DBMS

30/12/2019

db - centralised collection of inter-related dat
ms - a set of programs to provide an environment for convenient and efficient access and storage of data from/to database.

Datastructure to store data in secondary storage → file
File Processing System.

① Redundancy and inconsistency:
Applications developed by no. of programmers over a period of time.
(Each team - carries the copy of the data they require.
Same data is being maintained by no. of teams.)
If there exists multiple copies of same data, we say it is 'data redundancy'.
(a) wastage of space.
(b) may lead to data inconsistency.
If all the copies are not updated simultaneously, then there is a mismatch of data which we say is data inconsistency.

② Difficulty in accessing data:
Suppose we have a file → "RESULT"
If criteria or requirement for finding a data is changed → new code.
As per requirement, programs are written to access data, if requirement is changed, one may have to modify the program or write a new code.

(3) Isolation of data: Data may spread over number of files. We have to collect it from all files maintaining the correspondence
↳ it is also difficult

(4) Concurrent access anomaly: Same piece of data being accessed by number of users simultaneously. If they modify, then there will be incorrect reflections.

(5) Security: All users must not access or operate on all data.

(6) Integrity enforcement: Data is supposed to satisfy certain criteria or constraints. So, once constraints are specified, DBMS takes care to enforce those.

02/01/2020

↳ Data Abstraction
↳ set of interrelated files and set of programs. → DBMS provides ⇓ data abstraction
⇓
hiding complexity of representing and accessing data from user.

Physical level
↳ How the data is physically stored (complexity)

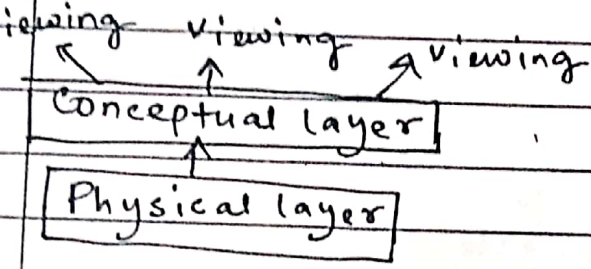
Conceptual / Logical level → what data is stored

View level — for user, whole data will not be visible.

02/01/2020

for eg. struct Student
{ int roll;
char name[31];
int score;
}

→ roll, name, score → logical level.



Independence layer –

① One can develop the application, so that change in the physical definition does not bear any impact on the program. Then physical level independence is achieved.

② If change in conceptual level does not lead to change in application program, then logical level independence is achieved.

Data Model → Any db designed or structured based on model or data model.
Data model is a set of tools to represent data, their interrelation, semantics of data and constraints on data.

We will focus on Record-based model.
file is also a record.

① Relational model: let, we have students db and ~~rollc~~ department db.

02/9/2

to maintain relation

Student — Dept.

Reg No. Name Ph. Dcode — Dcode Dname

In relational model, to maintain relation among records of different type and attribute value which is common to both is stored as a part of record.

(2) Network model:

Student — Department

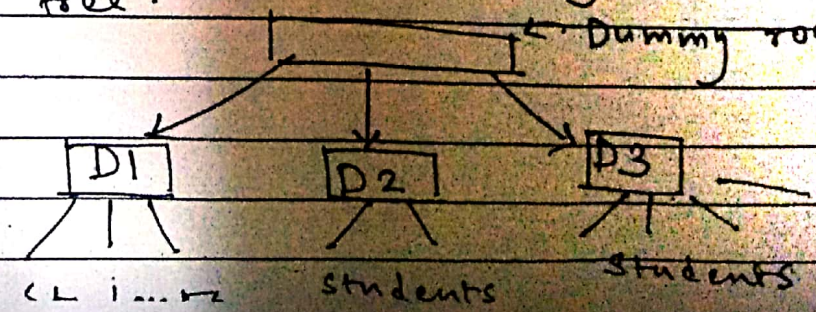
Reg No. Name Ph ~~Dcode~~ — Dcode Dname

ptr. or Link to related record

To maintain relation, as a part of the record, instead of keeping attribute value, pointer to corresponding record is stored.

Drawback :- Relocation of corresponding records will lead to massive change in the former ones.

(3) Hierarchical model: → Organised form of network model. To maintain relation, pointer to related record in kept. But, records of different types are organised according to certain hierarchy. So, it looks like a tree.



02/01/2020

Instance and schema of a data:

Schema stands for structure of a db.

foreg In student db, record has roll (num), name (string), score (num).

Instance — content of the db at a particular instance or at a particular point of time. So, it may vary with time.

schema also may change but less frequent. Frequently changed as it is not static.

In order to interact with db, we need language. Commercially, we have SQL (Structured Query Language).

① DDL ⇒ Data definition language
② DML ⇒ Data manipulation language.

These are required to specify schema of db.
These are required to store, retrieve, update or delete data in/from db.

DML can be of 2 types — (a) Procedural
(b) Non-procedural.

(b) what data is required.
How to get it → is not specified.

to
(a) we will have, specify both — what is required or how to get it.

As a part of SQL → non-procedural.

Major functional units of DBMS:

(1) Database Manager:- It is a s/w module forming the ~~tool~~ core part of DBMS. Major task is -(a) interacts with the file manager (part of OS) to work with data files.
(b) Integrity enforcement.
(c) Security. → Backup a recovery.
(d) In case of failure, how to recover.
(e) Concurrency control.

(2) DDL compiler: It translates DDL statements.
Basically it generates meta data which is stored into Data Dictionary
↑
part of db.

In one place of db, in Data Dictionary (a) Name of db, (b) Who is operator/owner? are stored.
(a) Name of db and associated attributes are stored in Data Dictionary.

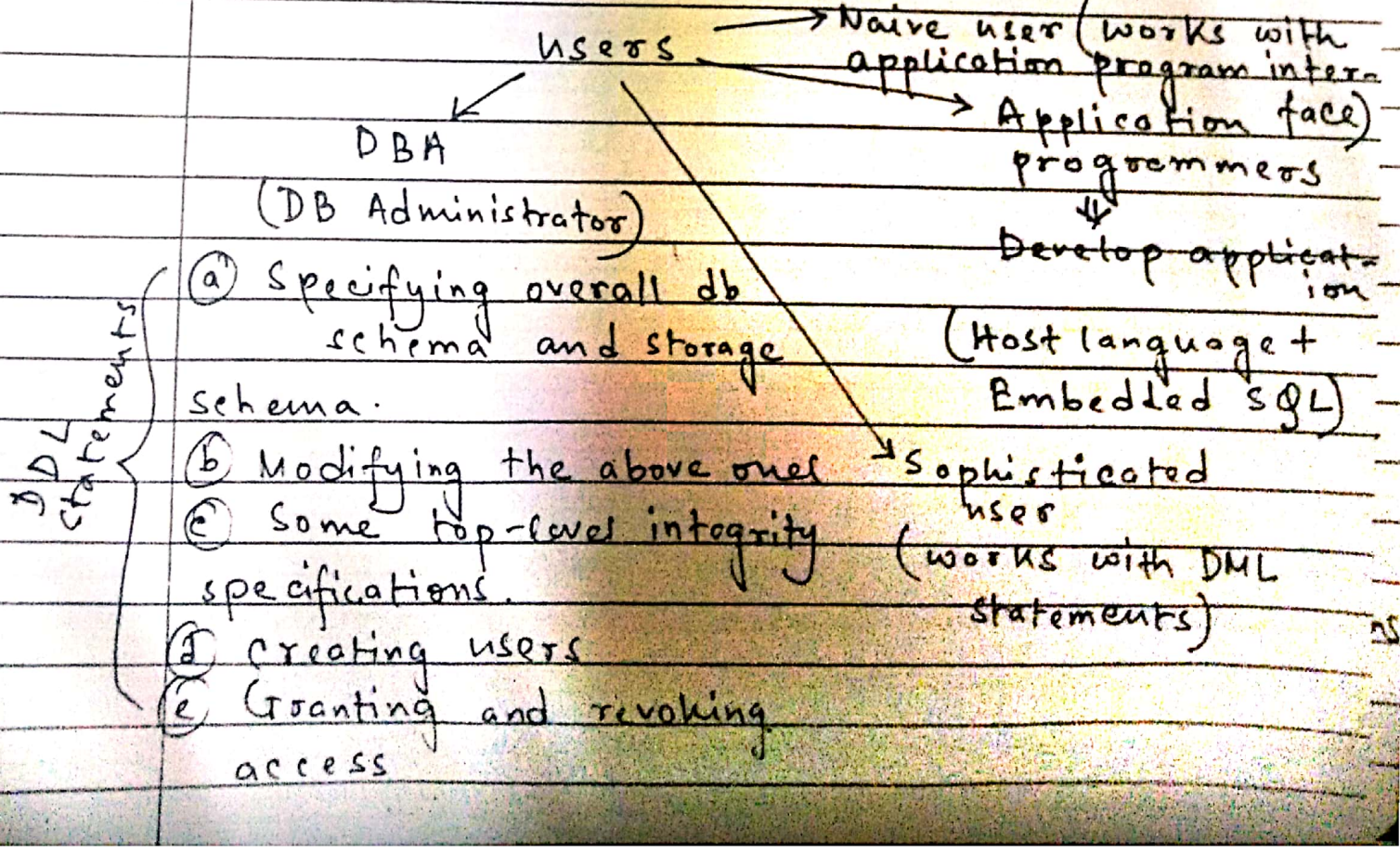
Metadata - is some information / connection about the data, ie data about the data.

(3) DML Pre-compiler: DML statements that will be used to interact with db are non-procedural. But, while developing an applications, procedural constructs are required.

This helps in interacting with db efficiently so, applications developed using language that provides procedural support in that program, DML statements are embedded to interact with db. Sometimes, it is called embedded SQL and language in which application has been developed is called Host Language.

So, DML precompiler converts embedded DML statement or embedded SQL to Host language.

(4) Query Processor:- ~~It generates~~ A query submitted by user can be accomplished by no. of query processors. It to do so finds an efficient way and makes detailed execution plan.



