15/2/2021

DBMS

Prof. Sanjoy Kumar Saha

DATABASE MANAGEMENT SYSTEM

① DBMS is a collection of interrelated data (i.e. database) & to manage a set of programs is required to interact with the database

→ Data stored in secondary storage (only way for permanent storage)

FILE

② DBMS provides an efficient & convenient interface for storing & retrieving information to/from database.

High level Languages

↳ How to store data in files
How to work with files

if Database ⇒ Collection of files.

↳ we conventional file processing system through high level languages.

→ COBOL was very popular before 90s

Common Business Oriented Language.

Easy for file handling.

Why DBMS? (WRT Conventional file processing systems).

⇓

Lots of difficulties.

Programmer will have to take care of a no. of issues where DBMS will take the major responsibility. (less Burden on programmer).

① → Data Redundancy & Inconsistency

multiple copies

If same information is kept ⇒ Redundancy

Appln Program ⟶ No. of ~~models~~ modules.

Admission → Student
Result → Student
Account → Student

sharing info across the modules.
→ Info. appearing at multiple places

Redundancy ⇒ wastage of space.
leads to Inconsistency.

If multiple copies of same data exist & they are not updated simultaneously then the copies Differ.
In consistency

②
→ Difficulty in accessing data. Write the programs as per the requirement to retrieve necessary data.
Requirement is changed.

λ = 60% → M parameter
40%.

If requirement is changed, existing program may have to be modified / new prog. may have to be written.

DBMS ⟶ SQL Statement

```
SELECT ROLL, NAME
FROM STUDENT
WHERE _____ ;
```

③
→ Isolation of Data
Data of an entity may spread across no. of files.
to get the complete information. These are to be required.

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

Result.
→ ROLL
→ Sub
→ ~~Score~~ Score

Accounts
→ Roll
→ fees paid upto.

④ Concurrent Access Anomalies.

↓

If some piece of data is simultaneously accessed by no. of process/threads, then it is concurrent access.

↓

has to be controlled.

→ serialized.

⑤ Security enforcement.

→ has to be enforced.

→ No. of users for the database. User can access only part of the data.

HOD CSE — HOD ETCE — HOD CE.

↓

- CSE UG I
- CSE UG II
- ...

⑥ Integrity Problem

→ Data must satisfy certain constraints

Each student has Roll (unique), Name, ...

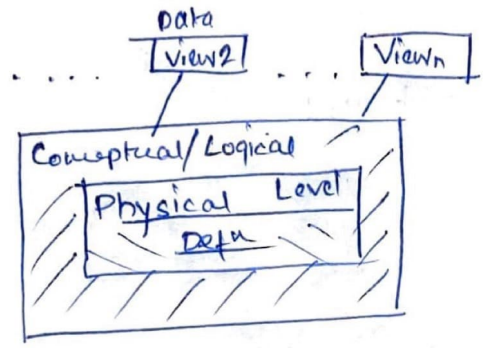
```
STORE()
{
⇒   check_entity_or_not(..)
}
```

→ DBMS provides constructs for specifying the constraints on data & DBMS ensures that constraints are not violated.

Data Abstraction

→ hiding the complexity of internal representation.

→ DBMS provides abstract view of data.



Physical Level

⇓

1) How data is stored ?

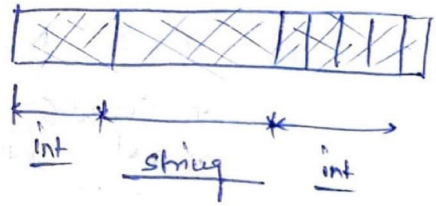
Conceptual Level

⇓

2) What data is stored ?

```
struct Student {
    int roll;
    char name[8];
    int score[5];
}
```

2 + 3 + 10 = 15



View Level

The way data is to be seen by the user.

Data Model

→ There is the concept of data model underlying the structure of a database.

→ It is the collection of conceptual tools to describe the data, their interrelationships data semantics & constraints.

- → Relational Model
- → Network Model
- → Hierarchical Model.

