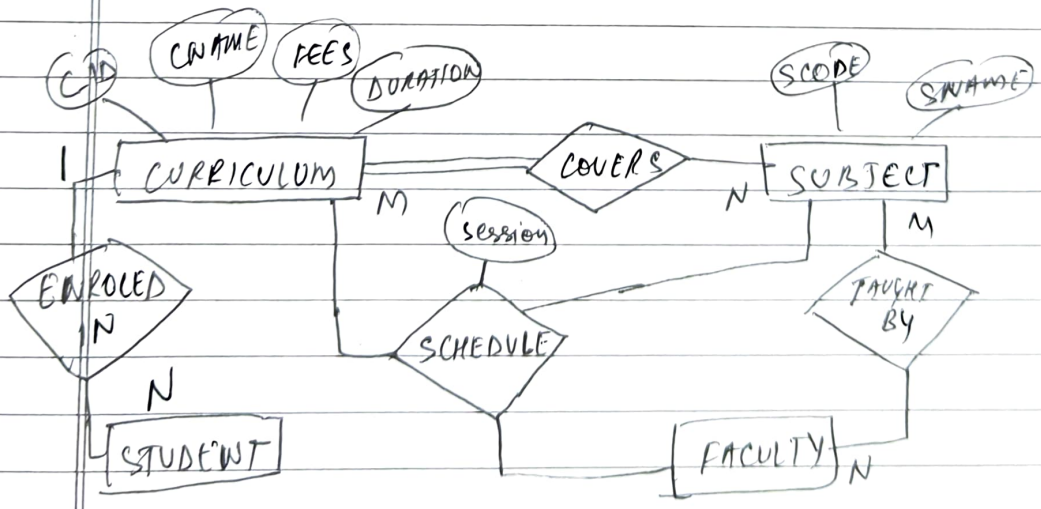
Weak Entity Type

→ 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 relⁿ.
Attr - Its own attr. ∪ key of the entity on which it depends.
NOMINEE (ECODE, NAME, RELATION).
(FK)
Key ⇒ Partial key ∪ 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 rel^n.

for each ~~rel~~. subclass, one rel^n.

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

Additional attr SUB1 (A1 (FK), B1, B2, B3).

SUB2 (A1 (FK), C1, C2, C3).

→ No relation for superclass.

for each subclass, one rel^n.

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 subclass2

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

→ key of superclass will be the key.

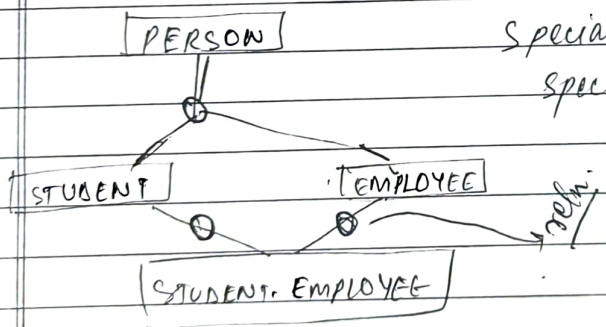
Disjoint → Single type
Overlapped → Multiple types

TYPE → Binary string of length N
N ⇒ $T_i = 1$, belongs to type c_i.

→ 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.