02/01/2020

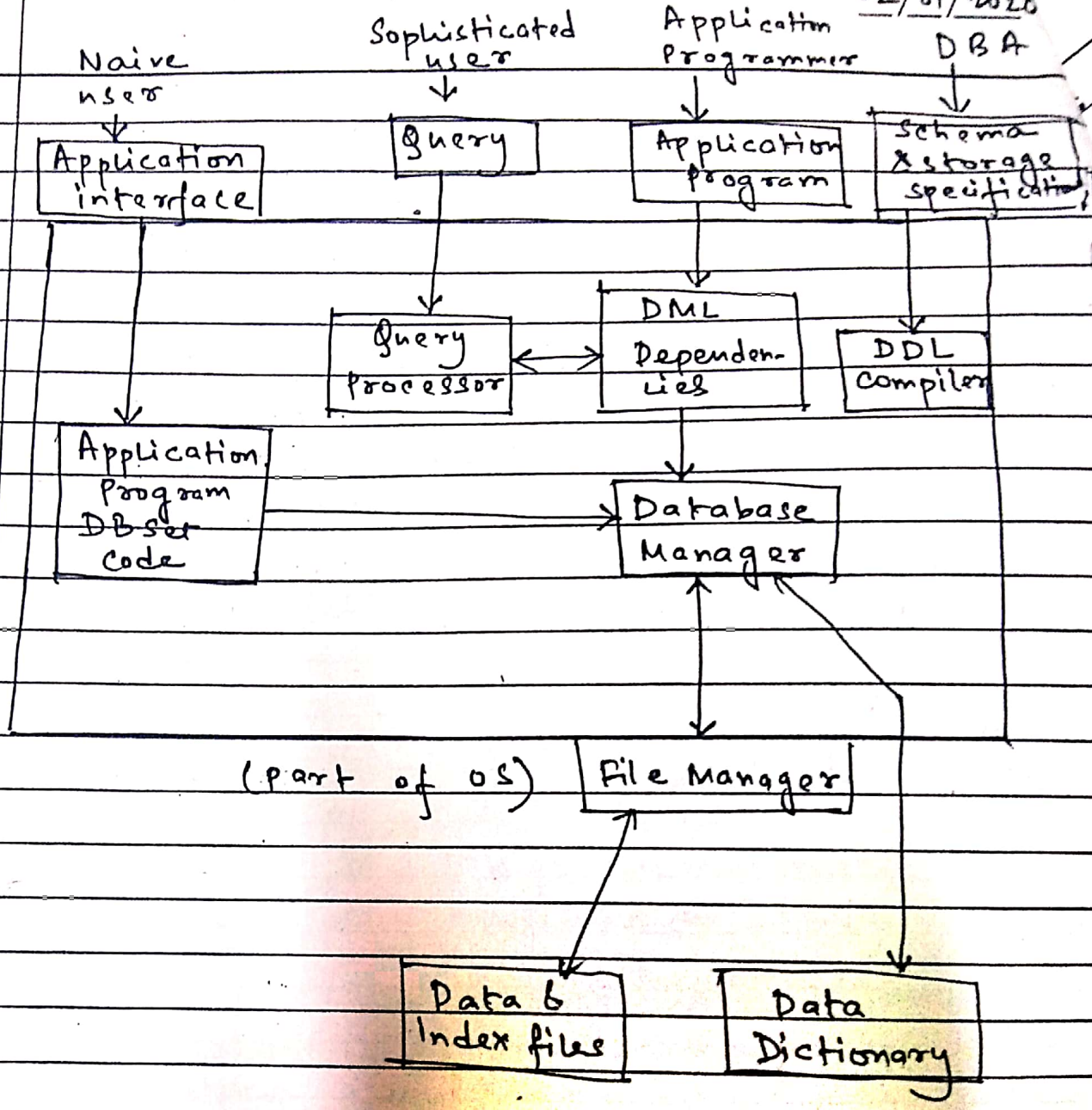


Fig: Overall structure

06/01/2020

Relational model

db = collection of relations

Information

COL Relation ⇒ file consisting of records
Table of values

Each column = has a name
↓ attribute

Row ⇒ collection of related data.

Domain ⇒ set of (atomic) values → specific to appl.n
← data type

An attribute has a domain

Name: 1st name, Middle name, Last name

By specifying data, we specify domain.

Relation
(Intension) Schema
(Extension) State of relation

A relation schema is defined as $R(A_1, A_2, \ldots, A_n)$ where R is the relation and $A_1, A_2, \ldots, A_n$ are the attributes. Each A_i has a domain D_i.
Attribute is the name of role played by its domain in the relation.

Relation and attributes help to interprete data.

A relation or state of a relation of $R(A_1, A_2, \ldots, A_n)$ is denoted as $r(R)$.
This is a set of n-tuples $\{t_1, t_2, \ldots, t_n\}$.
Degree of relation = no. of attributes in schema

For time being, t_i is an ordered collection of n values $\langle v_1, v_2, \ldots, v_n \rangle$ where v_i is the value the A_i coming from Domain (A_i) or it is null.
↳ Absence of any value

Student (Roll, Name, Score)

Roll	Name	Score
1	ABC	95
2	XYZ	96
3	PQR	97
⋮	⋮	⋮

A relation $R(A_1, A_2, \ldots, A_n)$ is a mathematical relation of degree n on $DOM(A_1), DOM(A_2), \ldots, DOM(A_n)$. &

$$r(R) \subseteq DOM(A_1) \times DOM(A_2) \times DOM(A_n).$$

$$|r(R)| \leq |DOM(A_1)| \times |DOM(A_2)| \times \ldots$$

Characteristics of a relation:

(1) Tuples are unordered. Relational model does not rely on any assumptions regarding the order of tuples.

(2) Attributes in a tuple may or may not be ordered. $t_i = \langle v_1, v_2, \ldots, v_n \rangle$

$r(R)$ ↳ State of relation $R(A_1, A_2, \ldots, A_n)$ is a collection of mapping.

$v_j = t_i[A_j]$

$v_j \in DOM(A_j)$

ordered collection of values.

06/01/2020

t_i is a mapping which maps from R to D ⇒ union of domain of A_1, domain of A_2, ..., domain of A_n.

t_i is a collection of attribute value pairs. < (Roll, 1), (Name, '—'), (Score, 80)>
< (Name, '——'), (Score, 90), (Roll, 5)>

(3) Value of attribute ∆ Null.
Attribute value ∈ domain of corresponding attribute or it may be null (absence of values).
⇒ Value of an attribute must be atomic and it must be single valued (one value for a tuple).
↑ not relation
Multivalued → multiple values for a tuple.
Relational model ensures 1NF (1st normal form).
Why any attribute value is null?
→ Not applicable.
→ value is not known.
→ Intentionally not shown.

(4) Interpretation – A relation can be taken as a type declaration or assertion.
Student(Roll, Name, Score)
⇒ 1 'ABC' 80
↑ Instance → fact of an instance.

~~Each data m~~

Constraints of data models

(1) Model based constraint → which are inherent in the model itself.
→ tuples are unordered
→ values in a tuple may or may not be ordered
→ attributes are atomic and singlevalued.
→ no 2 tuples are same.

(2) schema based constraints → these can be specified along with schema definition using DDL statements.

(3) Application based constraints → These involve application logic and cannot be handled by model and schema based constraints will require special mechanism. One such mechanism is TRIGGER.

Functional dependency and multivalued dependency — need to be handled at the design time.

$X \rightarrow Y$

If $t_1[X] = t_2[X]$ These are considered
$\Rightarrow t_1[Y] = t_2[Y]$ during normalisation.

Domain constraints:
Primary key constraint:
$R(A_1, A_2, \ldots, A_n)$
Superkey — ~~collection~~ subset of schema such that for any two tuples $t_1, t_2 \in r(R)$

06/01/2020

Candidate key ⇒ is a superkey such that no proper subset of candidate key is a superkey. It is possible to have multiple candidate keys. It is minimal superkey.

If there are multiple candidate keys, designer selects one as primary.

Primary key is used as identifier of tuple
→ Smaller in size is preferred.
→ Alphabetic is less preferred.
→ In the context of application, candidate key by which instances are identified in general should be taken as primary key.

Entity integrity constraint ⇒ as primary key is used to identify the instance, primary key must not be null.

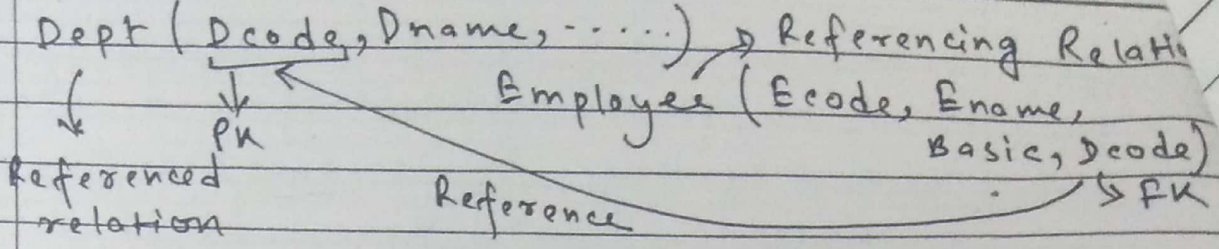
Relation state ⇒ collection of instances
(a) valid state → all tuples must satisfy the constraints.
(b) Invalid state.

DB → collection of relations.
Database schema is set of schema of all individual relations $\{R_1, R_2, \ldots, R_n\}$ and a set of constraints IC.

State of the db → set of state of relations $\{r_1(R_1), r_2(R_2), \ldots, r_n(R_n)\}$.

07/01/21

Referential Integrity and foreign key.



say R_1 be a relation & R_2 be another relation. Let, PK = primary key of R1

FK = subset of attributes in R2 such that

a) Domain of PK in R1 and Domain of FK in R2 are same.

b) for any tuple $t_2 \in r(R2)$ either $t_2[FK] = null$ and $\exists\ t_1 \in r(R_1)$ such that (*)

R1 is the reference relation and R2 is referencing relation and the whole thing is known as referential integrity.

Referencing relation	Ref. relation	DML operation	Impact of Foreinkey in
Add record	→		No impact of foreign key. Primary key is unique here. Allow if the forein key value being inserted is present in corresponding attribute of reference relation. Otherwise, not allowed.
Modify record		If FK is changed and new value is not there in referencing relations, then not allowed.	If PK is changed and old value is used in referencing relation, then not allowed.

(*) $t_1[PK] = t_2[PK]$

In R2, fk is the foreign key.

Delete record — No impact. If at corresponding PK value is used in referencing relation, then not allowed.

Attendance (Exam Roll, Sub code, DT-Exam)
PK

FK → Primary key ← FK → Individually → not purposeful.

Score (Exam-roll, Sub-code, Score)
PK FK

If 1 student sits in the exam of that subject does not imply, it is him/her

Student (Roll, class-roll).
Sublist (sub-code, Sub-name)
Candidate (Exam-roll, DT-Exam)

Any model must provide language to interact with the relation.

Relational algebra → Procedural → set of operations

Operations to be applied in order to retrieve data.

Relational calculus

Non-procedural
→ no operation
→ specifies data

{t ∈ Employee
t.basic-pay < 500}
{t.name for employee name}

Relational algebra :- A relational algebra expression consists of a sequence of operations applied on one or more relations and as output, it provides another relation.

(a) It provides formal framework for data.

(b) It is used as internal representation for query optimisation and execution.

