30/12/2019

Database Management Systems

DATABASE
Centralized collection of interrelated data

MGMT SYSTEMS
A set of programs to provide an environment for convinient and efficient access and storage of data from/to database

Data Structure to store data in secondary storage

FILE
→ File Processing System ⇒ (DATABASE / DBMS ?)
- Redundancy & Inconsistency

Application → Developed by no of programmers
Each team → creates the copy of the data they require
↓
i.e. → Same data is being maintained by no. of teams.

Multiple copies of same data

DATA REDUNDANCY
→ WASTAGE OF SPACE
→ MAY LEAD TO DATA INCONSISTENCY

If all the copies are not updated simultaneously, then there is a mismatch of data

→ **DIFFICULTY IN ACCESSING DATA**

Result :-

(SUB 1 >= 80 AND ___) / OR (___)
60

As per requirement, programs are written to access data.
If requirements is changed, one may have to modify the program / write a new one.

→ **ISOLATION OF DATA**
* Data may spread over number of files
↓
Collect from all files maintaining the correspondence. — DIFFICULT.

→ **CONCURRENT ACCESS ANOMALY**
Same piece of data being accessed by no. of users simultaneously.
↳ If they modify ⇒ INCORRECT RESULTION

→ **SECURITY**
All users must not access / operate on all data

→ **INTEGRITY ENFORCEMENT**
Data is supposed to satisfy certain criteria / constraints.
↳ Once constraints are specified, DBMS takes care to enforce them.

02/01/2020

DATA ABSTRACTION

→ Set of interrelated data files and set of protocols
DBMS → provides DATA abstraction
[Hiding the complexity of representing & accessing data from user.]

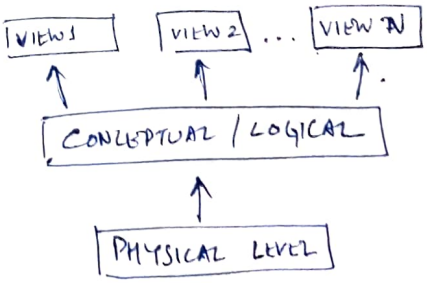
⇒ **PHYSICAL LEVEL**
↳ How the data is physically stored? (COMPLEXITY)

⇒ **CONCEPTUAL LEVEL / LOGICAL**
↳ What data is stored?

Ex.
```
struct student
{
  int roll ⇒ 2
  char name [31] ⇒ 31
  int score [5] ⇒ 10
}
```
43 Byte.

- roll name : int
and 5 marks int

⇒ **VIEW LEVEL**
↳ To a user, whole data may not be visible.
PART → view for the user
MULTIPLE VIEWS



⇒ INDEPENDENCE LEVEL

↳ If one can develop the application so that change in physical defination does not bear any impact on the program, then physical level independence is achieved.

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

↳ Difficult to achive

DATA MODEL

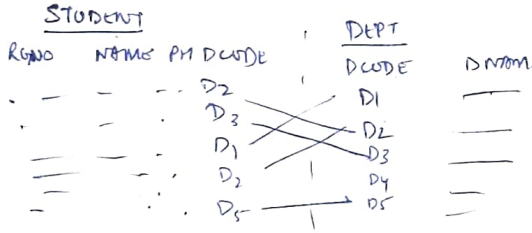
Database is designed/structured based on datamodel

↳ Set of tools to represent data, their inter-section, semantics of data & constraints on data

→ RECORD BASED MODEL

• Relational Model

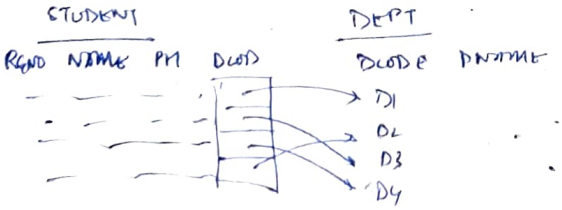
→ To maintain relation among the records of different types, an attribute value (common to both) is stored as a part of record



• NETWORK MODEL

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

→ Complex graphs may be formed (*)



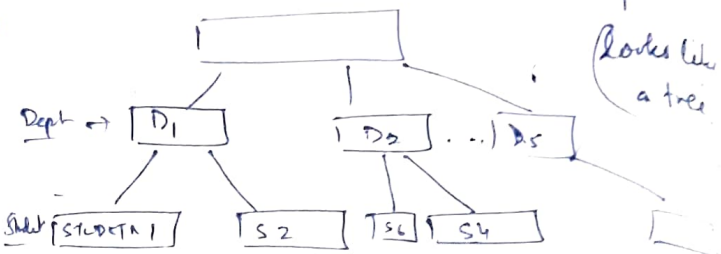
• HIERARCHICAL MODEL

U

Organized form of Network Model.

→ To maintain relation in record ptr. to related record is kept

⇒ Records of different types are organized according to certain hierarchy.



INSTANCE AND SCHEMA OF A DATABASE

⇒ Schema :- Structure of a Database

↳ May change but less frequent

Student Database, record has ROLL (num)
NAME (string)
SCORE (num)

⇒ Instance : Content of the database at a particular point of time

↳ It will change frequently.

LANGUAGE

→ SQL (Structured Query Language)
→ DDL (Data Definition Language)
→ DML (Data Manipulation Language).

DDL → To specify the schema of

DML → to store, retrieve, update/delete data in/from database

→ PROCEDURAL
→ What is required and how to get it? (PROCEDURE)
→ NON-PROCEDURAL
~~→ How to get it? (field data~~
→ What data is required?
[How to get it?] → NOT to be specified

MAJOR FUNCTIONAL UNITS OF ~~DATABASE~~ DBMS

• DataBase Management

→ A software module forming the core parts of DBMS.

Major tasks are :-

→ INTERACTS WITH FILE MANAGER & (PART OF OS) to WORK WITH DATA FILES.
→ INTEGRITY ENFORCEMENT
→ SECURITY
→ BACKUP AND RECOVERY
→ CONCURRENCY CONTROL

- **DDL Compiler**
  - Translates DDL STMT
  - Generates METADATA (Data about Data) which is stored into DATA DICTIONARY (PART OF DATABASE)
    - → NAME OF DATABASE? / WHO IS CREATOR / OWNER? → NAME OF DATABASE AND ASSOCIATED ATTRIBUTES.

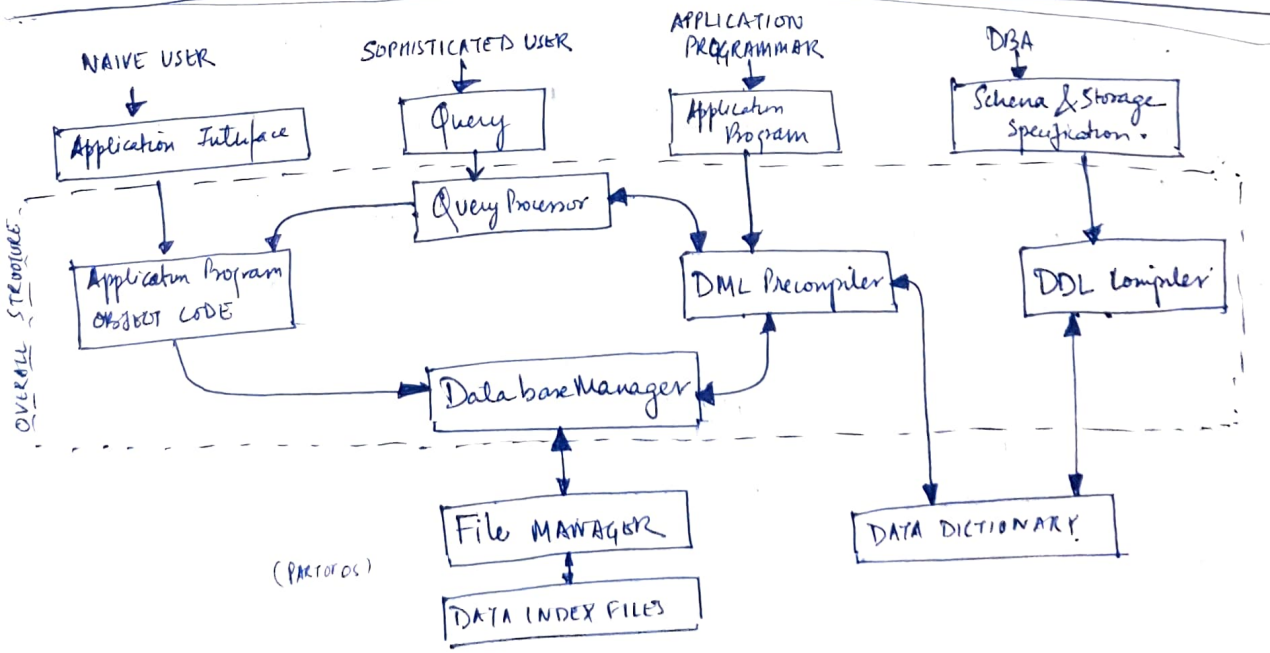
- **DML PRECOMPILER**
  - DML ⇒ Non-Procedural
  - To develop an application, procedural constructs are required
    - ↓ var, constants | loop, decision making
    - DML → helps in interacting with database efficiently.
      - JAVA, BASIC } PROCEDURAL ASPECTS.