P. T. O

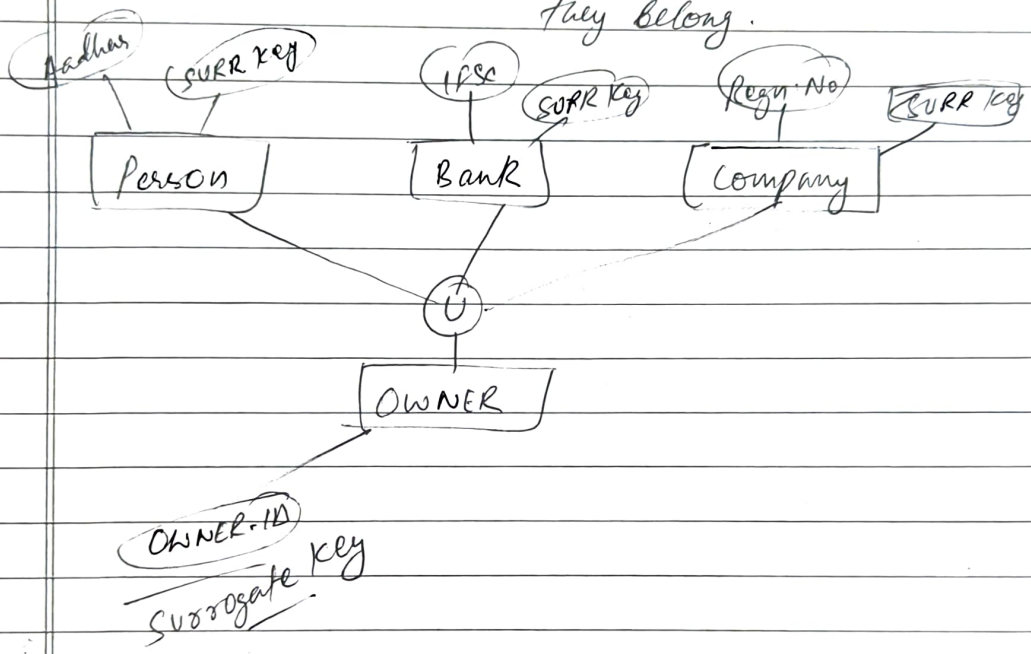
SUBCLASS

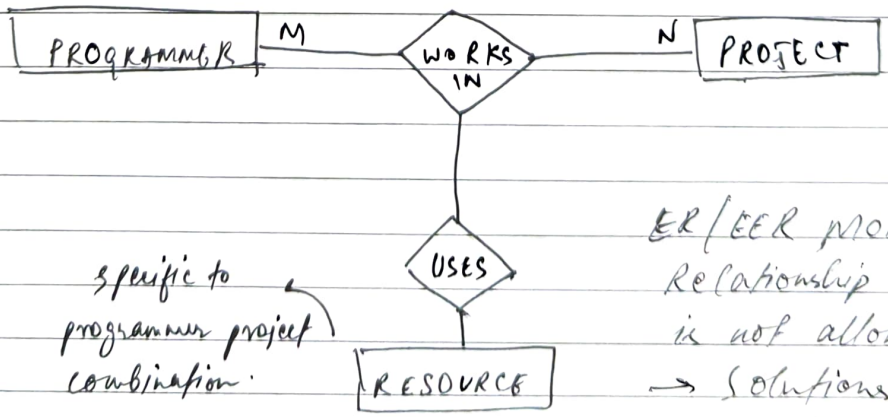
↓ Subset of intersection of superclass elements . .

Category

↓ Subset of union of superclass elements .

Attribute of category => depends on which superclass elements they belong.





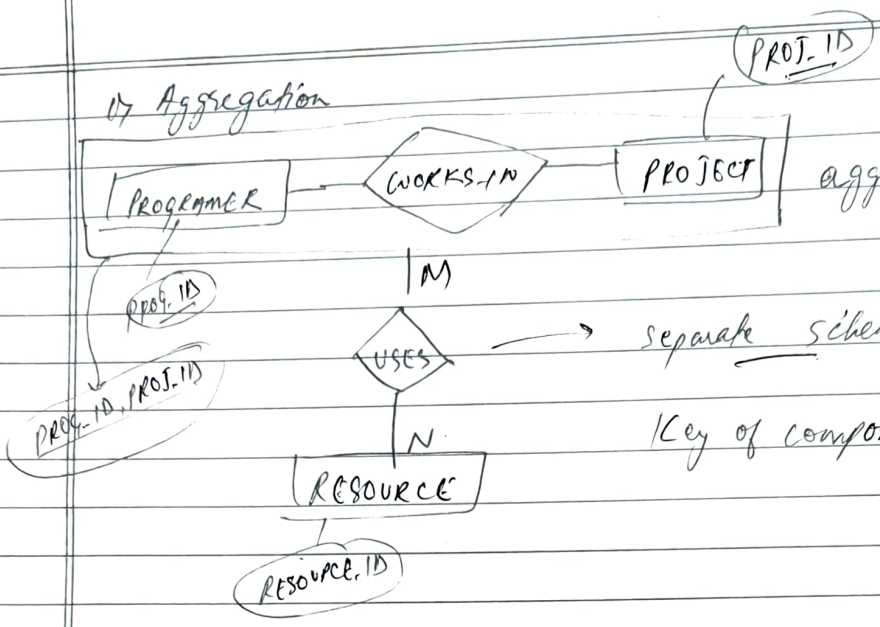
specific to programmer project combination.