(c) Commercial SQL statements have lots of similarities.

SELECT Operation:

$\sigma_{predicate}(R)$

Tuple satisfying the predicate will form the state of o/p relation

Schema of o/p relation is same as schema of i/p relation.

Emp (Ecode, Ename, Dcode, Basic)

$\sigma_{Basic \geq 50000}(Emp)$

$\sigma_{Basic \geq 50000}(\sigma_{Dcode='D1'}(Emp))$

$\sigma_{Dcode='D1'}(\sigma_{Basic \geq 50000}(Emp))$

→ commutative

$\sigma_{D_1 \text{ AND } D_2}(R) = \sigma_{D_1}(\sigma_{D_2}(R)) = \sigma_{D_2}(\sigma_{D_1}(R))$

Project Operation:

$\Pi_{attributelist}(R)$

$\Pi_{list1}(\Pi_{list2}(R)) \neq \Pi_{list2}(\Pi_{list1}(R))$

$\Downarrow$ list1 $\subseteq$ list2 ; list2 $\subseteq$ list1

Say, The projection list does not contain any candidate key.

No. of o/p tuples $\leq$ No. of i/p tuples

↳ Due to duplicate elimination.

→ Duplicate elimination projection.

RENAME Operation:

$\rho_{R_1(A_1, A_2, A_3, \ldots, A_n)}$

changing both → $\rho_{R_2(B_1, B_2, \ldots, B_n)}(R_1)$

$\rho_{R_2}(R_1)$ → no change in name of attr.

$\rho_{(B_1, B_2, \ldots, B_n)}(R_1)$ → change in name of attr. only.

$\rho_{(B_1, B_2, \ldots, B_n)} \Pi_{EName}(\sigma_{Basic \geq 50000}(Emp))$

↳ filtering

$\Pi_{EName}(\sigma_{Basic \geq 50000}(\Pi_{Basic, Ename}(Emp)))$

$T(B_1, B_2) \leftarrow T_{(A_1, A_2)}(R_1)$

Assignment

Cartesian Product

R_1 , R_2

R_1 → degree = n1, no. of tuples = N1

R_2 → degree = n2, no. of tuples = N2

$R_1 \times R_2$ → cartesian product.

Degree = $deg(R_1 + R_2) = (n1 + n2)$

No. of tuples = $N_1 \times N_2$

Dept

Dcode	Dname
D1	A
D2	B

Emp

Emp Code	Ename	Dcode
E1	X	D1
E2	Y	D1
E3	Z	D2
E4	W	D3

DEPT X EMP

Dcode	Dname	Emp Code	Ename	Dcode
D1	A	E1	X	D1
D1	A	E2	Y	D1
D1	A	E3	Z	D2
D1	A	E4	W	D3
+ D2	B	—		
		—		
		-		
		-		

Meaningful → Dcodes same.

$\sigma_{Dept.Dcode = Emp.Dcode}$ (Dept X Emp)

X → 50000
Z → 50000
Y → 60000
... → Names whose Basic = same

$\sigma_{Emp.Basic}$ (Emp X ρ_{Emp1} (Emp))

13/01/2020

Relational Algebra:

Basic operations:- Select (σ)
Project (π)
Cartesian Product ($\times$)
Rename (ρ)
Union
Set difference

Set operations:- $R(A_1, A_2, \ldots, A_n)$ AND $S(B_1, B_2, \ldots, B_n)$ are two relations. set operation on them can be done if they are union compatible. → same degree
→ $DOM(A_i) = DOM(B_i)$ for $1 \leq i \leq n$

(a) Union → o/p relation will be of degree same as that of R and S. State includes all the tuples appearing in either R or S or in both. Tuples present in both will appear only once.

Employee (Name, Age)
Student (Name, Age)

$R \cup S \equiv S \cup R$ → commutative
$(R \cup S) \cup W \equiv R \cup (S \cup W)$ → assosciative

Match (Team1, Team2, Date)

$\pi_{Date}(\sigma_{Team1='IND' \text{ OR } Team2='IND'}(Match))$

$\pi_{Team2, Date}(\sigma_{Team1='IND'}(Match)) \cup$

$\pi_{Team1, Date}(\sigma_{Team2='IND'}(Match))$

(b) Set Difference → R − S (union compatible)

o/p relation → schema and degree same as R/S.

Tuples of R which are not in S will be present in o/p relation.

$R - S \neq S - R$ → not commutative

(c) Intersection → union compatible

$S \cap R \equiv R \cap S$ → commutative.

$(R \cap S) \cap W \equiv R \cap (S \cap W)$ → associative

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

↳ Derived set operation, not basic.

Cartesian Product:

$R \times S$

(d) Join operation → $R \bowtie S$

predicate

$\sigma_{\text{predicate}}(R \times S)$

θ-join, equijoin

↓

if predicate is based on equality of attribute values

DEPT(DCODE, DNAME)

EMP(ECODE, ENAME, BASIC, DCODE)

FK

$EMP \bowtie DEPT$

$\sigma_{\text{Emp.Dcode} = \text{Dept.Dcode}}(EMP \times DEPT)$

Emp.Dcode = Dept.Dcode.

13/01/2020

o/p relation → Schema. same

Ecode	Ename	Basic	Dcode		Dcode	Dname
Emp					Dept	

$\Pi_{Ecode, Ename, Basic, Dcode, Dname}$ (Emp ⋈ Dept) Emp.Dcode = Dept.Dcode

Natural Join → Emp ✱ Dept

Same as equijoin, but common attributes which are taking part in equijoin will appear only once in o/p schema.

↳ Ecode Ename Basic Dcode Dname

→ Equality of all common attribute pairs (ie. with same name).

($\int_{Emp(Ecode, Ename, Basic, Dcode)}$ (Emp)) ✱ Dept.

R(A1, A2, A3, X, Y, Z)

S(B1, B2, X, Y)

$R * S = \Pi_{\ldots}$ (R ⋈ S, S.X = R.X AND S.Y = R.Y)

Outer Join → Equijoin operation on R and S. common attribute → X

R(..., X), S(..., X)

R ⋈ S, R.X = S.X

13/01/

left outer join

$R \overset{L}{\bowtie} S$ $\Rightarrow$ all tuples of R will appear in output, if it has a match in S, then corresponding information will be a part of o/p tuple, otherwise, it will be matched with a dummy tuple of S.

$R \overset{R}{\bowtie} S$ → right outer join

$R \overset{F}{\bowtie} S$ → full outer join

Outer union → $R \cup S \rightarrow X \cup Z$

schema of $R \equiv X \cup Y$

Student (Name, Dept, Roll, Social_Sec_No, ...)
Faculty (Social_Sec_No, Name, Dept, ...)
we cannot carry out union operations, but we can carry out outer union.

o/p schema → $X \cup Y \cup Z$
for a tuple in R, if there is a match in S $[t_R[X] = t_S[X]]$
This result into a single tuple in o/p.

For a tuple in R, not matched with S, is padded with dummy tuple of S.

$R \overset{F}{\bowtie} S$ → full outer join based on equality of common attributes anded together) ≡ outer union.

13/01/2020

Division operation → Student(Roll, Name, Skill-No) (R)
Skill(Skill-No, Skill-name) (S)
Roll of students with all skills

$R(X, Y_2)$ $S(Y)$ X, Y are subset of attributes.

$R \div S \Rightarrow$ o/p schema will be X.
(schema of R − schema of S)

$T_1 \leftarrow \Pi_X(R)$ → all roll nos.

Skill No → Y

$T_2 \leftarrow (T_1 \times S) - R$ → roll skill combination which does not exist.

$\Pi_X(T_2)$ → roll no of student which does not have at least 1 skill, i.e. at least 1 skill is missing.

$R \div S \leftarrow T_1 - \Pi_X(T_2) \Rightarrow$ all roll nos. with all skills.

Schema of S ⊂ Schema of R
(Proper subset.)

o/p schema = schema of R − schema of S.

In the output, we will get a set of tuples $\{t\}$ such that for t, there are tuples t_r in R and t_s in S and for every tuple t_s in S, with $t = t_r[X]$ there is $t_r[Y] = t_s[Y]$

Subject(Scode, Sname, full marks, pass marks)
Result(Roll, Scode, Score)

$R \leftarrow \Pi_{Roll, Scode}$ (Result ⋈ Subject
Result.Scode = Subject.Scode AND Score ≥ PM)

$S \leftarrow \Pi_{scode}$ (Subject)

$R \leftarrow S$ → roll with ~~passed~~ all subjects passed

Aggregate function :-

$T(\text{Total-Basic}) \leftarrow \mathcal{F}_{SUM\ BASIC}$ (EMP)

50,000
60,000
NULL

SUM(BASIC) ← o/p schema

$\mathcal{F}_{AVERAGE\ BASIC}$ (EMP)

= 55,000 → NULL will not be considered, it will be ignored.

$T(f1, f2) \leftarrow \mathcal{F}_{function1(attribute)\ function2(attribute)\ \ldots}$ (Relation)

$T(DC, SB) \leftarrow \mathcal{F}_{Dcode\ SUM\ BASIC}$ (EMP)

(Things will be grouped based on this)

$\Pi_{UPPER(Ename)}(\sigma_{\_}(EMP))$

It aggregates information in a group. It works on a group of Tuples. for the group, one tuple is returned.

Average basic for every designation in every department.

13/01/2020.

$T(DC, Deg, AB) \leftarrow {}_{Deode, Desig}\mathcal{F}_{AVERAGE\ BASIC}(EMP) \equiv {}_{Desig, Deode}\mathcal{F}_{AVERAGE\ BASIC}(EMP)$.

15/01/2020

RELATIONAL CALCULUS

- In relational calculus expression, in a retrieval query, we specify what is required unlike relational algebra expression, no operation is specified. In comparison to relational algebra, relational calculus is non-procedural.

- But, using basic operation whatever is expressed using relational algebra, those can also be expressed in relational calculus.
  ⇒ power of expression is same.

- A query language is said to be relationally complete, if it can ~~be~~ expressed every query what can be expressed by relational calculus.