- **APPLICATION**
  - → Developed using language that provides procedural support in the program; DML statements are embedded to interact with database → EMBEDDED SQL
    - ↓ Host language.
  - → DML Precompiler converts embedded SQL/DML statements into equivalent host language.

- **QUERY PROCESSOR**
  - → A query submitted by user can be accomplished by a number of ways.
  - → Query processor finds an efficient way to do so and makes detailed execution plan.

USER
- → 1. DBA (Database Administrator)
  - DDL STMT
    - Specifying overall database schema and storage schema
    - Modifying
    - Integrity specific
    - Creating users and granting/revoking access privileges
- → 2. Application Programmer
  - → Develop Application (Host language + Embedded SQL)
- → 3. Sophisticated User (DML Statement
- → 4. NAIVE USERS
  - → Works with application program interface

FIGURE



RELATIONAL MODEL

06/01/2020

DATABASE = Collection of Relations

Informally,
Relation → file consisting of records

Roll	NAME	PH_NO	SCORE
1	ABC	9X123	95
2			
3			

Each column has a name (attribute)

A row → collection of related data → tuple

Domain (data type) ⇒ Domain corresponds to set of atomic values
- atomic → indivisible
- specific to application
- ATOMIC / COMPOSITE

RELATION
- Schema (Intension)
- STATE OF RELATION (Extension)

- A Relation schema is defined as $R(A_1, A_2, \ldots A_n)$ where R is the relation name, $A_1, A_2 \ldots A_n$ are the attributes.

→ Each attribute A_i has a domain D_i

→ Attribute is the name of role played by its domain in the relation

ITEM

ICODE	CHAR[5]

EMP

ECODE	CHAR[5]

→ Relation Name, R_1 & attributes help to interpret Data.

→ A Relation (or state of a relation), $R(A_1, A_2 \ldots A_n)$ is denoted as $r(R)$

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

↳ t_i is an ordered collection of n values $\langle v_1, v_2 \ldots v_n \rangle$

where v_i is the value for Attribute A_i, coming from DOM (domain) or it is NULL (absence of any value)

→ Degree of a relation ⇒ No of attributes in the schema.

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

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

Roll — Int (3)
Score — Int (2)

Roll: 0, 1, ..., 999
Score: 0, ..., 10

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

Characteristics of a relation

- Tuples are unordered

Relational model does not rely on any assumption regarding the ordering of tuples.

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

$R(A_1, A_2 \ldots A_n)$

$t_i = \langle v_1, v_2 \ldots v_n \rangle \mid v_j = t_i[A_j] \; v_i \in DOM(A_i)$

- $\langle v_1, v_2 \ldots v_n \rangle$: ordered collection of values.
- $t_i[A_j]$: attribute

- State of Relation $R(A_1, A_2 \ldots A_n)$ is a collection of mappings $\{t_1, \ldots t_k\}$.

→ t_i is a mapping, maps from R to D

D: Union of Domain $A_1, A_2 \ldots$

R → D

t_i is a collection of attribute value pair

$\langle (ROLL, 1), (NAME, "ABC"), (SCORE, 80) \rangle \ldots$

- Value of an attribute and NULL

→ Attribute value ∈ domain of corresponding attribute or it may be null

Value of an attribute must be atomic & it must be single valued
↳ for a tuple, one value

Relational model ensure 1NF (1st Normal form)

NULL →
- Not Applicable
- Value is not known
- Intentionally not stored

- INTERPRETATION

Relation can be taken as a type declaration or assertion

STUDENT (ROLL, NAME, SCORE) ⇒ 1 "ABC" 85 — fact of an instance

CONTRAINTS

MODEL LEVEL CONTRAINTS

(CONSTRAINTS which are inherent in the ~~atom~~ model itself)

→ tuples are unordered

→ Values in a tuple → may or may not be ordered

→ Attributes are atomic & single valued

[no two tuples are same]

- SCHEMA LEVEL CONSTRAINTS

~~Cons~~ constraints that can be specified along with schema defination using DDT STMT

APPLICATION BASED CONSTRAINTS

- Constraints that involve application logic and can not be handled by model & schema based constraints will require special mechanism ⇒ Trigger

FUNCTIONAL DEPENDENCY, MULTIVALUED DEPENDENCY } ⇒ CONSTRAINTS TO BE HANDLED AT DESIGN TIME (NORMALIZATION)

$X \rightarrow Y$ if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

- DOMAIN CONSTRAINTS
- PRIMARY KEY CONSTRAINTS.

$R(A_1, A_2 \ldots A_n)$

Superkey (SK) ⇒ Subset of $R(A_1 A_2 \ldots A_n)$ such that for any two tuples $t_1, t_2 \in r(R)$

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

Regno →	Roll	Name	Phone	Score
↓				
Candidate Key				

Candidate Key :- Candidate Key is a super key such that no proper subset of candidate key is a super key.

→ Multiple candidate keys may be there
↳ One of the candidate key is(s), designer sets as primary key.
(Minimal Superkey) ↳ Used to identify a tuple

→ Smaller Size preferred
→ Alphabetical is less preferred

* In the context of application, candidate key by which instances are identified in general should be

- ENTITY INTEGRITY CONSTRAINT

As primary key is used to identify the instance, it must not be NULL

Relation State ⇒ VALID STATE ⇒ A tuple must satisfy the constraint

⇒ INVALID STATE ⇒ Database ↓ Collection of Relation

Database Schema

- Set of schema of all individual Relation, $R = \{R_1, R_2 \ldots R_n\}$ & a set of constraints i.e.. State of the database $\{r(R_1), r_2(R_2), \ldots r_n(R_n)\}$

RELATIONAL MODEL

07/01/2020

Relation — Schema, State.

MULTIPLE DEFINATION

- Characteristics
- Constraints
  Database.

Referential Integrity and Foreign key

Referenced ~~Referenced~~ Relation	R_1	DEPT (DCODE, DNAME, ...)	P.K.
Referencing Relation	R_2	EMP (ECODE, ENAME, BASIC, DCODE)	F.K.

Say R_1 be the referenced relation and R_2 be the referencing relation.

Let Primary key be the primary key of R_1, Foreign key be the subset of attributes in R_2 such that :-

(i) domain of P.K. in R_1 & domain of F.K. ~~be the subset of attributes in R_2 such that~~ in R_2 are same.