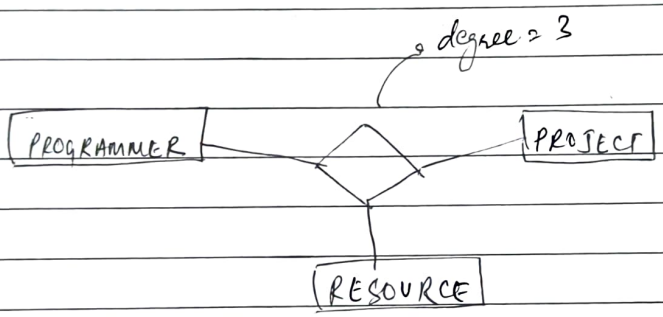
ER/EER MODEL

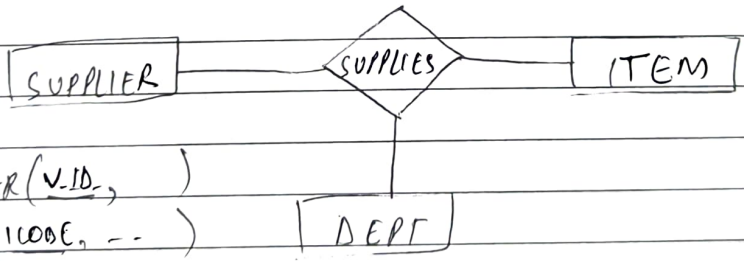
Relationship with relation is not allowed.

→ Solutions in next page.