- A relational code in for
  $\{t \mid cond(t)\}$
  ↓ tuple variable
  set of tuples satisfying the condition

  tuple variable ranges over the tuple of a relation → range related
  $\{t \mid EMP(t)\}$
  ↳ set of tuples from relational EMP

- $\{t \mid EMP(t)\ AND\ t.BASIC > 50,000\}$
  t.attr
  t[attr]

  Dept (Dcode, Dname)
  Emp (Ecode, Ename, Deode, Basic)

- $\{t.NAME \mid Emp(t)\ AND\ t.SALARY > 50,000\}$

AND

- {e.ENAME, d.NAME | Emp(e) Dept(d)}
  AND e.CODE = d.CODE
  free variables

- Bounded variables
  accompanied by quantifier
  existential, universal
  (∃x)(R(x) AND ....) ← existential
  there exists a tuple x such that
  {e.ENAME | EMP(e) AND (∃d)(DEPT(d) AND
  d.DCODE = e.ECODE
  AND d.DNAME = "ABC")

Emp	Dept	
— D2	D1	XYZ
— D3	D2	PQR
	D3	ABC
	D4	

- SUBJECT(SCODE, SNAME) → su
  STUDENT(ROLL, NAME) → st
  ATTENDANCE(ROLL, SCORE) → a
  universal
  (∀x) for all
  Range relation
  ⇒ universal
  (∀x)(cond)
  tuple from any relation
  negate

{a.NAME | STUDENT(st) AND (∀su)(∃a)(NOT
SUBJECT OR
AND a.ROLL = st.ROLL AND ATTENDANCE)}
a.SCODE = su.SCODE

15/01/2020

- $(\exists x)(P(x)) \equiv NOT\,(\forall x)(NOT\ P(x))$

equivalent to

$(\forall x)(P(x)) \equiv NOT\,(\exists x)(NOT\ P(x))$

$(\exists(x))(P(x)\ AND\ Q(x))$

$(NOT\,(\forall x))(NOT(P(x))\ OR\ NOT(Q(x)))$

P implies Q

$P \Rightarrow Q$

$(\forall x)(P(x)) \Rightarrow (\exists x)(P(x))$

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

$\Downarrow$ $(\exists x)(NOT\ P(x))$

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

Safe expression ⇒ Relational calculus must result into finite number of tuples. Then it is safe.

$\{t \mid NOT\ EMP(t)\}$

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

20/01/2020

Entity Relationship ~~of~~ model / diagram:

$\Downarrow$

(a) used to capture the data requirement of an application. (b) reflects the requirement at conceptual level.

(c) Shows the different types of entities involved and assosciation / relation b/w them.

(d) also reflects their attributes and constraints.

Step towards db design:

(1) Entity – is an element that exists physically or it may be abstract concept. One entity is distinguishable from another.

(2) Relation – Association among entities.

(3) Collection of similar types of entities ⇒ entity set.

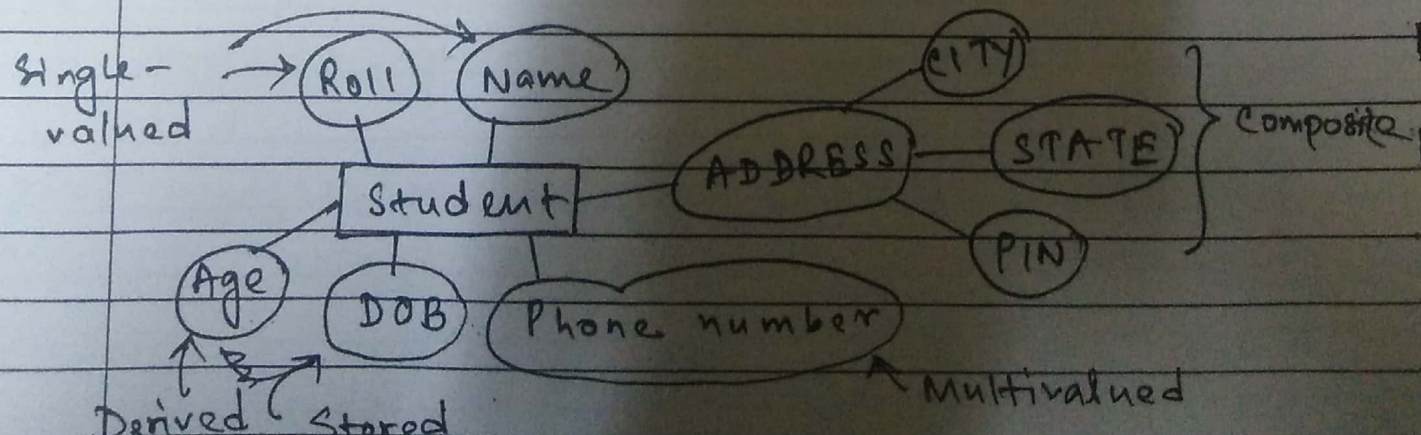
(4) Entity type denotes – (a) structure of an entity set, (b) defines or describes the entity set.
Entity set is described in terms of attributes.

Relational model – everything is relation.
ER diagram – entity, relation.

An attribute maps an entity set to its domain.

for eg. Student

Roll	Name	Score
1	ABC	90
2	XYZ	86



20/01/2020

Entity Type = Schema of relation (intension)
Entity set = state

An attribute ⇒(a) simple (atomic) or composite (collection of no. of simple attributes)

⇒(b) single-valued or Multivalued

An instance (tuple) can have one value

An instance can have multiple values

⇒(c) stored or derived

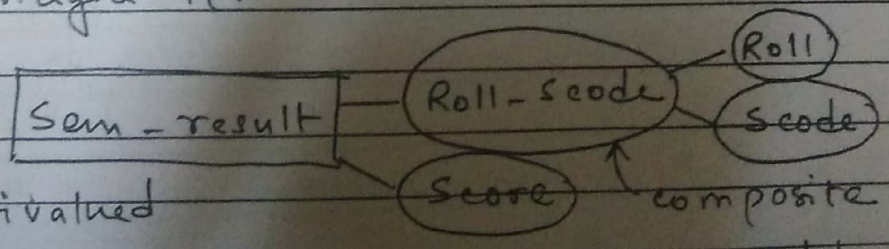
only way to have the attribute value by storing this in db. for eg. name.

If the value can be derived from other stored attributes.

Inconsistency is an issue.
Attribute can have null value (disturbing).

⇒(d) Complex / Nested Attributes → combination of composite and multivalued attributes.

⇒(e) Key attributes → underline primary key to highlight it.



Multivalued

{ Contact-Details (House address, {landline nos}) }

House address

landphone nos.

(Premises no., street, city, PIN)

Multivalued

{contact_details (House_Address (Premises, Streets, City), {Phone_no.})}
↑ Composite + Multivalued.

An attribute → Domain
A → value set, V
↓
set of values that consist from the domain and the attribute can take one or more domains values from this.

$A \Rightarrow V \{v_1, v_2, \ldots, v_n\}$
↓
${}^nC_1$ ← Single-valued (cannot be null)
↑
Mandatory and single-valued
optional and single-valued → ${}^nC_1 + {}^nC_0$
(including null)
Multivalued → ${}^nC_2 + {}^nC_3 + \ldots + {}^nC_n$
without restriction → 2^n → Power set $P(V)$

$A : E \rightarrow P(V)$ ⇒ An attribute is a function for Mapping of entity type to power set.

composite attribute → $A \rightarrow A_1, \ldots, A_n$
↓ ↓
$P(V_1)$ $P(V_n)$

$P(A) = P(V_1) \times P(V_2) \times \ldots \times P(V_n)$

Relationship type defines the association between the entities of different type.

degree = 5

20/01/2020

→ ER model

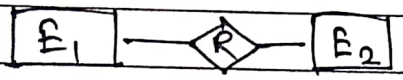
Degree ⇒ no. of entity types involved in a relation.

$E_1, E_2, \ldots, E_n$

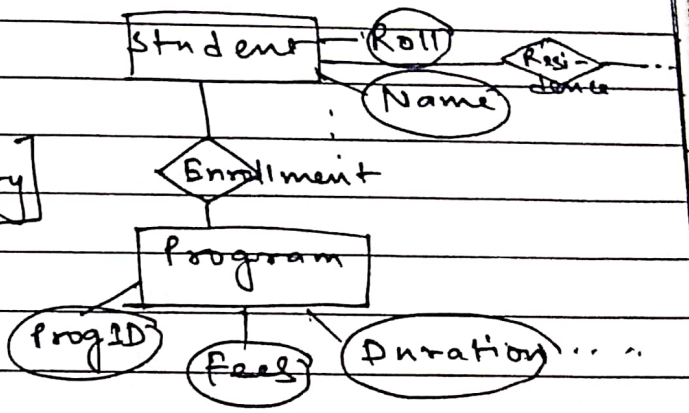
Set of similar type of association ⇒ Relationship Set

$\{(R_1, R_2, \ldots, R_n) \mid R_i \in E_i\}$

Relational model → degree ⇒ no. of relation
→ no. of attributes in the schema.



[Degree = 2 → Binary]



Constraints on Relations:

① cardinality / mapping constraint
② Participation constraint

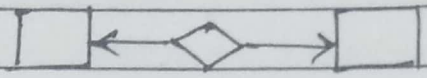
① An element of 1 entity type can have association with how many elements of related entity type.

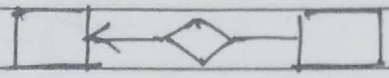
From A to B → one to one – one element of set A can have at most element of set B related to it and vice-versa.

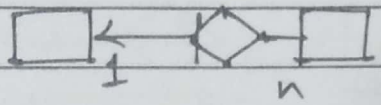
One to Many – From Set A, an element can have relationship with number of elements in set B.

But, an element of B can have at most one element of A related to it.
(Many to one (from B to A).

Many to Many: An ~~relation~~ element from either sides can have multiple relations with the other.


one to one


one to many

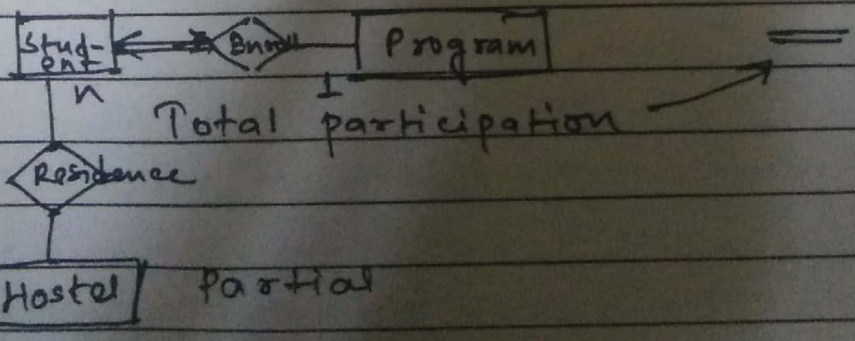


one to many

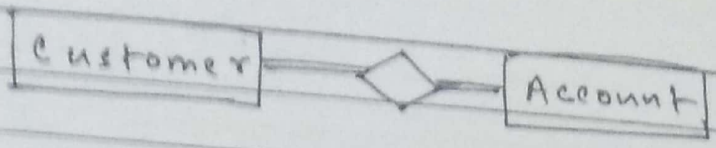
(2) Participation constraint is either partial or total. Taking part in an association is mandatory or not.
If mandatory → Total participation

Every element must have association with elements of other type

If optional → Partial participation.