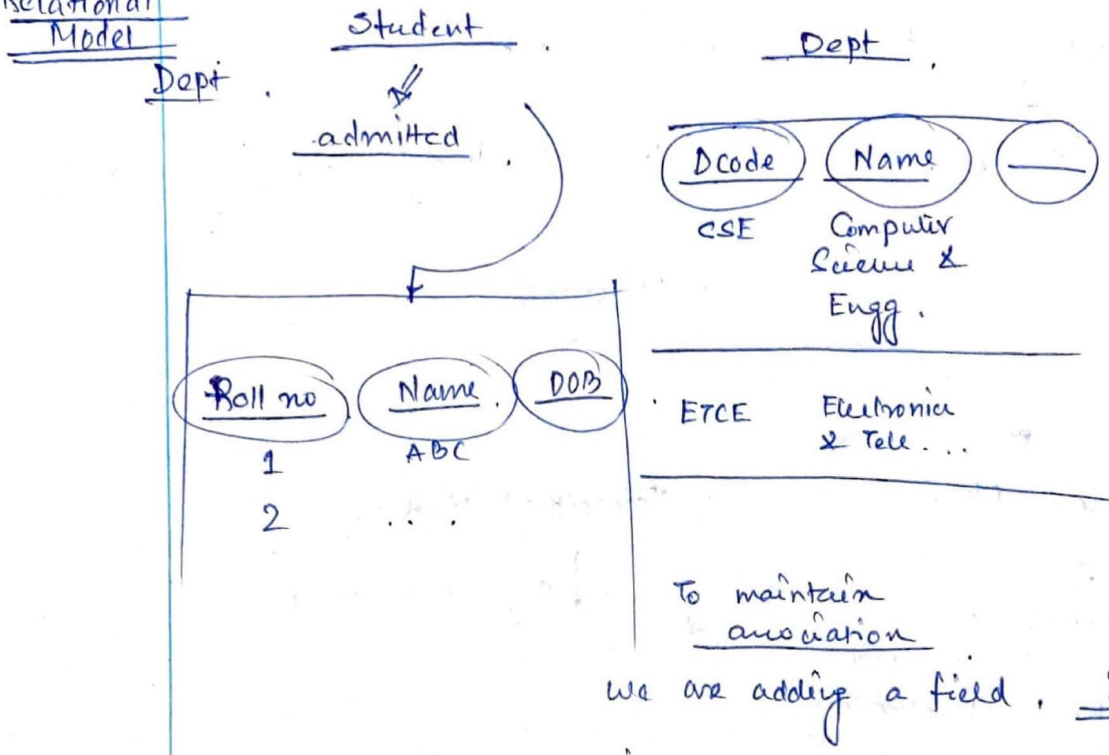
Record Based Model

⇓

Database is a collection of files. File is a collection of records.

Record Base Model

Relational Model



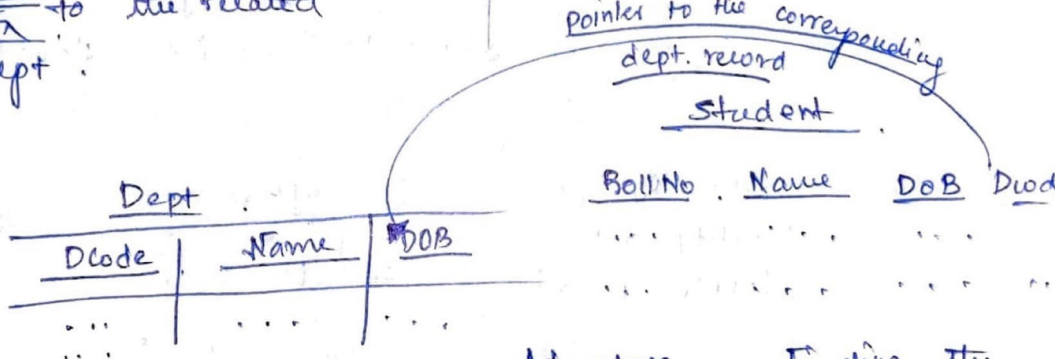
To maintain association we are adding a field, Dcode

In a relational model to establish the relation b/w types of records as a part of the record, a common attribute is kept based on the value of that attribute association is maintained.

Dcode	Roll No.	Name	DoB

Network Model

Instead of keeping the value of common attribute in the record, a pointer/~~ring~~ link to the related record is kept.



Advantage Finding the related record based on the link is faster

Cons. If Blocks are relocated, all links have to be changed.

Different types of related entities

⇓

No. of files → Links in different files give rise to a forest

→ Unorganized

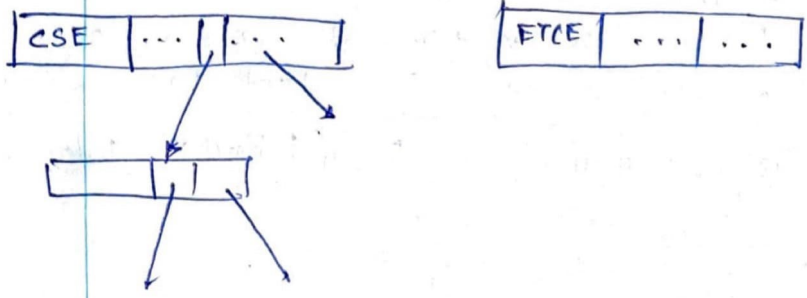
Hierarchical Model

⇓

Records are organised as per a certain hierarchy.