(ii) for any tuple $t_2 \in r(R_2)$ either $t_2[FK]$ is null or there exists a tuple $t_1 \in r(R_1)$ such that :-

$t_1[PK] = t_2[FK]$

then in t_2, FK is the foreign key and it references PK of R_1

[Referential Integrity]

IMPACT OF FOREIGN KEY IN DML OPERATION

	REFERENCING RELATION	REFERENCED RELATION
Add Record INSERT	Allowed if the foreign key value being inserted is present in corresponding attribute of referenced relation otherwise not allowed	No impact of foreign key [P.K. is unique]
MODIFY	If FK is changed and new value is not there in referenced relation then not allowed	If P.K. is changed and old value is used in referencing relation, then not allowed.
DELETE	NO PROBLEM	If corresponding PK value is used in referencing relation, then not allowed

Model → Must provide language to interact with the relation.

RELATIONAL MODEL

- Relational Algebra (Procedure)
  - Set of operation
  - Relational Algebra expression
  - operations to be applied in order to retrieve data
- Relational Calculus (Non-Procedural)
  - → No operation
  - → What data we need
  - $\{t_{ENAME} \mid Emp(t) \text{ and } t.BASIC \geq 50000\}$

RELATIONAL ALGEBRA

A relational algebra expression consists of a sequence of operation applied on one or more relation & as output it provides another relation.

→ It provides formal framework for data manipulation.
→ Used as internal representation for query optimization and execution
→ Commercial SQL statements have lots of similarities with relational algebra.

SELECT OPERATION

$\sigma_{PREDICATE}(R)$

→ Tuples satisfying the predicate will form the state of output relation.
→ Schema of output relation is same as schema of input relation.

EMP (ECODE, ENAME, DCODE, BASIC)

$\sigma_{BASIC \geq 50000}(EMP)$

$\sigma_{BASIC \geq 50000}(\sigma_{DCODE = D_1}(EMP))$ } same operation commutative
$\sigma_{DCODE = D1}(\sigma_{BASIC \geq 50000}(EMP))$ }

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

PROJECT OPERATION

$\Pi_{attribute\ list}(R)$

$\Pi_{ENAME}(EMP)$ $\Pi_{ENAME, BASIC}(EMP)$

$\Pi_{list1}(\Pi_{list2}(R))$ } $list1 \subseteq list2$
$\Pi_{list2}(\Pi_{list1}(R))$ × } Not commutative.

The projection list does not contain any candidate key.

$\Pi_{BASIC}(EMP)$
↳ Duplicate eliminating projection.

no. of tuples (o/p) < no. of tuples (i/p)

RENAME OPERATION

$R_1(A_1, A_2 \ldots, A_n)$

$\rho_{R_2}(R_1)$ { only renaming relation name

$\rho_{(B_1, B_2 \ldots B_n)}(R_1)$ [only attribute names]

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

↓

renaming in both relation name and attribute name.

$T(B_1, B_2) \leftarrow \Pi_{A_1, A_3}(R_1)$ [Assignment]

CARTESIAN PRODUCT

$R_1 \rightarrow$ deg. n_1 → No of tuples N_1
$R_2 \rightarrow$ deg. n_2 → No of tuples N_2

$R_1 \times R_2$ No of tuples $N_1 * N_2$

o/p relation : Schema $n_1 + n_2$ attribute
deg → sum of degs of participating relations.

DEPT

DCODE	DNAME
D_1	A
D_2	B

EMP

ECODE	ENAME	DCODE
E_1	X	D_1
E_2	Y	D_1
E_3	Z	D_2
	W	

DEPT × EMP

DCODE	DNAME	ECODE	ENAME	DCODE
D_1	A	E_1	X	D_1
D_1	A	E_2	Y	D_1
D_1	A	E_3	Z	D_2
D_2	B	E_1	X	D_1
D_2	B	E_2	Y	D_1
D_2	B	E_3	Z	D_2

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

Example

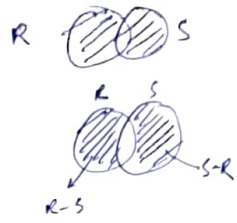
XZ
YW

$\sigma_{EMP.BASIC * EMP1.BASIC \ \wedge \ EMP.ECODE < EMP1.ECODE}(EMP \times \rho_{EMP1}(EMP))$

TEMP
$\sigma_{-}(T \times EMP)$

SET DIFFERENCE (R-S)

o/p relation → schema & degree same as R/S
→ tuples from R which are not in S will be present in o/p relation



S-R ≠ R-S → Not commutative

INTERSECTION

→ must be union compatible

$R \cap S = S \cap R$ → commutative
$(R \cap S) \cap W = R \cap (S \cap W)$ → associative

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

↳ can be produced from basic operations hence, not a basic operation.

13-01-2020

Basic Operations :-

Select (σ)
Project (π)
Cartesian Product ($\times$)
Rename (ρ)
Union ($\cup$)
Set Differences ($-$)

Not Basic :-
Intersection.

SET OPERATIONS

$R(A_1, A_2 \ldots A_n)$ AND $S(B_1, B_2 \ldots B_n)$ ARE TWO RELATIONS.
SET OPERATIONS ON THEM CAN BE DONE IF THEY ARE

UNION COMPATIBLE

* $DOM(A_i) = DOM(B_i)$ for $1 \le i \le n$

UNION R ∪ S

⇒ o/p relation will be of degree same as that of R & S.
→ state includes all the tuples appearing in R or S or both
→ tuples present in both will appear only once.

$R \cup S = S \cup R$
$(R \cup S) \cup W = R \cup (S \cup W)$

} Union is commutative and associative.

EMPLOYEE (ENAME, AGE)
STUDENT (NAME, AGE)

Example UNION

MATCH (TEAM1, TEAM2, DT_OF_MATCH)

Q Find all dates of all matches of India.

$\pi_{DT\_OF\_MATCH}(\sigma_{TEAM1 = "IND" \ OR \ TEAM2 = "IND"}(MATCH))$

Q. Also find the opponents they play.

$T_1 \leftarrow \pi_{TEAM2, DT\_OF\_MATCH}(\sigma_{TEAM1 = "IND"}(MATCH))$

$T_2 \leftarrow \pi_{TEAM1, DT\_OF\_MATCH}(\sigma_{TEAM2 = "IND"}(MATCH))$

Ans $T_1 \cup T_2$

JOIN OPERATION

$R \bowtie_{predicate} S$

→ θ - join
→ Equi Join
→ If predicate is based on the equality of attribute value
→ NATURAL JOIN
* Same as equi join but common attributes (taking part in equi join) will appear only once in output scheme.)

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

Example.

DEPT (DCODE, DNAME)
EMP (ECODE, ENAME, BASIC, DCODE) or DEPTCODE

EMP ⋈ DEPT
EMP.DCODE = DEPT.DCODE → if attribute names are same
or EMP.DCODE = DEPT.CODE → if " " are different

Schema

ECODE ENAME BASIC DCODE DCODE DNAME
→ multiple entry of DCODE

Solution

$\pi_{ECODE, ENAME, DCODE, DNAME}(EMP \bowtie_{Predicate} DEPT)$

Other form :→ Natural Join

Same as equi join, but common attributes (taking part in equi-join) will appear only once in o/p scheme.

ECODE ENAME BASIC DCODE DNAME

↳ EQUALITY OF ALL COMMON ATTRIBUTES PAIR
ATTR NAME
If different name → we can go for rename.

Ex.

If EMP (ECODE, ENAME, BASIC, DEPT CODE)

$\pi_{EMP(ECODE, ENAME, BASIC, DCODE)}(EMP) * DEPT$

$R \bowtie S$

→ If no predicate is mentioned.
It will be considered for all the attribute name that are same.

Outer Join ⇒ $R \bowtie S$ or $R \bowtie$ or $\bowtie S$

Say Equijoin on R & S.

R(, X) , S(, X)

{1,2,3,4,5} common attribute {1,2,3,6}

$R \bowtie_{R.X = S.X} S$ → tuples of R without match in S will not appear in join and same for $S \bowtie R$.

Left Outer Join ⇒ $R \bowtie S$

→ all tuples of R will appear in o/p, if it has a match in S then ___ info from S will be part of o/p tuple.

Otherwise, it will be matched with a dummy tuple of S

Right Outer Join ⇒ $R \bowtie S$

→ tuples in S with no match will be matched to dummy entry of R.