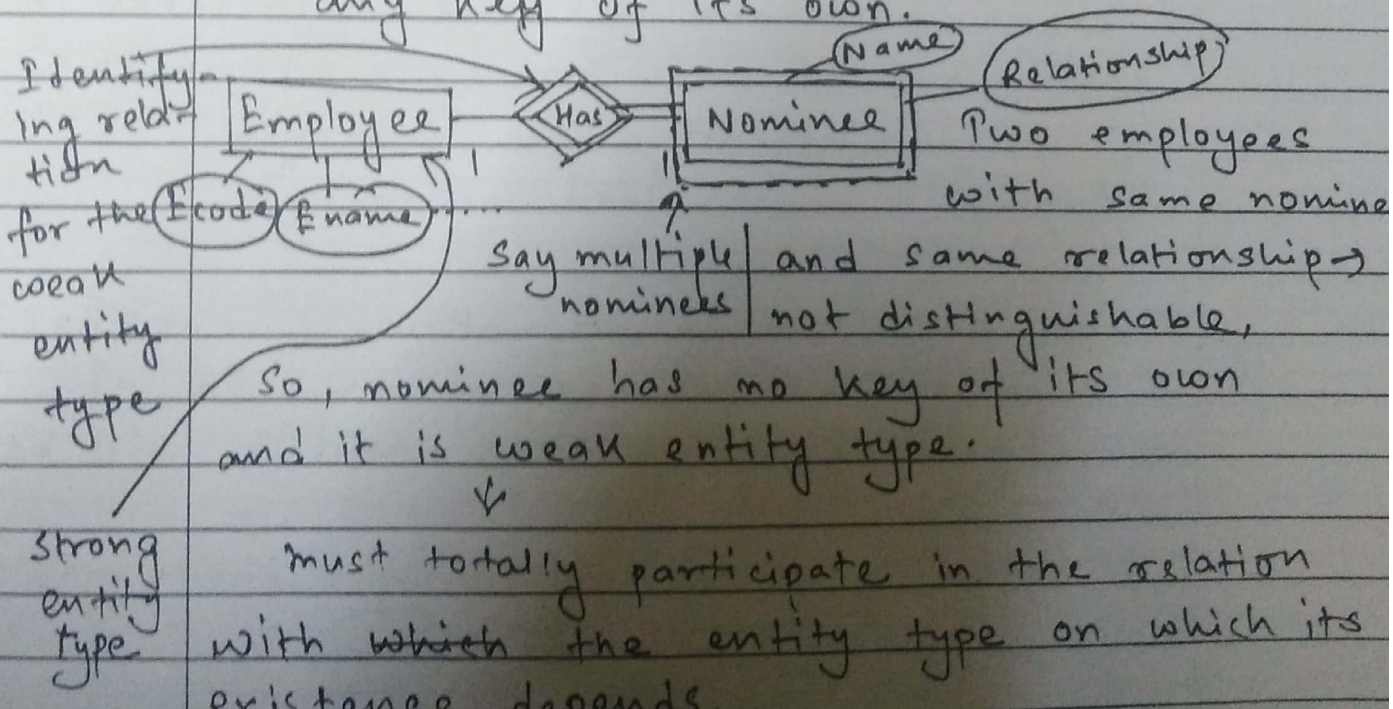
20/01/2020



A customer cannot have more than 1 account → 1 to 1
can have multiple accounts → 1 to n
Joint accounts but not multiple accounts → n to 1
Total participation is also called "existential dependency". → At the very onset of instance creation, it must be associated.

for eg. for an account to exist, we need customer or a student to exist, we need program.

Weak Entity Type :- An entity type w/o any key of its own.



Identify
To find an element with weak entity type
① find the key of entity on which it depends (Ecode)

(2) Traversing the relation (in which weak type actually participates) one can get set of weak elements. (name of nominee)

(3) To identify an element in this ~~relation~~ retrieved set weak entity type has an attribute (e) that can act as key → discriminator

22/01/2020.

ER Diagram to Relation design:

⇓

Relational model.

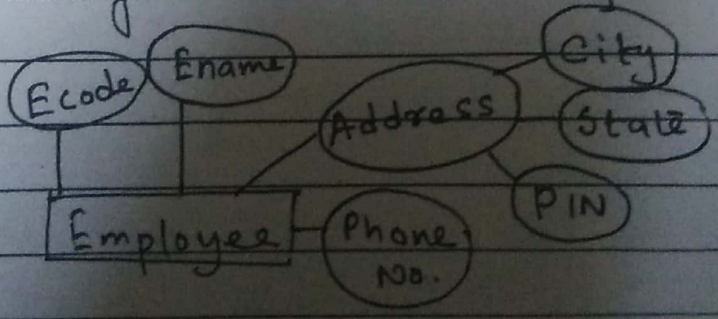
ER Diagram: Entity types & Association: Among them — consider a relation (A/c to relational model)

For each entity type (strong)

→ Schema will have all the attributes of entity type. Replace composite attributes by its constituent simple attributes. (Exclude multivalued attribute).

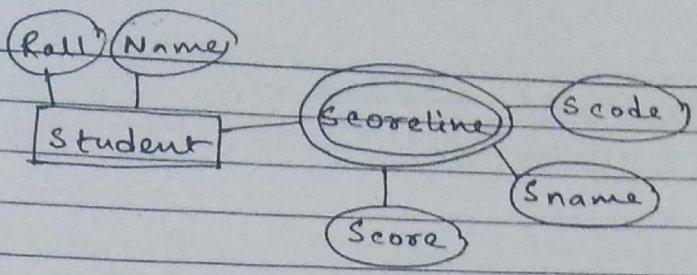
→ For each multivalued attribute consider a separate relation copy of the PK of corresponding entity type and it will be FK —

Referring to the relation for entity type.

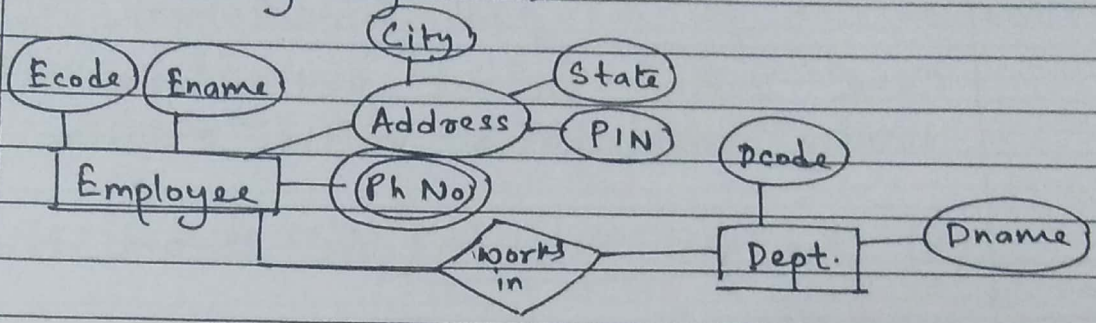


Employee (Ecode, Ename, City, State, PIN, Ph-No)

22/01/2020

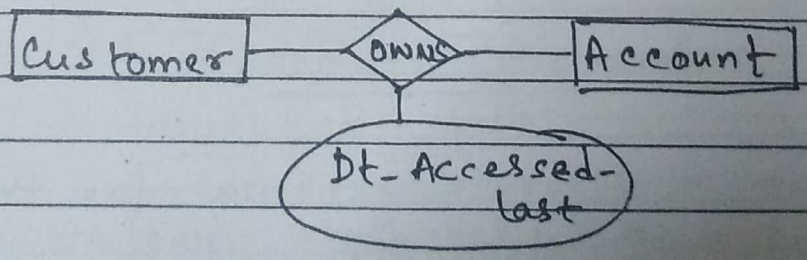


Scode, Sname, Score
(Roll, ~~Scoreline~~)
Primary Key (P.K.)



Dept (Dcode, Dname)
Emp (Ecode, Ename, City, State, PIN)
Emp-ph (Ecode, Ph.No.)

Mapping of relation of ER diagram to Relational Model:



{Ci, aij}

(a) Forein key based approach:

In one side, Keep the Primary key of the entity as foreign key and also copy the attribute of relation (if any).

relation

A(a1, a2, a3, b1, r1)
B(b1, b2, b3)

Customer(Ccode, ..., Acc-no, Dt-last-access)
Account(Acc-no, ...)

Foreign Key

Take action on the entity type that totally participates in the relation.

(b) Merged relation (one-to-one relation):

Combine the entity types and relation into a single schema of relation

ARB(a1, a2, a3, r1, b1, b2, b3)

a1 ↑ Ccode
b1 ↑ Acc-no.

→ Both the entity types are totally participating.

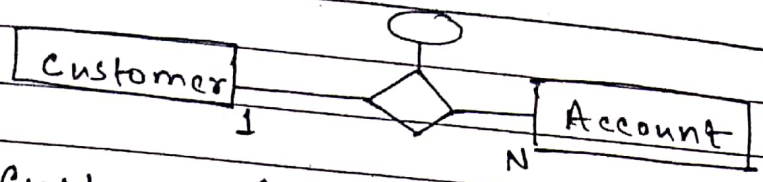
(c) Cross-referencing relation:

Consider a separate schema for the relation of ER diagram.

Attributes → Relation of related entity types and attributes of relation itself (if any).

P.K. of this schema — They are also F.K.

22/01/2020



Customer (ccode, Cname, Caddress)
Account (Acc-no, Balance, Ccode, last-dt-access)
F.K.

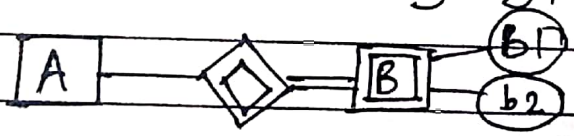
In many sides, copy P.K. of related entity type. (It will be f.k. and also attributes of relation (if any)).

Cross-ref relations: Ownership (ccode, Acc-no, Dt-last-access)
F.K. (f.k. of entity type of many sides)

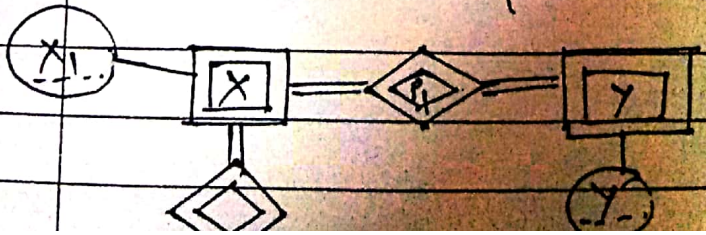
Ownership (Ccode, Acc-no, last-dt-access)
f.k. f.k.
P.K.