Networked Model

↓ organised version



Instance & Scheme (schema) of a database

→ Overall structure of the database.

→ Student (Roll: int, Name: String, DoB: Date

→ Schema also get modified

But ⇒ Infrequent

add Record
Modify data
delete Record

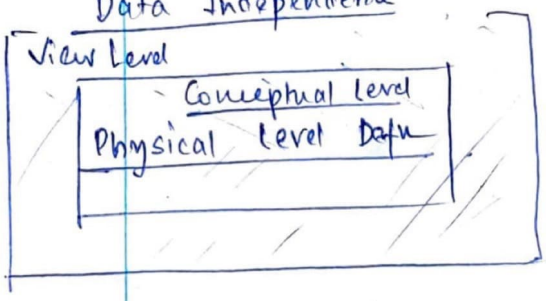
} at different points of time.

Content get changed.

d_1 ⇒ Instance$_1$

d_2 ⇒ Instance$_2$.

Data Independence



→ Capability of modifying the defn at a level without affecting the higher level defn.

Appln.

Physical level defn. independence is achieved if change in the physical level defn does not have any impact on the appln program code.

Conceptual Level Independence

⇓

Conceptual defn changed but no change in appln program.

Prof. Sanjoy Kumar Saha

Physical Level Independence is achieved if changes in physical defn do not have any affect on appln program.

Achievable

To Work with database

⇓

Languages.

DBMS.

Conceptual Level Independence → Abstraction mostly we deal with logical def

Commercially → SQL.

Structured Query Language.

Procedural — Non-procedural

Manipulation Language.

DDL.

(Data Defn. Language)

⇓

Used to specify/ modify the schema of the database.

DML.

(Data Manipulation Language)

⇓ Delete

CR UD operations

Create, Retrieve, Update

STUDENT

```
struct rec
{
    char name[31];
    int roll;
    ...
};
```

Along with what data is read, how to get the data is also specified.

Only what is reqd. & hasn't to be specified not how.

The non-procedural part of SQL is in the context of 3rd and 4th gen prog. lang & DML.

Database Manager

⇓

S/W module forming the core part of DBMS.

→ Interaction with file manager of operating system.

File management ⇒ Responsibility of DISK Blocks.

Access data → Interact with secondary storage. Access the disk

Database manager interacts with file manager to work with data in disk.

Database Manager

→ The core part of DBMS S/W Module forming

→ Integrity enforcement.

Data must satisfy condition/ consistent.

Whenever there is a change in database instance ⇒ constraint.

↳ Are to be validated;

→ Security Enforcement

→ All data is not accessible by each & every user.

→ Who can access what & for what purpose

→ Security is enforced

→ Backup & Recovery

→ Failure may occur.

→ Has to be Recovered to ensure that database remains in valid state

Account

ACNO

BALANCE

Old valid state (No transaction)

⇓

New valid state (The transaction will succeed)

A —10,000→ B

⇓ ⇓

Balance −10000 Balance +10,000

→ Concurrency Control.

DBA (Database Administrator)

→ Overall Schema Defn.

→ Storage structure access specifier.

Oracle.

→ Student(—)
→ Dept(—)

What are the files?

Functional modules of DBMS.

→ Database manager.

→ DDL Compiler. ⇒ Translate DDL statement & Generate metadata which is stored into data dictionary. (part of database)

```
CREATE TABLE STUDENT
{
    ROLL NUMBER UNIQUE.
    ...
}
```

① Student created by user

② Student ⇒ Roll Type Name —

Generate.

→ DML Pre-compiler

DML ⇒ Non-procedural ⇒ For efficient & Convenient interaction with database.

→ In an application procedural aspects are also required.

↳ Loop, if else, var, constant, etc.

HLL → Procedural feature, etc.