Full Outer Join → $R \bowtie S$

Outer Union :-

for R U S ⇒ [Union Compatible]

Scheme of R = X U Y
Scheme of S = X U Z
→ not union compatible

Ex.

R STUDENT (ROLL_NO, NAME, DEPT, PROGRAM

S FACULTY (SUB_SEL_NO, NAME, DEPT, DEGREE

R outer union S

o/p - schema : X U Y U Z

For a tuple in R if there is a match in S

i.e. [$t_R(x) = t_S(x)$]

They result in a single o/p tuple

For a tuple in R, without a match in S.

o/p is padded with dummy tuples of S and vice versa.

DIVISION

R STUDENT (ROLL, SKILL_NO)
SKILL (SKILL_NO, SKILL_NAME)
S SKILLS (SKILL_NO)

R(X, Y) X, Y are subset of attributes
S(Y)

$R \div S$ ~~condition :- schema of S~~

condition :-
schema of S ⊂ schema of R

$R \div S$
⇒ output schema will be X.
→ i.e. (schema of R − schema of S)

$T_1 \leftarrow \Pi_X(R)$ — all roll. no.

$T_2 \leftarrow (T_1 \times S) - R$ — [those roll skill) not present].

$T_3 \leftarrow \Pi_X(T_2)$ → roll no of students who do not have at least one skill

$T_4 \leftarrow T_1 - T_3$ → desired output → students will all skills.

↳ Q. Show all roll numbers of students to have all the skills.

In the output, will get a set of tuples $\{t\}$ such that, for t, there are tuples t_r in R with:

$t = t_r[X]$ and for every tuple t_s in S

$t_r[Y] = t_s[Y]$

SUBJECT (SCODE, SNAME, FM, PM)

RESULT (ROLL, SCODE, SCORE)

$\Pi_{ROLL, RESULT.SCODE}$ (RESULT ⋈ SUBJECT RESULT.SCODE = SUBJECT.SCODE AND SCORE ≥ PM)

↓

R (ROLL, SCODE)

S (SCODE) ← Π_{CODE} (SUBJECT)

$R \div S$

↓

*?

Operation in sequence = Procedural (Relation operator)

AGGREGATE FUNCTIONS

$\mathcal{F}$ → scripted F

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

↓

T (Total Basic)

SUM (BASIC)

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

$\mathcal{F}_{SUM_{BASIC}, AVG_{BASIC}}(EMP)$

$\mathcal{F}_{fn1\ attr, fn2\ attr}(Relation)$

T (DCODE, S_BASIC)

$_{DCODE}\mathcal{F}_{SUM_{BASIC}}(EMP)$

↓ Grouping on dcode → Aggregates fn, if works on a group.

For a group, one tuple is return.

$\sigma_{S\_BASIC > 1LAKH}(T)$

EMP (ECODE, ENAME, BASIC, DESIG, DCODE)

$_{DCODE\ DESIG}\mathcal{F}_{AVG_{BASIC}}(EMP)$

15-01-2020

Relational Calculus

→ does not have a set of relation

In relational calculus expression in a relational query, we ~~stop~~ specify what is required unlike relational algebra expression, no operation is specified.

In comparision to relational algebra, relational calculus is ~~not pr~~ 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 completely if it can relationally express every query that can be specified by relational calculus.

A relational calculus expression is of the form (tuple relational calculus)

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

- ↓ tuple variable
- set of tuples satisfying the condition

→ tuple variable ranges over the tuple of a relation

usually $\{t \mid emp(t)\}$

- ↓ set of tuple from relation EMP.
- ↓ range relation

Let the scheme be :-

DEPT (DLCODE, DNAME)

EMP (ELCODE, ENAME, DLCODE, BASIC)

→ can use relational operation.

$\{t \mid EMP(t) \text{ AND } t.BASIC \geq 5000\}$

t.attr or t[attr] {if t belongs to the any of above two}

If only name is required —

$\{t.NAME \mid EMP(t) \text{ AND } t.BASIC \geq 5000\}$

↓ so, we can take the projection also.

$\{e.name, d.DNAME \mid EMP(e) \text{ AND } e.DLCODE = d.DLCODE\}$

↓ as they are appearing left of the

we say it free variable

as d is not coming in o/p, we can write it to the left

[the condition]

Bounded Variable

↓ accompanied by quantifier

- Existential
- Universal → for all ($\forall x$)

$(\exists (x))(R(x) \text{ AND } \ldots)$

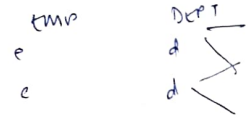
↓ x ranges over R and satisfies the

↓ Range relation ⇒ universal relation

There exists a tuple x such that

$\{e.NAME \mid EMP(A) \text{ and } (\exists d)(DEPT(d) \text{ AND } d.DLCODE = e.DLCODE \text{ and } d.DNAME = 'ABC')\}$

so there exist a D.



EQUIVALENCY

① $(\exists (x))(P(x)) \equiv \text{NOT } (\forall (x))(\text{NOT } P(x))$

② $(\forall (x))(P(x)) \equiv \text{NOT } (\exists (x))(\text{NOT } P(x))$

③ $(\exists (x))(P(x) \text{ AND } Q(x))$

↓ OR

Implied P ⇒ implies Q.

$P \Rightarrow Q$

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

$\text{NOT } (\exists (x))(P(x)) \Rightarrow \text{NOT } (\forall (x))(P(x))$

$(\forall x)(\text{NOT } P(x))$ $\qquad$ $(\exists (x))(\text{NOT } P(x))$

SAFE EXPRESSION

A relational calculus must result into finite no. of tuples then it is safe.

$\{t \mid NOT\ EMP(t)\}$ → not safe

$\{t \mid EMP(t)\}$ → safe.

EX		Variable
SUBJECT (SCODE, SNAME)	—	su
STUDENT (ROLL, NAME)	—	st
ATTENDANCE (ROLL, SCORE)	—	a

Q→ The name of the students who have appeared in all subjects.

Ans

$\{st.NAME \mid STUDENT(st)\ AND\ (\forall(su))(\exists(a))\ (NOT\ SUBJECT(su)\ OR\ (ATTENDANCE(a)\ AND\ a.ROLL = st.ROLL\ AND\ a.SCODE = su.SCODE)\}$.

→ important for using universal quantifier

if su wasn't belonging to SUBJECT, we forcefully relate this by ∀

; can be written here also.

$(\forall(su))(cond\ |\ -\ \ .$

it will satisfies all condition for the database in universe.

20/01/2020

ENTITY-RELATIONSHIP MODEL / DIAGRAM

↓

- → used to capture the data requirement of an application. ~~reflects~~
- → Reflects the requirements at conceptual level
- → Shows the different type of entities involved and association / relation b/w them
- → Reflects their attributes and constraints.

ENTITY :- is an element that exists physically or it may be ~~associated~~ abstract concept.

→ One entity is distinguishable from another.

RELATIONSHIP :- Association between entities.

→ Collection of similar type of entities ⇒ ENTITY SET

→ Entity type denotes the structure of an entity set.

↳ defines / describes → the ENTITY SET.

→ Entity set is described in terms of attributes

Relational Model	E-R Diagram
⇓	⇓
everything is relation	entity & relations

STUDENT ⇒ ROLL, NAME, SCORE

1	ABC	90
2	XYZ	86

→ An attributes maps an entity set to its domain

AN ATTRIBUTE

⇓

→ simple or ~~atomic~~ composite
(atomic)

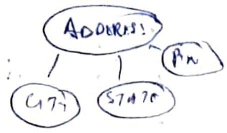
composite ⇓ collection of number of simple attributes.

→ Single valued or multivalued.

Single valued ↓ an instance (tuple) can have one value

multivalued ↓ an instance can have multiple instance

Add. ↓ city, state, pin



NOTE *

> ENTITY TYPE → Schema of a relation (intension of a relation)
> ENTITY SET → state (extension of a relation)

201

→ stored | derived

stored ↓ ~~only~~ only to have the attribute value by storing it

derived ↳ if the value can be deduced from other stored attribute

↓ inconsistency is an issue

{ needs to updated, ~~...~~ → due to change → due to time etc.