17 Aggregation

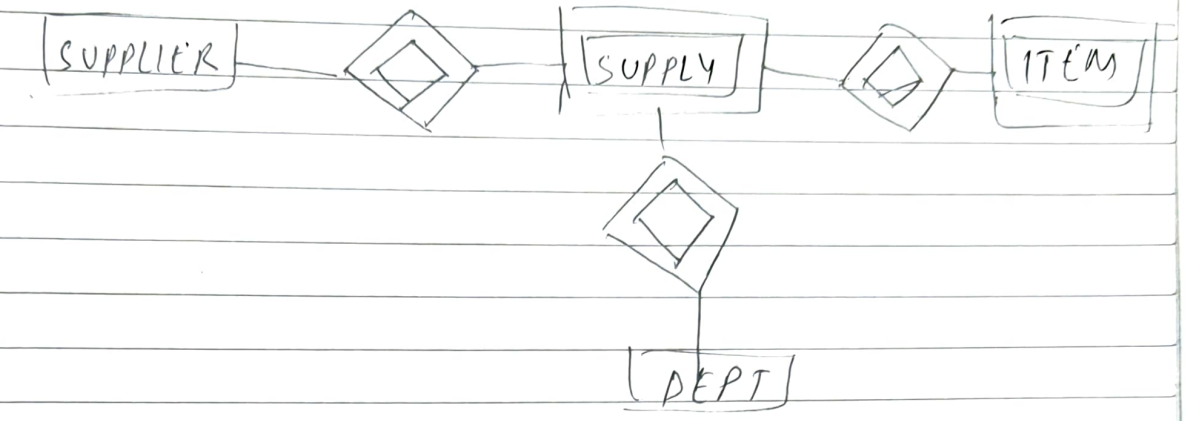


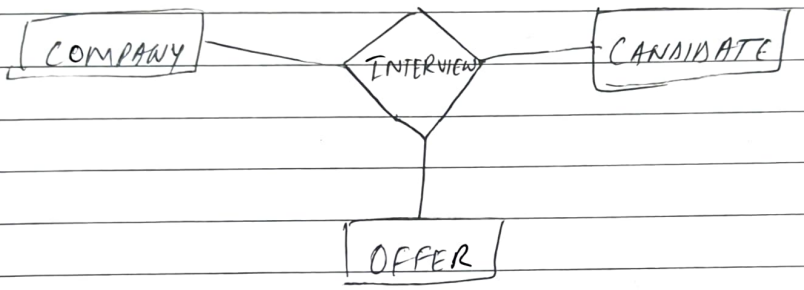


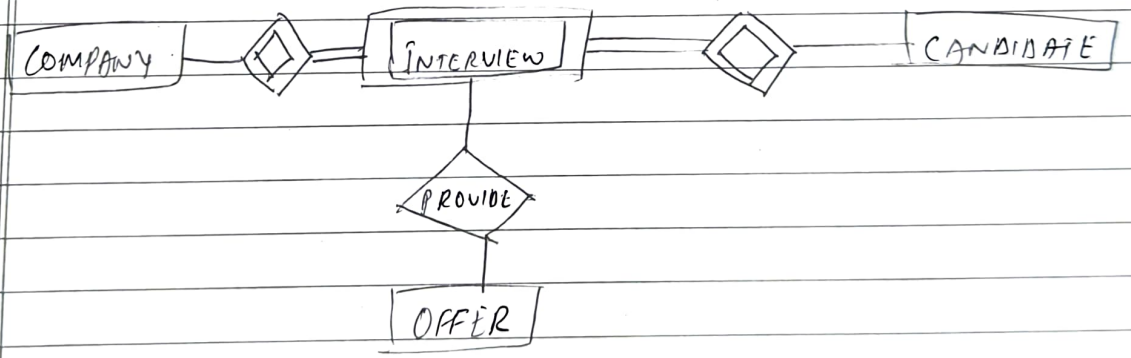


→ SUPPLIER(V_ID,)
→ ITEM(ICODE, - -)
→ DEPT(DCODE, - -)
→ SUPPLY(VID, ICODE, DCODE, QTY, - - -)

→ 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.

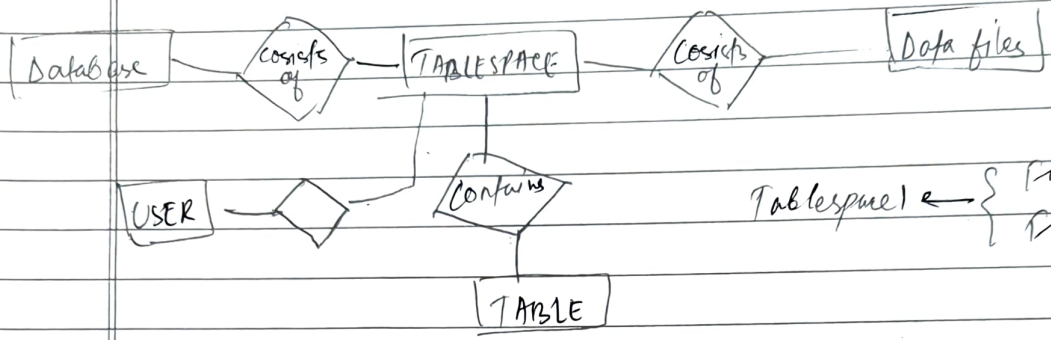

SUPPLIER
SUPPLY
ITEMS
DEPT


COMPANY
INTERVIEW
CANDIDATE
OFFER


COMPANY
INTERVIEW
CANDIDATE
PROVIDE
OFFER

SQL (Structured Query Language)

→ DDL statement ⇒ to define a rel^n.



User1 → Tablespace1 → Table1
Table2
Table3.

User2 → Tablespace1 → Table4.
Table5

System tablespace → gets created when Oracle is installed.

Eg – DEPT (DCODE, DNAME).

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

→ press enter.

→ termination of the command 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 NUMER(3,0) CHECK(SCORE >= 0)
FOREIGN KEY (ROLL, SCODE) REFERENCES
ATTENDANCE (ROLL, SCODE)).
GRADE CHAR(1) CHECK(GRADE IN ('A', 'B', .. 'E')

OR

GRADE CHAR(1) CHECK (GRADE BETWEEN 'A' AND 'E')

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 ('01')

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 (DOB, '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 * P
     FROM STUDENT
     WHERE NAME LIKE 'RA%'
```

→ RAM, RAHIM.

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

'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^
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)

classmate
Date
Page

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)
↓ 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 posn, which occurrence)
↓ 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,
ORDER BY SCODE

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

ROLL
1
1
1
2
2
2

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 SUBJECT S, 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 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 = 'S1'
```

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 rel^n. of outer query say, candidate tuple.
For the cand. key, innery 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 DNAME FROM DEPT D
    WHERE EXISTS
    ( SELECT * FROM STUDENT S
        WHERE D.DCODE = S.DCODE)
```

If it returns at least 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, SUB_CODE
)
```

```
ALTER TABLE tablename.
ADD (column descrp1,
        column descrp2 )
MODIFY ( column descr1,
        column descrp2)
```

DATABASE DESIGN :

ER Model → Mapping to rel^n.

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 —

Eg. STUD(RollNo, Name, dob, addr, courseNo, 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 type)

→ 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 $t_1[Y] = t_2[Y]$

Ex- RESULT(ROLL, score, grade)

SCORE → GRADE
X Y

→ if we know, value of X, we know value of Y.
→ for student 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.
Ex→ for SCORE → GRADE
GRADE → SCORE ✗

→ R(A, B, C, D ...)
A → B
B → C
A → C.

Given a set of func. dependencies F specified on rel^n 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 infessed from F using armstrong's axioms will be valid and holds on R.

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