P.K. ↓ P.K. of related entity type → Partial key.

Weak entity type: ↓ Totally participated
↓ Primary key of related entity type



Weak entity type
↓
its own attributes - P.K. of the attributes on which it depends (this will be f.k.)



Y is weak entity type

27/2/

Extended ER (EER Diagram)

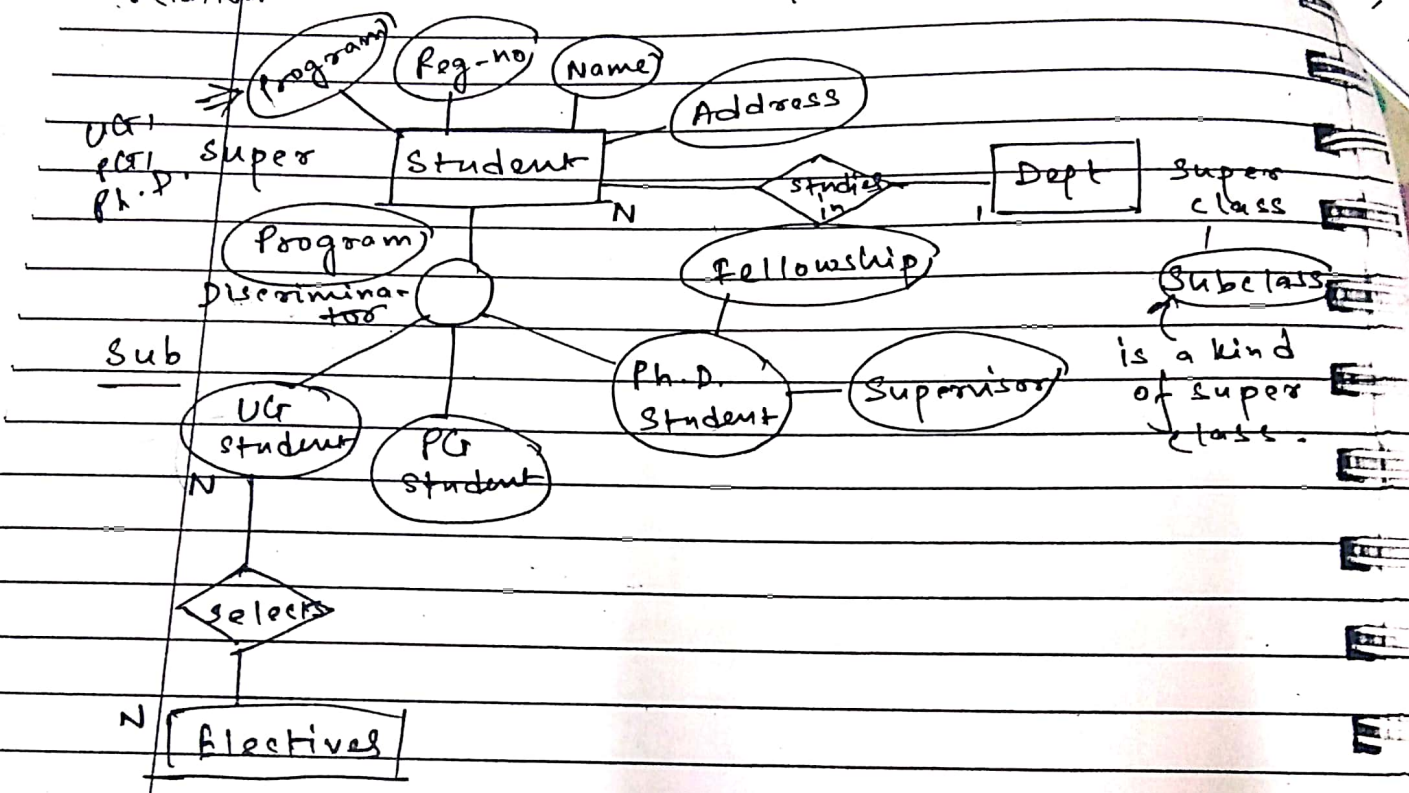
→ Inheritance

Specific attributes of the subtype class

- Certain attributes are applicable to a subset of entity set of a type (not for all).

Specific relation

- In certain association only a subclass takes part.



In a superclass - subclass association, a subclass is related with one superclass.

Specialisation
Generalisation

Top down

Started with a generic class/ Super class. thereafter, finding the special cases.

→ form the subclass.

Bottom up

Started finding super

27/01/2020.

Bottom up. started with subclasses. Later, finding commonalities to form the superclass.

A superclass instances
⇓
To which subclass it belongs.
↙ ↘
Predicate defined (As per certain criteria) ⇓ If it can be defined based on a particular attribute of super class. Attribute-defined.

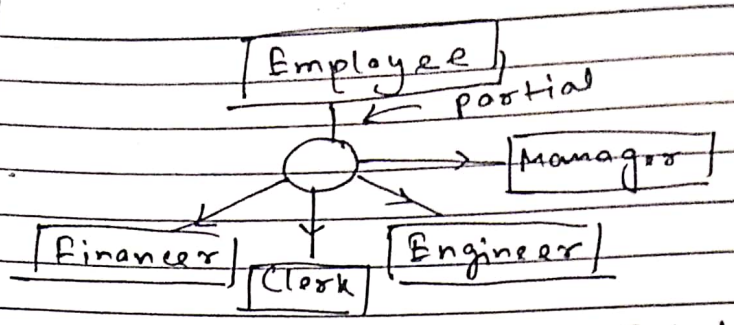
user defined (If not predicate defined, then to be unknown for user)

Constraints to define sub-class:

(A) → Disjointness constraint: If a super class instance can belong to only one subclass, then the association is disjoint.

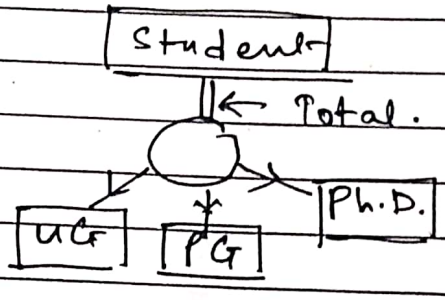
If a super class instance can belong to multiple subclasses ⇒ overlapped.

An employee can be both, an engineer and also a manager.
→ This is overlapped, not disjoint.



(B) → completeness constraint : Association b/w super class and subclass is total if each super class instance belongs to at least one subclass, else partial.

↳ Total / Partial



Mapping superclass - subclass scenario to relation of relational model:

using multiple relations

(A) → using one relation for super class

One relation for each of subclasses

Specific attributes ∪ Primary key of super class

using single relations

(C) → Superclass attributes ∪ all subclass attributes,

Also a type attribute denoting which subclass it belongs.

(PK referencing super class)

Δ also PK here

27/01/2020.

(B) → No relation for super class, only one relation per subclass.
Attributes of super class ∪ specific attribute
If the participation is partial, then the scheme ~~does~~ is not applicable.

If it is overlapped, inconsistency issues will come up as number of copies of same info.

So, this is well suited for Total and Disjoint.

(A) is well suited for all, be it total/partial and disjoint/overlapped.

(D) → super class attributes ∪ all subclass attributes ∪
Total/Partial Type field/fields.
↳ of n bits ↷ ith bit 1 → denotes ith subclass

Total/partial In (C) → disjoint, as only one type can be denoted. Overlapped ✗
↳ type null
In (C) and (D), null values can occur.
If specific attributes are less in number.

In (D), if there are n subclasses, type1, type2, ..., typen.
It handles both disjoint/overlapped.

(A) Super(A1, A2, A3) → PK
FK here → Sub1(A1, B11, B12, B13)
Sub2(A1, B21, B22)

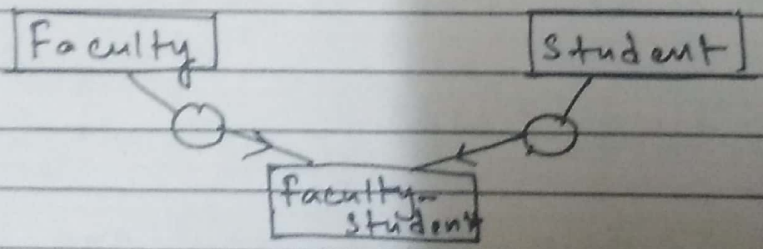
To get all details of instances, we need to perform equijoin operation.

→ To get common info for all (SUPER relation)

(B) Sub1(A1, A2, A3, B11, B12, B13)
Sub2(A1, A2, A3, B21, B22, B23)
→ To get all details of a sub

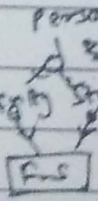
to get all common info for all instances
→ outer union.

MULTIPLE INHERITANCE:

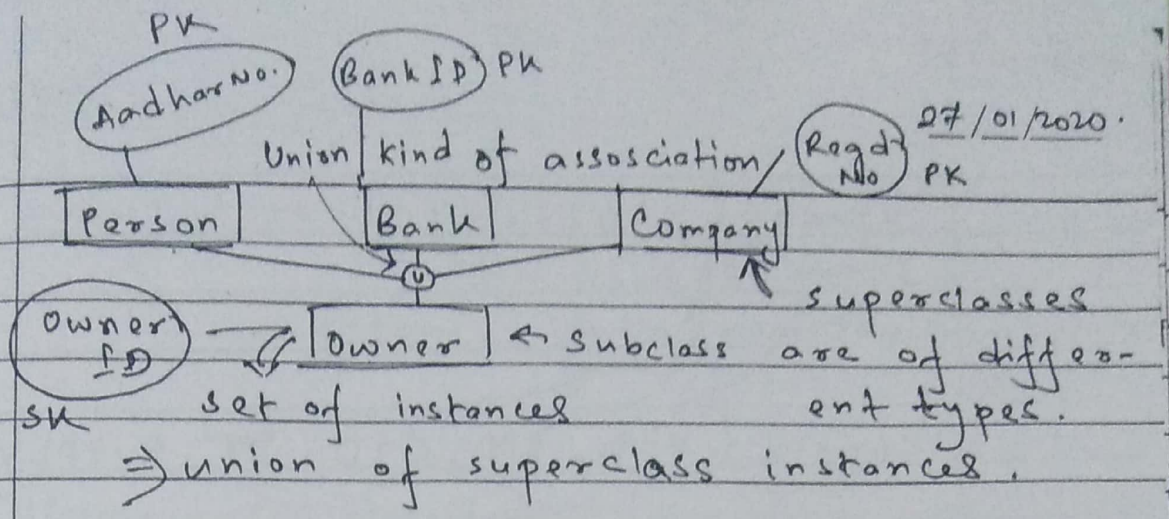


If a class has multiple superclasses, then the subclass is called shared sub-class.

This is super class Lattice, not a superclass hierarchy.