→ still used, when frequently the results are asked.

Ex. D.O.B and AGE

→ attributes can have NULL value

→ complex / nested attribute

↓

combination of composite & multivalued attribute.

{ {CONTACT_DETAILS { HOUSADD, {CONTACTNUM }}}
(Premise, STREET, CITY, PIN)

→ key

??

AN attribute → domain

A —HAS→ a value set, V

↓ a set of values that come from a domain

⇓ attribute can take one or more values from this.

$V = \{v_1, v_2 \ldots v_n\}$

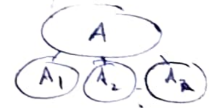
$A : E \rightarrow P(V)$

An attribute is a function for mapping ~~an attribute~~ of entity type to its power set.

Ex. Composite attribute consisting of $A_1, A_2 \ldots A_n$,

Value set ⇒ V_1, V_2, ..., V_n

$P(A) = P(V_1) \times P(V_2) \quad P(n)$



RELATIONSHIP TYPE DEFINES THE ASSOCIATION BETWEEN THE ENTITIES OF DIFFERENT TYPE

NOTE *

Relational Model

↓

degree of relation
(no. of attribute in the schema)

→ RELATIONSHIP TYPE DEFINES :-
THE ASSOCIATION BETWEEN THE ENTITIES OF DIFFERENT TYPES

↳ DEGREE → NO. OF ENTITY TYPES INVOLVED IN RELATION

$E_1, E_2, E_3 \ldots E_n$

Set of similar type association ⇒ Relationship set

Student
1
Course
C_1
C_2
C_3

E_1
Subset $\{(e_1, e_2 \ldots e_n) | e_i \in E_i\}$
$(1, c_1)$
$(2, c_2)$
$(1, c_3)$

Degree 2 ⇒ BINARY RELATION.

→ constraints on binary relation

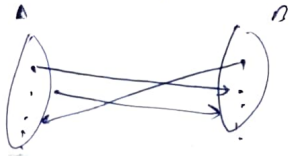
→ Cardinality / Mapping Constraints
→ Participation Constraint (Maximal Cardinality constraint) → Partial, Total

Mapping Constraints

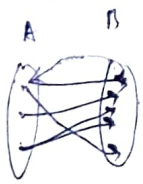
→ AN ELEMENT T ON ONE ENTITY type can have association with how many elements of related entity type.

From A to B.

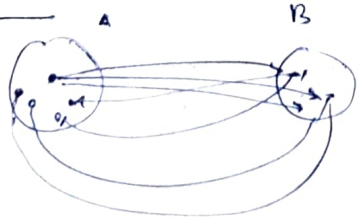
ONE to ONE :→ One element of set A can have link with almost one of set B & vice versa.



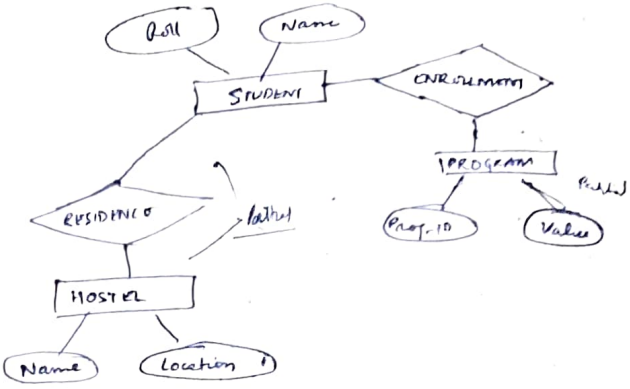
ONE TO MANY :→ From set A one element can have relation with no. of element in B,
But an element of B can have relation with almost one in B.



MANY TO MANY



MANY TO ONE (FROM B TO A)



→ PARTICIPATION CONSTRAINT → for an entity type whether taking part in an association is mandatory or not.

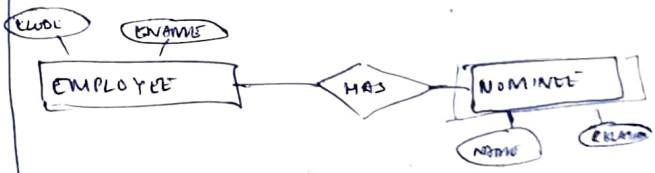
Partial
Total

~~Moran entity~~

→ An entity type participates totally
→ every element must have associations with elements of that type.

Total participation is also referred as :- ~~EXISTENCE~~
→ EXISTENTIAL DEPENDENCIES
⇓
at the very onset of the instance creation it must get associated.

Weak Entity Type
⇓
ENTITY TYPE WITHOUT ANY KEY OF ITS OWN



↓
weak entity type must totally participate in the relation with the entity type on which its existence depends.

To identify an element in weak entity set
⇓
→ find the key of the entity on which it depends.
→ Traversing the relation (in which weak type totally participates), one can get set of related weak elements.
→ To identify an element in the retrieved set, weak entity type has an attribute (s) that can act as key (discriminator).

22-09-2020

ER DIAGRAM TO RELATION: DESIGN
RELATIONAL MODEL

ER DIAGRAM: Entity types and association among them

- consider a relation (acc. to relational model) for each entity type — (strong)

Key of entity type will be primary key

→ Schema will have all the attribute of entity type.

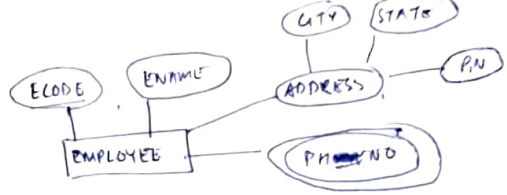
Replace, composite attribute by its constituent simple attribute.

(exclude multivalued attribute)

→ For each multivalued attribute, consider a seperate relation

Copy the Primary key of corresponding entity type and it will be foreign key here referring to the relation for entity type.

Ex1



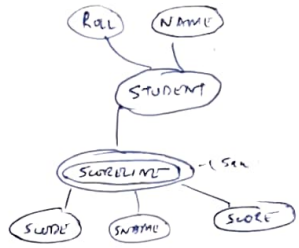
EMPLOYEE (ECODE, ENAME, CITY, STATE, PIN)

EMPHONE (ECODE, PH_NO)

F.K.

P.K

Ex2



ROLL:

NAME

SCODE	SNAME	SCORE

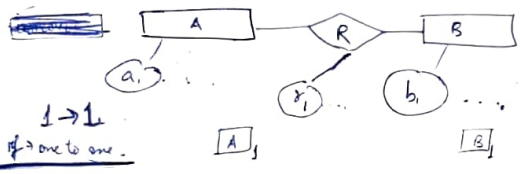
(Same)

(ROLL, SCODE, SNAME, SCORE)

Common P.K

Ex.

MAPPING OF ER-DIAGRAM TO RELATION OF RELATIONAL MODEL



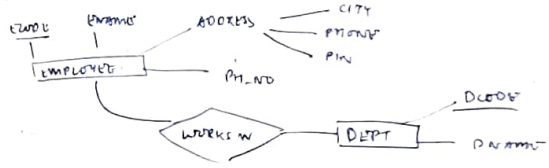
1 → 1

If → one to one.

A

B

Ex3



a) FOREIGN KEY BASED APPROACH

→ In one side, keep the primary key of one entity as foreign key

A (a_1, a_2, a_3) → A (a_1, a_2, a_3, b_1)

B (b_1, b_2, b_3)

→ And also copy the attribute of relation (if any)

A $(a_1, a_2, a_3, b_1, r_1)$

Other way

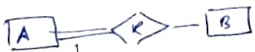
A (a_1, a_2, a_3)

B (b_1, b_2, b_3) → B $(b_1, b_2, b_3, a_1, r_1)$

→ Note:- (choosing which way)

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

Ex.



↓ then chose A

Total participation.

→ Relations can also have attributes

Ex

ER.