HLL + DML

⇓

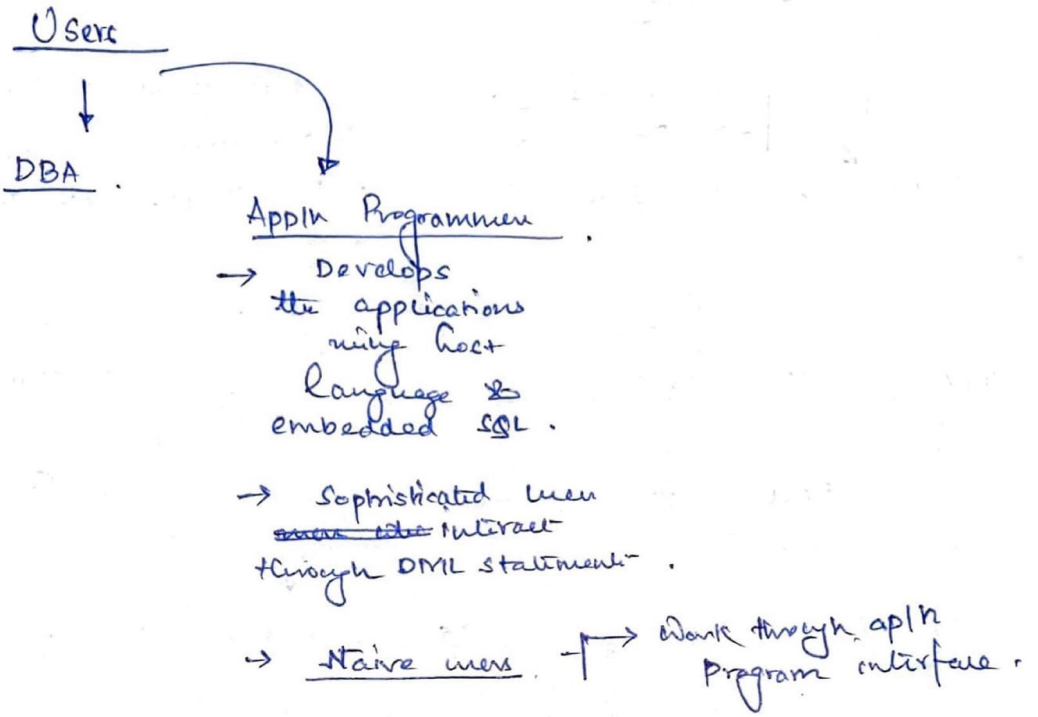
Host Language..

DML statement are embedded, ⟶ DML precompiler

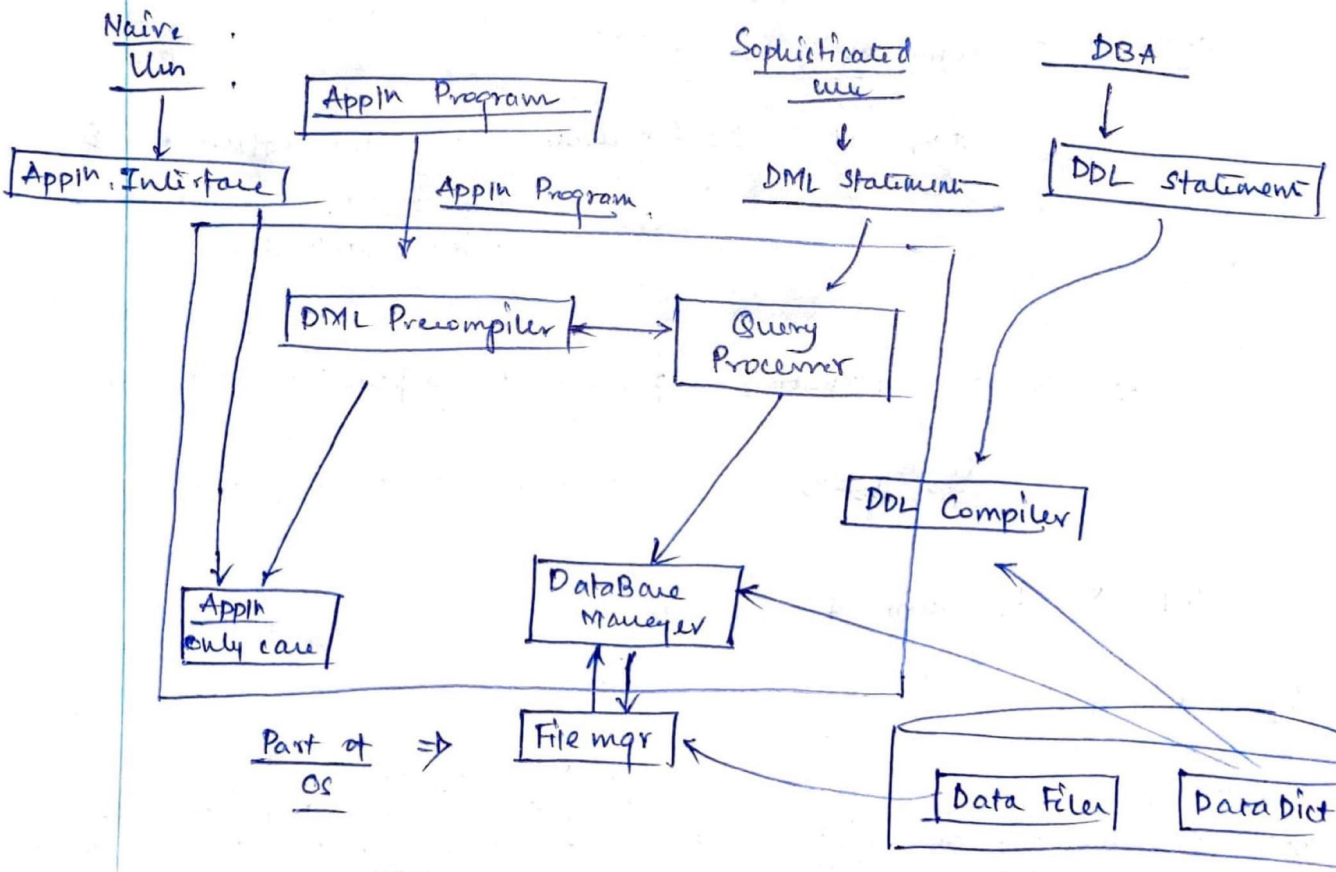
Convert embedded DML statement into equivalent host lang. statement.