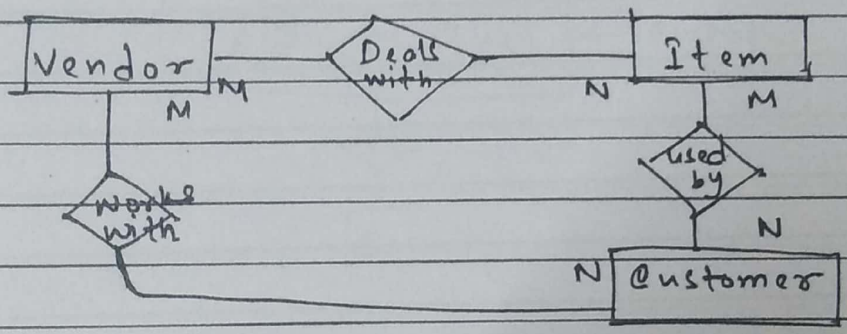
If its super classes have commonalities, then those to be included only once.



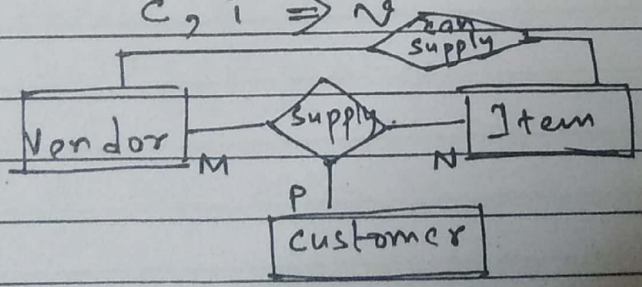
27/01/2020.

⇒ union of superclass instances.

PKs are of different types.
So, in subclass, introduce a new PK which will be known as surrogate key.
copy this surrogate key to superclasses also.



for a customer, for an item, who is vendor?



Ternary relation

for higher degree relation of ER diagram.

⇓

∪ PK of attribute of all participating entity type ∪ additional attribute

↳ FK referencing to corresponding entity type.

27/01/

Vendor (VID, ...)

Item (IID, ...)

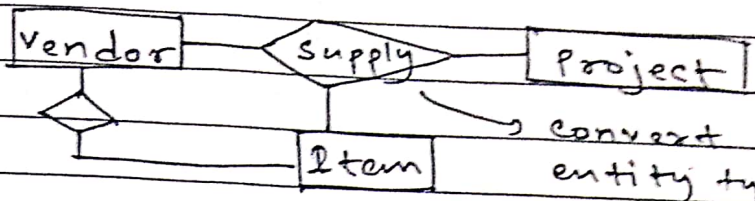
Customer (CID, ...)

supply (VID, IID, CID, DT_SUPPLY)

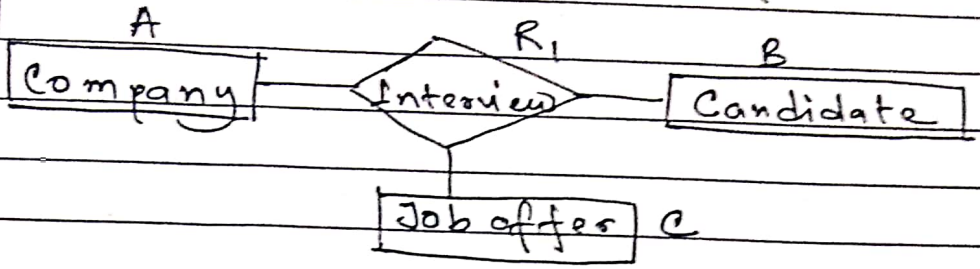
If cardinality for an entity type is 1, then its PK may not be required to form the PK of the relation.

03/02/2020

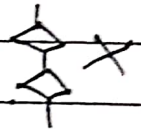
Ternary Relation:



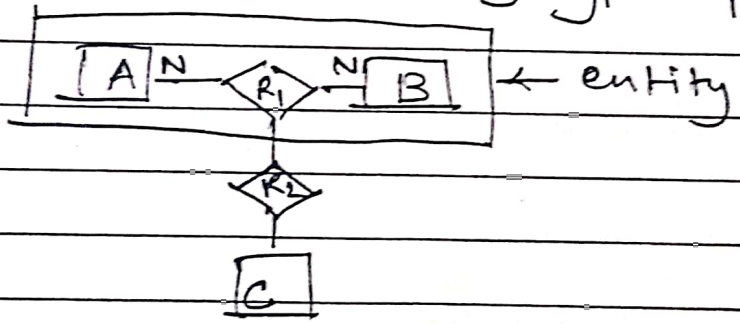
→ convert it to weak entity type which totally participates.



Relation with another relation is not allowed in ER diagram.

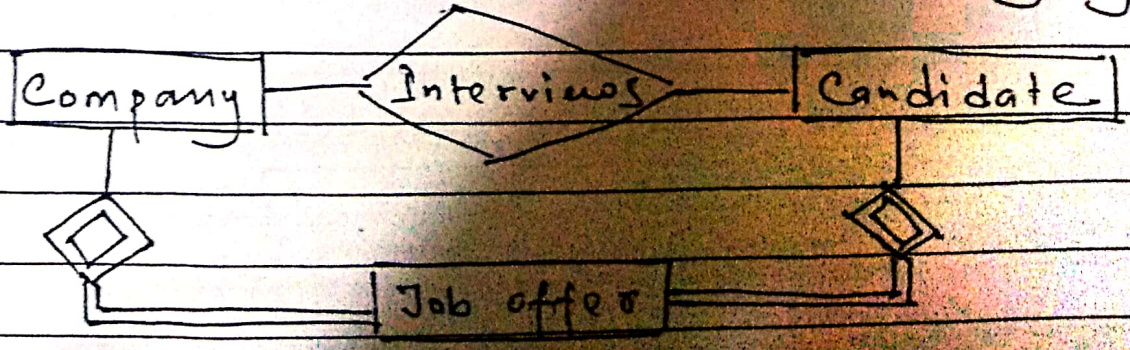


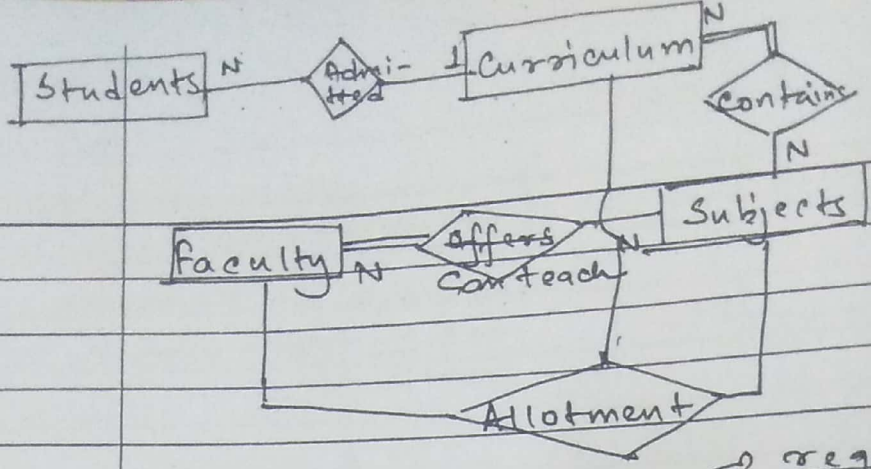
⇒ Relation b/w entity types.



Combining a relation along with related entity type into a composite entity type
↳ Aggregration.

A Job offer is not independent ⇒ weak entity type.





03/02/

→ requirement & appl.

Depending on scope of system, we have to develop the ER diagram first.

① How entity types are related?
② What are the queries?

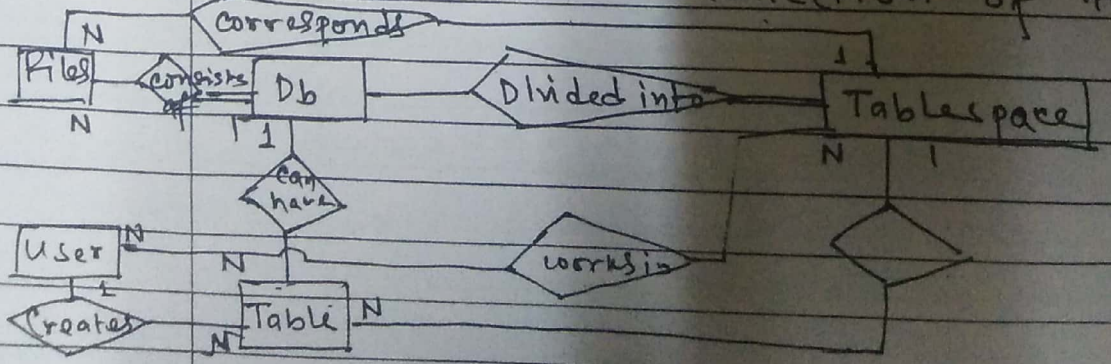
Depending on Relational Algebra, we have a commercial language, SQL. (Structured Query Language). ↔ Non-procedural

High level language is procedural.
Oracle provides a tool called SQLPLUS

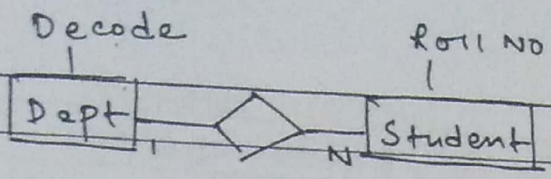
Database
↓
Logically partitioned into number of tablespaces

Oracle has one tablespace of its own, known as system tablespace.

Database is a collection of tables.



03/02/2020



SQL> CREATE TABLE DEPT (DCODE, CHAR(5) PRIMARY KEY, DNAME, CHAR(20)) ; ↵

col name — Type & size — col. level constraints

CHAR(Int)
VARCHAR(Int)
NUMBER(m, n)
m: total no. of digits
n: no. of places after decimal

NOT NULL

DDL

1 table created

SQL>

UNIQUE allows 1 null value

In oracle → CHECK (condition)

SQL> CREATE TABLE STUDENT
(Roll NUMBER(3,0) PRIMARY KEY CONSTRAINT PK_student,
Name CHAR(25),
SCORE NUMBER(5,2) CHECK(SCORE>=0 AND SCORE<=100),
DCODE CHAR(5) REFERENCES DEPT.DCODE) ; ↵

CHECK (SCORE BETWEEN 0 AND 100)

SCORE NUMBER(5,2) DEFAULT 50

ADD DATA-

SQL> INSERT INTO DEPT
VALUES('D1', 'SE') ;
DCODE DNAME

SQL> INSERT INTO STUDENT VALUES(1, 'ABC', 85, 'D1') ;

SQLPLUS> SELECT * FROM CAT ← SQL command
↑ Catalogue

Describe
SQLPLUS> DESC Tablename

(Roll, score, Dcode)
SQL> INSERT INTO STUDENT VALUES(2, 80, 'D1')
↳ Name → null

03/02/

SQL> SELECT * FROM DEPT; ← all columns

SQL> SELECT DNAME FROM DEPT;
Roll, score+5, UPPER(name)

SQL> SELECT * FROM STUDENT WHERE score>
SQL> UPDATE STUDENT
SET Score = Score + 5;
WHERE Score < 50;

SQL> DELETE FROM STUDENT ← Delete all w/o filtering
WHERE _____ ← conditions will be checked and then deletion.