CUSTOMER (CC_CODE, AC_NO, DT_LAST_ACCESS)

(one customer is related to one account)

→ attribute of relation on side of entity whose relation is mentioned).

ACCOUNT (AC_NO ... ER.)

c) CROSS REFERENCING RELATION

→ Consider a seperate schema for the relation of ER Diagram.

Attributes:- Primary key of related entity types and attribute of relation itself (if any)

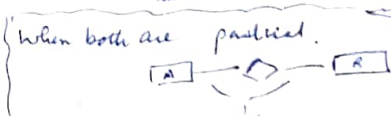
Example

OWNERSHIP (CCODE, AC_NO, DT_LAST_ACCESS)

F.K FK

Note

When both are partial.



(b) MERGED RELATION

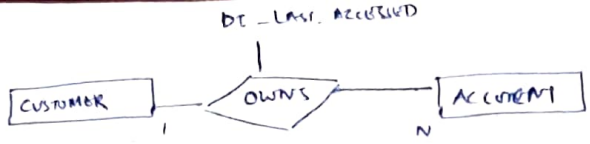
→ Combine the entity types & relation into a single schema of relational model

AnB $\{a_1, a_2, a_3, r_1, b_1, b_2, b_3\}$

CCODE ACC_NO

Case 1. If A totally participates, then for some instances of B, as A exists

⇒ IF BOTH THE ENTITY TYPES ARE TOTALLY PARTICIPATING → otherwise we have to add lot of NULL values



for one to many

CUSTOMER (CCODE , CNAME , CADDR)

ACCOUNT (ACC_NO , BALANCE)

a) FOREIGN KEY APPROACH

* for foreign key approach, attribute ACC_NO becomes multivalued, if put in CUSTOMER, hence we take this on many side

IN MANY SIDE , COPY Primary key of Related , Entity type (it will be F.K. & also keep attribute of relation) (if any).

b) CROSS REFERENCING RELATION

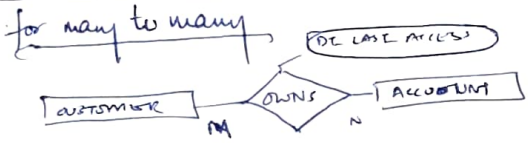
OWNERSHIP (CCODE , ACC_NO , LAST_ACCESS_DATE)

P.K

(Primary key of entity type in many side)

---

for many to many



* no option, but a separate schema (extra consideration)

OWNERSHIP (CCODE , ACC_NO , LAST_ACCESS)

FK FK

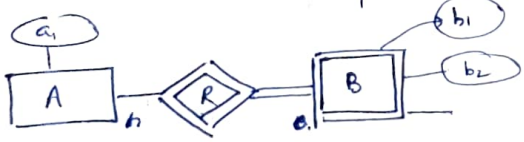
P.K.

WEAK ENTITY TYPE

⇓

Primary key of related entity type totally participates

→ P.K ⇒ P.K. of related entity type + Partial Key



Weak Entity Type

→ its own attributes + P.K. of the attribute on which it depends (this will be F.K.)

A (a_1 , a_2 . . .)

B (a_1 , b_1 , b_2 . . .)

⇓ F.K

GoTo

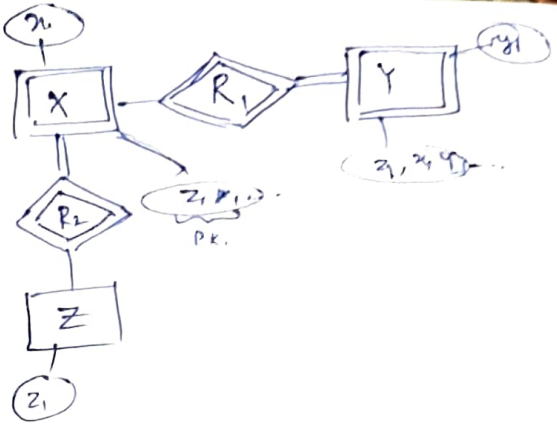
SEQUENCE

* → if [A]—[B] , then a_1 is okay (PK)

but one to many

[A]$_n$ - [B]$_1$, then a_1, b_1 is required

EX



NOTE .

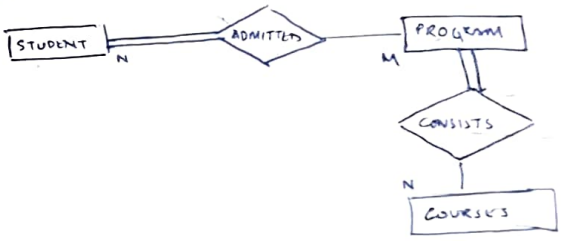
one to many or many-to-many, but one to one

→ P.K. is P.K of related entity type + its own partial key.

else

P.K of related entity type (one to one)

EX .



27-01-2020

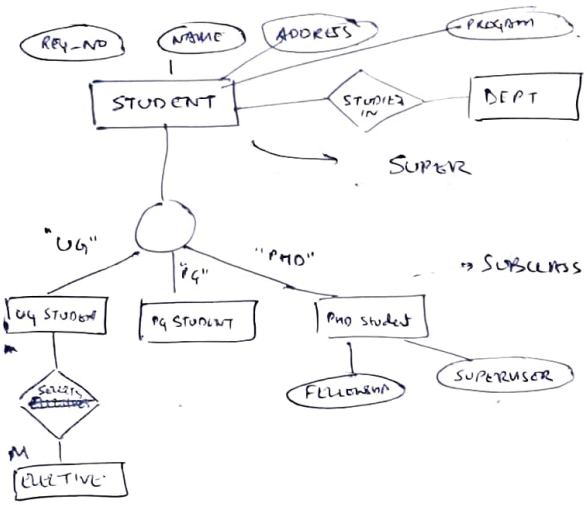
EXTENDED ER (EER) DIAGRAM

⇒ 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

SUPERCLASS
|
SUBCLASS → Subclass is a of superclass.



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

SPECIALIZATION → Started with a generic class/super class. Thereafter, finding the special cases. Form the subclasses → TOP DOWN APPROACH

GENERALISATION → Started with subclasses, later finding commanities to form the superclass → BOTTOM UP APPROACH

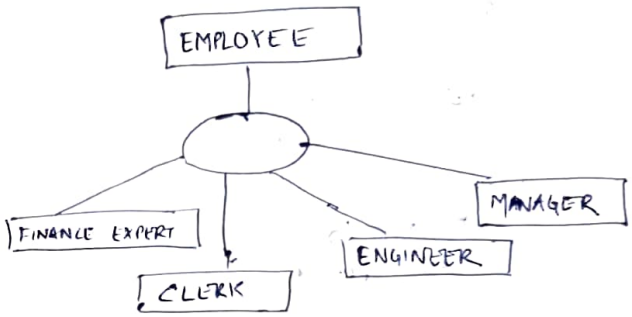
A SUPERCLASS INSTANCE
⇓
TO WHICH SUBCLASS IT BELONGS

Predicate Defined (As per certain criteria) ⇓ Defined
• If it can be defined based on a particular attribute of superclass
ATTRIBUTE DEFINED.

USER DEFINED (If not predicate defined, Then to be known from user)

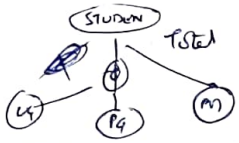
~~Constr~~ → Constraints to define the subclasses

- A SUPER-SUBS CLASS → Disjointness constraint in association, if a superclass instance can belong to only one subclass, the the association is disjoint
- If a superclass instance can belong to multiple subclasses ⇒ Overlapped.



* An employee may be both, an engineer and a manager. (Overlapped)

- Completeness constraint → Total/Partial.
→ Association between superclass and subclass is total if each superclass instance belongs to atleast one subclass else partial.
Ex. An employee not in all categories, hence partial



MAPPING SUPERCLASS - SUBCLASS SCENARIO TO RELATION OF RELATIONAL MODEL.