→ **Query Processor** ⇒ Query Processor translates user given query into a form that the system understands.

Also performs an optimisation to convert the query into an equivalent but efficient form.



Overall Structure of DBMS



22.02.2021

DBMS

Prof. Sanjay Kumar Saha

→ Network Model
→ Hierarchical Model
→ Relational Model

↓

→ Database is a collection of relations.
→ Relation is a table of value.
→ File containing records.

Column Heading

Roll	Name	DOB	Phone	Email
1	ABC	..	-	-
2				

Each row → describes an entity / concept

Collection of Related values.

Formal

Table → Relation.
Row → Tuple.
Column Header → Attribute

DOMAIN

→ A domain D is a set of atomic values

Indivisable

→ Each attribute has a DOMAIN.
→ In a tuple, attribute can take a value from its domain.
→ DATATYPE for an attribute describes the domain.

Roll . Number (3,0)

→ 0 – 999

Composite as it can be divided

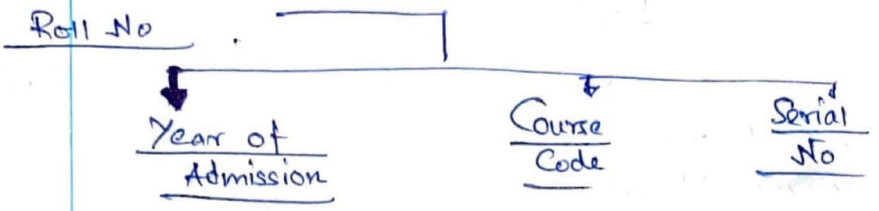
Phone No.
8-digited Nos.

area code → Serial Code

Atomic / Indivisibility is usecase specific

Name.

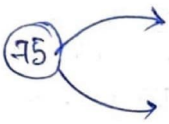
- String(50).
- First Name
- Middle Name
- Last Name.



→ A relation schema R, denoted by $R(a_1, a_2, \ldots, a_n)$ is made up of a name R and a set of attributes $A_1, A_2, A_3, \ldots, A_n$

→ Each attribute A_i is the name of the roll played by the value from domain of A_i. Each attribute A_i has a domain $DOM(A_i)$

Roll ⇒ 3 digited integer
Score ⇒ 3 digited int.



~~Degree of Relation~~ ⇒ No. of attributes in the schema.

A Relation/Relation State

⇓

Relation Schema $R(A_1, A_2, \ldots, A_n)$ is denoted by $r(R)$.

↓

A collection of n-tuples.

$r = \{t_1, t_2, t_3, \ldots, t_m\}$

t_i is a n-tuple of the form

$\langle v_1, v_2, v_3, \ldots, v_n \rangle$

↓

Each n-tuple is an ordered list of n values v_i

$v_i \in dom(A_i)$

v_i is the value for ith attribute

R ⇒ STUDENT (ROLL, NAME, DOB, PHONE).

degree - n

tuple ↳ (1 ABC — —)

t ⇒ tuple t[Ai] ⇒ value of Ai in tuple t.

Formal defn of Relation r(R)

A relation r(R) is a mathematical relation of degree n on Dom(A1), Dom(A2) .... Dom(An).

r(R) ⊂ DOM(A1) × DOM(A2) .... DOM(An).

⇓

Subset of cartesian product of the attribute domains.

ROLL ⟶
Name ⟶ All possible allowable values } Domain.
DOB ⟶
PH ⟶

Cardinality of domain cartesian pool → |DOM(A1)| × |DOM(A2)| × ... × |DOM(An)|

→ tuples in r(R) ≤ _____

Relation state at a time ⇒ Instance of the relation,

Consisting of valid tuples of that point of time.

What is a relation Intension ?

① Relation Intension ⇒ Schema — NOT Intention

↓

- SION

② Relation Extension of a relation ⇒ Instance Relation State.

Characteristics of relation in relational model

- Tuples are unordered.