STUDENT(Roll, ...) SUBJECT(scode, ..)
EXAM-ATT(Roll, Scode, ...)
RESULT(Roll, SCODE, SCORE)

SQL> CREATE TABLE EXAM-ATT;
(Roll NUMBER(3,0) REFERENCES STUDENT.Roll,
Scode NUMBER(5) REFERENCES SUBJECT.Scode,
PRIMARY KEY(Roll, SCODE));

SQL> CREATE TABLE RESULT
(Roll NUMBER(3,0), SCODE NUMBER(5),
Score(3,0), PRIMARY KEY(Roll, SCODE),
FOREIGN KEY(Roll, SCODE) REFERENCES
EXAM-ATT(Roll, SCODE));

Class Test-1 (17th February).

SQL

05/02/2020

INSERT INTO STUDENT VALUES(1,'abc','01-Jan', ...)

STUDENT(Roll, Name, Dt-Birth, ...)

Name was not stored ⇒ find our corresponding roll

SUBJECT(SCODE, SNAME, FM, PM)

RESULT(ROLL, SCODE, SCORE)

SELECT ROLL FROM STUDENT WHERE NAME IS NULL;

SELECT * FROM RESULT WHERE SCORE >= 50

LTRIM() ⇒ Removes leading spaces.

RTRIM() ⇒ Removes trailing spaces.

TRIM() ⇒ Removes both.

Pow(m,n) ⇒ m is rounded off upto n places after decimal.

ABS(n)

ROUND(175.967,2) ⇒ 175.37 (175.365, -1) ↳ 180.00

BUILT-IN FUNCTIONS:

- Single Row / Non-aggregate fn. ⇓ Acts on individual tuple attribute and returns a value for that.
- Aggregate fn. ⇓ Acts on a collection and for the collection, it gives a single value.

SUBSTR(string, n, m)

↓

Starting from position returns in char.

1st position ⇒ 1.

SYSDATE

↓

System Date

DUAL → Special table in ora...

SQL> SELECT ~~DATE~~ SYSDAT

SQL> SELECT SYSDAT FROM DUAL;

SQL> SELECT CEIL(174.36) FROM DUAL;

TO-CHAR(n)

TO-NUMBER(A)

Number string

TO-CHAR(Date, 'DD-MM-YYYY')

↓

2-digit month

MON → 3 char month

MONTH → Full month

MONTHS-BETWEEN (dt1, dt2)

DT1 > DT2

= >= <= !=

SQL> SELECT * FROM ~~ALL~~ RESULT WHERE SCORE BETWEEN 50 AND 70;

SQL> SELECT * FROM RESULT WHERE SCORE IN (49, 69, 79, 89);

Equality of the values

SCORE = 49 OR .....

IN	BETWEEN
NOT IN	NOT BETWEEN

SELECT * FROM RESULT WHERE SCORE >= ANY(40, 50, 60, 70);

↑

One can specify the relational operator. If the relation holds for any value in test then it is true.

05/02/2020

ROLL, NAME, SCORE for a particular SCODE.

SQL> SELECT S.ROLL, NAME, SCORE
FROM STUDENT S RESULT R
WHERE STUDENT.ROLL = RESULT.ROLL
S R
AND SCODE = 'S1';

Self-Join

EMP (ECODE, ENAME, BASIC)

SQL> SELECT e1.ENAME, e2.ENAME
FROM EMP e1, EMP e2
WHERE e1.BASIC = e2.BASIC
AND e1.ECODE < e2.ECODE

SIMPLE SUBQUERY — Show the result of all students for SNAME == DBMS.
MATCH CASE → UPPER (SNAME) = 'DBMS'

Bad method	Good method
SELECT ROLL, SCORE FROM RESULT, SUBJECT WHERE RESULT.SCODE = SUBJECT.SCODE AND UPPER(SNAME) = 'DBMS'	SELECT ROLL, SCORE FROM RESULT WHERE SCODE = (SELECT SCODE FROM SUBJECT WHERE (SNAME = 'DBMS') → Inner Query outer Query

Inner query is executed first and its result is used in outer query.

- It should be return exactly one value (otherwise, special measure has to be taken).
- Type of value expected in outer query and that provided by inner query must be compatible.

→ IN, ANY, ALL

If subquery returns multiple values.

10/02/2020

SELECT DEPTNO, GRADE, SUM(BASIC)
FROM EMP.

WHERE clause acts on individual tuple.

GROUP BY DEPTNO, GRADE.

SELECT DEPTNO, SUM(BASIC)
→ FROM EMP
→ WHERE DT-IN >= 01/01/2000.
→ GROUP BY DEPTNO
→ HAVING SUM(BASIC) >= 100000
ORDER BY SUM(BASIC) DESC;

SELECT *
FROM EMP
ORDER BY DEPTNO, BASIC DESC;

SELECT DEPTNO, SUM(BASIC), TOTAL-BASIC
FROM EMP

10/02/2020.

WHERE ----.
GROUP BY DCODE
WHERE TOTAL-BASIC = —
ORDER BY TOTAL-BASIC.

FROM
WHERE
GROUP BY
HAVING
ORDER BY

W/o Group By not possible.

Maximum w/o using MAX function

```
SELECT DISTINCT→BASIC
FROM EMP
WHERE BASIC >= ALL
      (SELECT DISTINCT BASIC
       FROM EMP);
```

DEPT(DCODE, DNAME)
EMP(ECODE, ENAME, ..., BASIC, DCODE, ...)

```
SELECT DNAME, SUM(BASIC)
FROM EMP, DEPT
WHERE EMP.DCODE = DEPT.DCODE
GROUP BY DEPT.DCODE, DNAME.
```

Inner query must return a value and it is used in outer query. Inner query is executed once.

CO-RELATED SUBQUERY

Q. Find out the employees with highest basic in corresponding department.

Ans:

```
SELECT ENAME
FROM EMP A
WHERE BASIC =
   (SELECT MAX(BASIC)
    FROM EMP B
    WHERE B.DCODE = A.DCODE)
```

in case of change of ~~type~~, decrease in size → only if NOT NULL

10/11

As here, outer query (EMP A) is refered in Inner query, so, Inner query is no longer executed once, as with change in outer query Inner query has to be executed again.
↳ Special CASE ~~↳ or with~~

```
SELECT DNAME
FROM   DEPT
WHERE  10 >=
  ( SELECT COUNT(*)
    FROM EMP
    WHERE EMP.DCODE = DEPT.DCODE)
```

Take a tuple in Outer query (candidate Tuple). Evaluate Inner query for this candidate Tuple. Use Inner query result to decide action on candidate tuple. Move to next tuple in outer query and repeat the steps.

```
SELECT DNAME          ← find out department
FROM DEPT               where nobody works.
WHERE NOT EXISTS      To delete, replace
  ( SELECT *            SELECT with DELETE.
    FROM EMP
    WHERE EMP.DCODE = DEPT.DCODE)
```

From ~~FROM~~ an existing table, we want to ~~get~~ create a table and also copy data.

EMP(ECODE, ENAME, DCODE, DT_JOIN, GRADE, BASIC, ...)

ADD (col.....)
MODIFY (modified definition of all columns) 10/02/2020
ADD CONSTRAINTS DROP CONSTRAINTS

OFFICER (ECODE, ENAME, BASIC)

SQL> CREATE TABLE OFFICER (EMP_CODE, EMP_NAME, SALARY)
AS
SELECT ECODE, ENAME, BASIC
FROM EMP
WHERE GRADE='A' ;

To bring only data in already created table.
INSERT INTO OFFICER VALUES
(SELECT ECODE, ENAME, BASIC
FROM EMP
WHERE ____);

DDL → DROP TABLE Tablename ← Delete schema and all
DELETE * FROM Tablename.

VIEW

↓

logical table

When a view is created, its definition is stored into data dictionary.
when it is referred, replaced by the definition

```
CREATE VIEW EMP_CSE
AS
SELECT *
FROM EMP
WHERE DCODE = 'CSE'
WITH CHECK OPTION
```

A view may be based on multiple tables.

Only SELECT ✓
INSERT }
DELETE } X
UPDATE }

If we want to change schema (though schema is static, thing → schema is inflexible)

FILES ⇒ collection of Blocks

collection of records ⇒ collection of related fields.

FILES → Fixed length record
→ variable length record.

make files for each type storing records of different types

⇒ If records are of same types then also size of individual records may vary, as (some fields may be of variable length) and (some attributes are optional).

To make it fixed length
→ some attributes are multi-valued.
for each field consider maximum length.
& Include such fields for all.
Keep provision for maximum number of values for each record.
lead to space wastage.

Variable Length Record
→ We need a record separator (for eg. '$').
→ variable length field ⇒ field terminator (#)
→ Lots of optional attributes <Fieldname, Value>
Fieldname separator. ← Making this mandatory (*) → huge space wastage.
→ Multivalued ⇒ Value Separator(,)

File → Unspanned Organisation (A record cannot span over multiple box) (R<B)
→ Spanned organisation. (can span)

Record size = R bytes, Block size = B.
R > B → only way is spanned organisation.

(bfr) Blocking Factor :- Number of records per block.

10/02/2020

~~For~~ fixed length record and unspanned org.

$bfr = \lfloor B/R \rfloor$

Records of variable length ⇒ Average number of records per block, as number of records per block is varying.

$b = \lceil r/bfr \rceil$

Allocation of Blocks:

- Contiguous (Seek time for each record is not required. So, reading will be faster) Problem → expansion is difficult.
- Linked (A block points to next. Sufficient to know Head pointer. Sequential access. May be slow. But expansion is easier).

Combination of above 2 ⇒ Cluster Allocation

A set of contiguous blocks ↓ cluster

Clustered and Linked.

Indexed Allocation ⇒ In a table, keep the set of block pointers.

For a file ⇒ (unspanned, fixed length), blocks are numbered as Block 0, Block 1, ...

In a block, records are numbered as 0, 1, ...

We want to get to the k-th record.

Offset in the block $= \lfloor k/bfr \rfloor$

$= k \pmod{bfr}$

We can go for ↓

Direct or random access kind of thing.