Using Multiple Relation | Single Relation

(A) → One relation for superclass
⇓
One relation for each of the subclasses
Specific attributes
P.K of Superclass
(F.K. referencing superclass and also it will be P.K)
TOTAL/PARTIAL
DISJT/OVERLAPPED

(B) → No relation for superclass
Only one relation per subclass
Attribute of Superclass ∪ specific attribute
→ TOTAL
If partial, scheme does not work
Overlapped → No of copies of same info

SINGLE RELATION

(E) SUPERCLASS ∪ ALL SUBCLASS ATTRIBUTES
ATTRIBUTES

→ Also a type attribute deciding to which subclass it belongs

DISJOINT → only one type can be denoted.

~~OVERLAPPED~~

TOTAL / PARTIAL → TYPE NULL

(D) Superclass ∪ All subclass ∪ Type field(s)
attributes attributes

N subclass
TYPE 1, TYPE 2 ...

eg. n bits,
ith bits ⇒ corresponds to ith subclass

DISJOINT / OVERLAPPED
TOTAL / PARTIAL

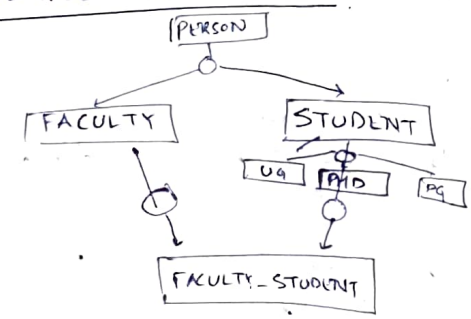
→ NULL VALUE
→ If specific attributes are less in number.

(A) SUPER (A1 , A2 , A3)
SUB1 (A1, B11, B12, B13)
SUB2 (A1, B21, B22)

EQUIJOIN. (to get all details of instances)
SUB1 (Natural join) SUPER.
Easy → to have common instances info for all (SUPER relation)

(B) SUB1 (A1, A2, A3, B11, B12, B13)
SUB2 (A1, A2, A3, B21, B22, B23)
→ to get all the details of a subclass.
→ to get common info for all instances
⟶ OUTER ~~JOIN~~ UNION

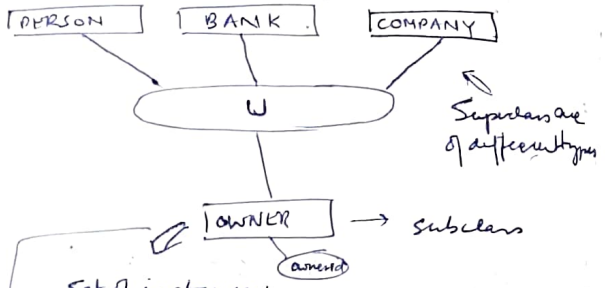
MULTIPLE INHERITANCE



→ If a class has a multiple superclasses
⇓
Sub class is called shared sub-class

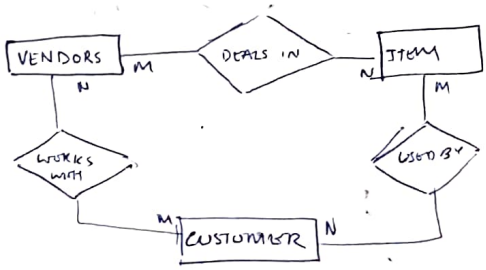
Superclass lattice if its superclasses have common attributes then those to be included only once.

SUPERCLASS HIERARCHY

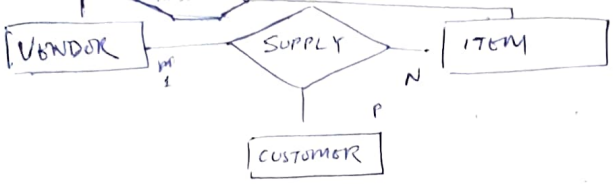


Set of instances;
⇒ union of superclass instances

→ Introduce a new P.K
⇓
Known as surrogate key
→ Copy this surrogate key to superclasses also.



for a customer, for an item who is the vendor
c, i ⇒ v?
[needs a heavy kind of operator]



How to implement?

→ for higher degree relation of ER. DIAGRAM

U P.K. of attribute of all participating entity type,
⇓
U addition attribute

F.K referencing to corresponding entity types.

~~Vendor~~ P.K → U.P.K of participating entity types.

VENDOR (VID, - - - - - - -)
ITEM (IID, - - - -)
CUSTOMER (CID, - - - - - -)

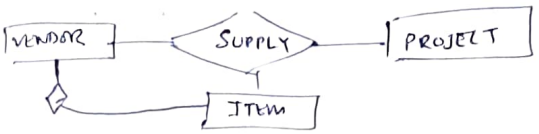
SUPPLY (VID, IID, CID, DT_SUPPLY)

If vendor is 1, VID is unique and needs not be a primary key.

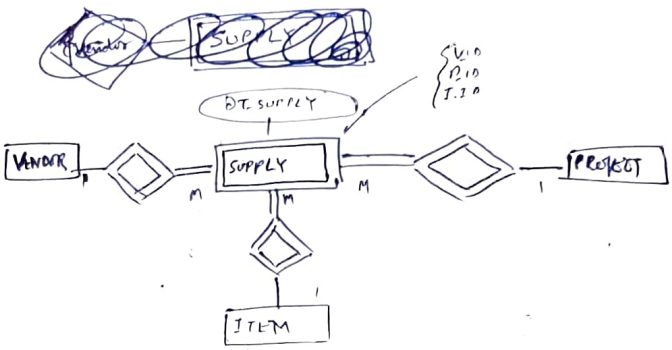
→ If cardinality for an entity type is 1, then its PK may not be required to form primary key ~~for~~ of the relation.

Ex

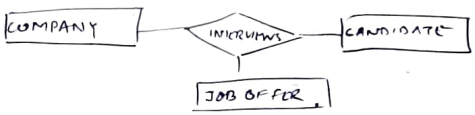
Ternary Relation



⇓



Ex.

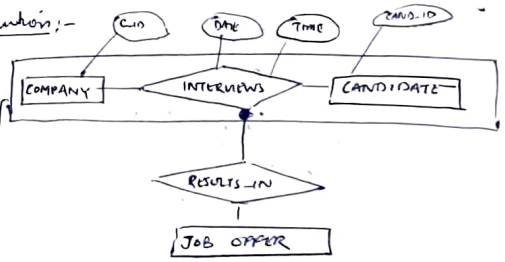


⇓

* In ER-Diagram, relation with other relation is not allow

⇒ Relation with another relation is not allowed in ER Diagram.
⇒ Relation b/w Entity Types.

Solution:-



AGGREGATION

→ A relation along with related entity type into a composite entity type.

03-02-2020

Generalization Specialization
MULTIPLE
INHERITANCE
⇓
• Set of instance of subclass
⇒ Intersection of super class instance
• Attr. of subclass is the collection of attributes of super class

~~UNION~~

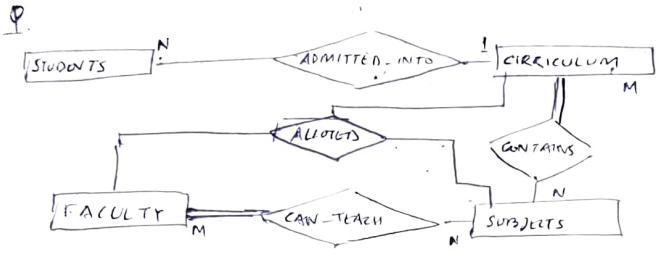
UNION
⇓
subclass instance set → C union of superclass instance set
⇒ not the intersection of super class
~~Operation of diff. types~~

⇒ Super class may be of diff types with diff. attributes
⇒ Attribute of subclass to be chosen carefully depending on the type of superclass.

Partial / Total participation.
⇓
subclass instance set include all the instances of all the superclass

Union may also be represented as generalization - specialization.