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

- Ordering of attributes in a tuple
  - May not be ordered.
  - May be ordered in a tuple

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

& $t_1 = \langle v_1, v_2, \ldots, v_n \rangle$ → depends on the tuple. are ordered according to the appearance of attribute in the relational scheme.

$v_i \in dom(t_i)$,

[One to one mapping]

may not be ordered

A relation $r(R)$ for the schema R is a finite set of mappings from R to D.

$$D = \bigcup_{i=1}^{n} Dom(A_i).$$

t_i ⇒ collection of Attribute value pairs.

$\{(A_i)(v_i)\}$ — v_i: Current value; A_i: Name of attribute

- → Tuples are unordered.
- → Attribute in a tuple may/may not be ordered.
- → In a tuple, an attribute can have Atomic value & a single value.

v_i ⇒ Value of A_i.

$v_i \in DOM(A_i)$.

Attributes are composite & single valued in a tuple.

non OR Simple

1st Normal Form

→ Null Value

An attribute value must be taken from its domain/ it may be NULL. ⇒ Does not mean 0 or Does not mean space.

↓

Absence of any value

Design Criteria

↓

Minimize the occurrence of Null values.

Null Value

→ Attribute is not applicable for that entity.

→ Applicable but value is not known.

↓ ↓

Intentionally Not Known — Truely.

⇒ ~~Meaning of a~~ relation, a relational schema can be Meaning of a ~~(check later)~~. interpreted ~~as~~

→ as a type of declaration/ assertion.

Student (Roll, Name, Dob).

→ Data must satisfy certain constraints.

Constraints in DBMS.

→ Model Based.
Constraints that are inherent in the underlying model.
Relational model ⇒ Tuples unordered attribute in tuple.
Attribute → Simple/ Non-Composite
↓
Single Valued

Schema based constraints
→ Constraints that can be defined alongwith schema specification.

→ Appln based constraints

→ Score cannot exceed full marks

→ Score < Total Salary of his/her boss.

Construct Trigger

Data dependency

Functional dependency

CREATE TABLE STUDENT (...).

For each ~~datatype~~ attribute → specify the datatype → way to create/define the domain.

{ Domain Constraints

(~~Please check Recording~~)

Primary Key Constraints

→ Let SK be a subset of attributes from the schema $R(A_1, A_2, A_3 \ldots A_n)$. Be such that any pair of tuples $t_1, t_2 \in r(R)$.

(~~Please check Recording~~)

$t_1(SK) \neq t_2(SK)$.

Then SK is called superkey of R

Superkey

(~~Please check Recording~~)

No. of Superkeys

Minimal Superkey is called Candidate Key.

→ of which no proper subset is a superkey

Result (Roll, Score, Scode).

Primary Key => Multiple candidate key exist in a Relation.

One of the candidate keys is chosen by the designer as the primary key

Unique ~~fond~~ Unique

Student (RegNo, Roll)

Within the scope of the application, which candidate key is often used in reality for identifying the ~~identi~~ entity denoted by the tuple.

→ Smaller in size is prefered.

→ Alphanumeric is prefered over alphabetic.

Constraints

- Model Level
- Schema
- Application.

Domain, Primary Key } Can be specified along with Schema Defn

valid state (consisting of tuples satisfying constraints)

Relational Database

A relational database is the collection of relations

schema of relational database consists of $S = \{R_1, R_2, R_3, \ldots, R_n\}$ (Set of Schema) & Set of Integrity Constraints

State of Relational Database. $= \{r_1, r_2, r_3, \ldots, r_n\}$
where $r_i = r_i(R_i)$.

→ Entity Integrity constraints

Primary Key is not NULL.

used to identify the entity/object tuple in a rel^n.

absence of any value.

→ Referential integrity constraint & Foreign Key.

DEPT

DCODE (Primary Key)	DNAME
D_1	-
D_2	-
D_3	-
D_4	-

dist of Dept.

STUDENT

NAME	DOB	PHONE	DCODE	REGN
			D_1	R_1
			D_2	R_2
			D_3	R_3
			D_4	R_4
			D_5	R_5

Any value kept here must be present in Dept.

R_1 & R_2 are two relations. Let P_K be the subset of attribute in R_1 and it is the primary key in R_1. Let F_K be the subset of attribute in R_2.

F_K is the foreign key in R_2 referencing P_K of R_1 If →

- Domain of P_K in R_1 and domain of F_K in R_2 are same.
- For any tuple $t_2 \in r_2(R_2)$, either $t_2(F_K)$ is null or there exist a tuple $t_1 \in r_1(R_1)$. such that $t_1[P_K] = t_2[F_K]$.

R_2 references R_1.

R_1 is called Referenced Relation.
R_2 is called Referencing Relation.
F_K of R_2 refers P_K of R_1.

Referential Integrity.