Solution 2

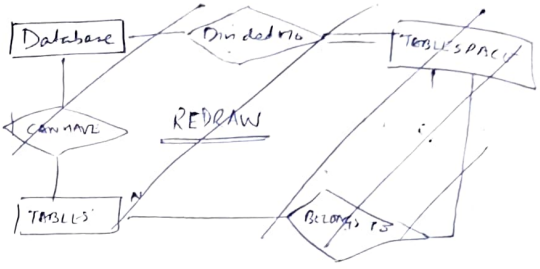


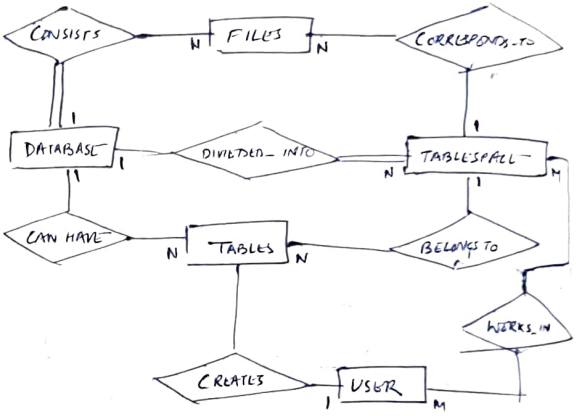
DATABASE

LOGICALLY PARTITIONED INTO NO. OF TABLESPACES

SYSTEM TABLESPACES

DATABASE is a collection of tables





SQL (Structured Query Language)

→ High level Language

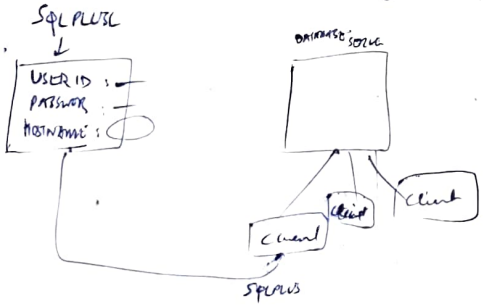
→ Non-Procedural

[We will follow ORACLE syntax]

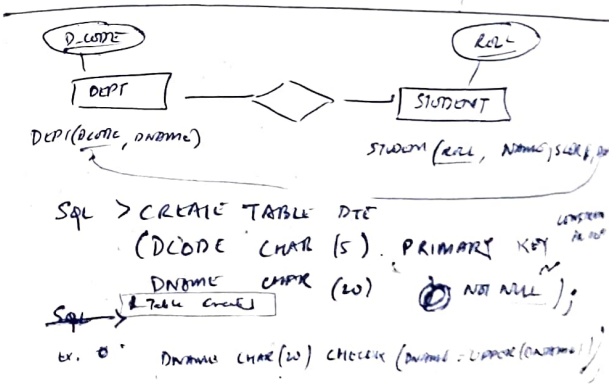
Oracles pinder-level called SQL PLUS

↓

Environment to put my commands/execute also.



SQL> → If successfully done.



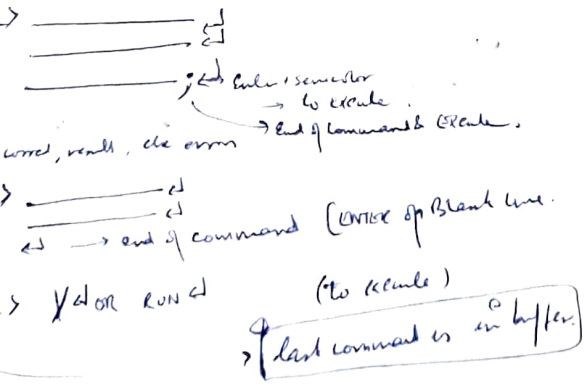
SQL > CREATE TABLE DTE

(DCODE CHAR(5) PRIMARY KEY

DNAME CHAR(20) NOT NULL);

SQL> → Table created

ex. DNAME CHAR(20) CHECK (DNAME = UPPER(DNAME));



SYNTAX :

col. Name | type & size | col. level constraints

CHAR (, int)

VARCHAR (int)

NUMBER (m, n)

m → No of digits

n → No of places after decimal

DATE

Length

Combination of above

UNIQUE

NOT NULL

CHECK (cond)

NOTE: Edit using Notepad

SQL> CREATE TABLE STUDENT
(ROLL NUMBER (3,0) PRIMARY KEY, ~~CONSTRAINT PK_ROLL~~,
NAME CHAR (25),
SCORE NUMBER (5,2) CHECK (SCORE >= 0 AND SCORE <= 100)

*

CHECK (SCORE BETWEEN 0 AND 100)
DEFAULT 0 → if value not given, then 0
DCODE CHAR (5) REFERENCES DEPT.CODE);

name can change — must be same of referred

TO ADD DATA

SQL> INSERT INTO DEPT
VALUES ('01', 'CSE') ; ↵ (use single quotes for strings)
↑ DCODE, | DNAME

SQL> INSERT INTO DEPT VALUES ('02', 'ETCE'); ↵

SQL> INSERT INTO STUDENT VALUES (~~65~~ 1, '~~ABC~~', 85, '01'); ↵

SQL> SELECT * FROM CAT ;

SQLPLUS> DESC tablename ;

SQLPLUS> DESC DEPT ;

NAM	TYPE ?
DCODE	CHAR (5)
DNAME	CHAR (20) NOT NULL

SQL > INSERT INTO STUDENT (ROLL, SCORE, DCODE)
VALUES (2, 90, '01');

SQL > SELECT * FROM DEPT ; → ALL columns

SQL> SELECT DNAME FROM DEPT;
SQL> SELECT ROLL, SCORE FROM STUDENT;
SQL > SELECT ROLL, SCORE AS FROM STUDENT
SQL> SELECT ROLL, SCORE AS, UPPER(NAME) FROM STUDENT

SQL> SELECT 7 FROM DEPT ;
↓
7 → for each tuple used

SQL> SELECT *
FROM STUDENT
WHERE SCORE >= 60 ; ↵

SQL> UPDATE STUDENT
SET SCORE = SCORE+5
WHERE SCORE <50 ; ↵

SQL > DELETE FROM STUDENT
WHERE _____ ; ↵

STUDENT (ROLL, ———)
SUBJECT (SCODE, —, ——)
EXAM ATT (ROLL, SCODE), ———)
RESULT (ROLL, SCODE, SCORE)

⇒ STUDENT & SUBJECT ~~ATT~~ made

SQL> CREATE TABLE EXAM_ATT
(ROLL NUMBER (3,0) REFERENCES STUDENT.ROLL,
SCODE CHAR (5) REFERENCES STUDENT.SCODE,
PRIMARY KEY (ROLL, SCORE)); ↵

SQL > CREATE TABLE RESULT
(ROLL NUMBER (3,0)
SCODE CHAR (5)
SCORE NUMBER (3,0)
PRIMARY KEY (ROLL, SCORE),
FOREIGN KEY (ROLL, SCORE) REFERENCES EXAM_ATT (ROLL, SCO

SQL 05-02-2020

STUDENT (ROLL, NAME, DATE OF BIRTH),)
SUBJECT (SCODE, SNAME, FM, PM)
RESULT (ROLL, SCODE, SCORE)

Q. Find out ~~all students~~ the score for all students in SCODE S1

SQL> SELECT * FROM RESULT
WHERE SCODE = 'S1';
↳ applied on each individual tuple and if it is satisfied then action is taken.

SQL> SELECT (ROLL, SCORE*5) → List of items
FROM RESULT ↓ Col. Name Expression.
WHERE SCODE = 'S1';

BUILT-IN FUNCTIONS

Single Row/Non Aggregate fn
↓
- Acts on individual tuple attribute and returns a value for that

Aggregate function
- Acts on a collection and for the collection it gives a single value.

Single Row function

CHR (int) → gives the character corresponding to ASCII value int.

ASCII (char)

UPPER (CHAR/STRING)
LOWER ( )

ABCXYZ → INITCAP ( ) → first char of each word in uppercase and rest in lowercase.
↓
Abc Xyz

LTRIM ( ) → removes leading spaces
RTRIM ( ) → removes trailing spaces
TRIM ( ) → remove both leading trailing spaces.
skip