DML Statement

Relation is defined ⇒ DDL Statement

Retrieve data from a Relation

- Add data
- Modify data
- Delete data

Impact of Referential Integrity on DML operations.

DML operation	Referenced Relation	Referencing Relation
Add Data	→ No impact of Referential Integrity. Tuple must have unique value for P_K. [P_K Constraint].	→ Value in foreign key field must be present in the primary key field of referenced relation else not allowed.
Modify Data	→ If primary key is changed (Bad practice) & if the old value is refered by referencing relation then not allowed.	→ If foreign key field is changed then new value must be present in primary key field of referenced relation, else not allowed.
Delete Data	→ If primary key value of the tuple to be deleted is refered in referencing relation, then not allowed.	→ No problem.

→ Model Level
  → Schema Lad.
→ Other constraints
  —— Appln. Level
  Trigger.

Functional Dependency
(Data Dependency).
→ Formal design of database.

Subject
SCODE FM PM

Result
ROLL SCODE SCORE

1. Adding Record → Cannot exceed FM

2. Modify Record → Piece of Code → Triggered by

Relational Model

→ To work with the Relations.
⇓
Model should provide certain language

→ Relational Algebra (Procedural)
→ Relational Calculus (Non-Procedural).

An expression is a sequence of operations on one/more relations + provides another relation as output.

Expression specifies what is required (No operation. How?) → Output one/more relation. o/p another relation.

SELECT
STUDENT
⇓
ROLL=1

$\sigma_{ROLL=1}$ (Student),
→ Select operation.

$\sigma_{SCORE > 80}$

Calculus
$\{ t \mid t \in STUDENT \text{ AND } t.ROLL = 1 \}$.
$t.ROLL > 80$.

Relational Algebra

→ Select operation

$\sigma_{cond}(R)$

works on a Relation & Returns a relation as output

Returns the Tuples from R satisfying the predicate

Schema → Same as input relation

State → formed by tuples in input relation satisfying the predicate

$\sigma_{DCODE = 'R_1'}(STUDENT)$

filtering cond"

$\sigma_{HS\_SCORE \geq 80}\left(\sigma_{DCODE = 'D1'}(STUDENT)\right)$

$\sigma_{DCODE = "D1"}\left(\sigma_{HSCORE \geq 80}(STUDENT)\right)$

⇓

Set of tuples with DCODE = "D1" & H.SCORE >= 80

Cascading of SELECT OP.

$\sigma_{P2}(\sigma_{P1}(R)) = \sigma_{P1}(\sigma_{P2}(R))$

select operation is commutative

Relational Algebra

Defn. Importance of relⁿ algebra.

→ Provides a formal foundation for relational model operation.

→ Basic of implementation & optimization of query on relations.

→ Some of the concepts are also in SQL.

SELECT - OPRN ⟶ filtering

→ PROJECTION

⇓

In order to consider selected attribute/expr. in the o/p schema.

$\pi_{list}(R)$

list ↑ — I/p Relⁿ

① No. of tuples in o/p schema ≤ No. of tuples in i/p schema.

② o/p schema is same as i/p schema.

Output relⁿ

⇒ No. of tuples is same as no. of tuples in i/p defn

⇒ Schema will consist of the attribute/defn given in list.

$\pi_{DCODE, BASIC}(EMP)$

Rules ⇒ Set of tuples.

Relⁿ Model constn.

⇒ No two tuples are same.

D1 - 70K
D2 - 70K

→ Duplicate Elimination Projection.

list 2 ⊂ list1

$\pi_{list1}(\pi_{list2}(R)) = \pi_{list2}(\pi_{list1}(R))$

⇓

$\pi_{list1}(R')$

list1 ⊂ list 2.

if list1 = list2 OK

list1 ⊂ list2 OR list2 ⊂ list1 Not happen.

Projection is NOT Commutative

Rename

$R(A_1, A_2, \ldots, A_n)$.

$\rho_S(B_1, B_2, B_3, \ldots, B_n)$.

In this expr, R is ~~retained as~~ renamed as S.
& A_i is renamed as B_i.

~~$\rho_S(R)$~~ $\rho_S R(A_1, A_2, \ldots, A_n)$.

only R is renamed as S.

~~Assignment~~ $\rho_{A_1, A_2, \ldots}$ R(—).

$A_1, A_2 \ldots$ renamed.

~~$S(B_1, B_2, \ldots)$~~.

Cartesian Product

$R_1 \times R_2 \longrightarrow$ $B_1, B_2, \ldots$ o/p Relⁿ.

$A_1, A_2, \ldots$ Schema. Schema of R_1 & Schema of A_2.

No. of tuples: Every tuple $t_1 \in r(R_1)$ will combine with each of $t_2 \in r_2(R_2)$, one at a time.

Can lead to spurious tuples

$N_1 \times N_2$

No. of tuples in R_1 — No. of tuples in R_2.

σ (DEPT × EMP)
DEPT.DCODE = EMP.DCODE

Cartesian Product needs filtering to retain only meaningful tuples.

$\pi_{ENAME, DCODE}\left(\sigma_{DEPT.DCODE = EMP.DCODE}(DEPT \times EMP)\right)$

JOIN.

$R_1 \bowtie R_2$

$\pi_{ENAME, DCODE}\left(DEPT \underset{DEPT.DCODE = EMP.DCODE}{\bowtie} EMP\right)$

θ-join.
but if it is based on equality of attribute values then it is called equijoin

Equijoin.

~~DEPT~~

EQUIJOIN.

Attributes are compared for equality coiu all appears in the schema.

For joining attribute in more than one column, same info. is obtained as o/p rolm. (Redundancy)

Natural Join

Name must be same.

→ Modified version of equijoin.
→ equality of all common attribute pairs from the schema being joined.

$R_1(x, y, z, w)$
$R_2(P, Q, X, Y)$

$R_1 \bowtie R_2$

$R_1 \bowtie R_2.X = R_2.X$ and $R_1.Y = R_2.Y$,

only one from both the input relation coiu be present.

$EMP \leftarrow \left(\rho_{DEPT(DCODE)}^{DEPT(DCODE)} \right)$

~~From~~ ~~to~~
From → To

CART - PRODUCT
↓
JOIN.

$\Pi_{EMP.NAME,\ TEMP.NAME}$

$= \sigma_{TEMP.BASIC = EMP.BASIC \ \wedge\ EMP.ENAME < TEMP.ENAME} \left(EMP \times \rho_{TEMP}(EMP) \right)$

SET OPERATIONS

UNION
INTERSECTION

} More than one input relations.
R_1 & R_2.
Input relations must be union Compatible

↓ Set operations on R_1 & R_2
① Both are of same degree
② Domain of (A_i) is same as domain of (B_i).

$R_1(A_1, A_2, \ldots, A_n)$
$R_2(B_1, B_2, \ldots, B_n)$

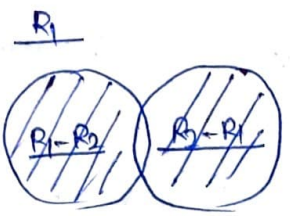
Union

$R_1 \cup R_2$

o/p schema = Name as R_1 (R_2).
It must have all the tuples from R_1 & R_2. common tuples must appear only one.

$R_1 \cup R_2 = R_2 \cup R_1$ (Commutative)

$R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3$ (Associative)



Intersection

$R_1 \cap R_2$ o/p schema = same as R_1 or R_2,
tuple that are common to both in R_1, R_2

Fundamental

1. Select
2. Project
3. Cart Product
4. Rename
5. Union
6. Minus

Join
Natural join
Intersection
↓
Can be derived from fundamental operation

$R_1 \cup R_2 - [(R_1 - R_2) \cup (R_2 - R_1)]$
$\Downarrow$
$R_1 \cap R_2$.

JOIN Operation

$R_1 \bowtie R_2$ ⇒ based on the ~~neq~~ equality of common attribute

tuples from $r_1(R_1)$ and $r_2(R_2)$ having match with ~~eo~~ each other will appear..
tuples in either of the relations without com. match do not appear in the output.

All employee tuple must and it possible with dept info.

Outer Join

EMP ⋈ DEPT

~~for an employee DEPT is not~~ ⇒ will not appear
(existance of that employee becomes invisible).

① EMP ⋈ DEPT
left outer join.
If DEPT is not present then it is padded with NULLS.

② EMP ⋈ DEPT
~~left~~ Right outer join

③ Full outer join
↳ Emp ⋈ Dept

Outer Union

deals with the relations having partial union compatibility.

$R_1 \cup R_2$

↓ ↓

Union - compatible

R_1 (X, Y, A, B, C)

R_2 (X, Y, P, Q).

R_1 OUTER UNION R_2.

Schema ⇒ X, Y, A, B, C, P, Q.

Tuple pair in R_1, R_2 that matches based on common attribute

⇓

Win contribute one tuple in the o/p. value of addn attribute of R_1 Win come from t_1 & addn attribute of R_2 Win come from t_2.

Unmatched Tuple

→ they are matched with a null tuple. ✓

OUTER UNION ✗ FULL OUTER JOIN are same. If joining is done based on the equality of all common attributes

Relational Algebra

Division Operation.

$R(Z)$ → schema.
$S(X)$ → schema.

$X \subset Z$

$Y = Z - X$.

$R(Z) \div S(X) = T(Y)$.

$T(Y)$ includes a tuple t if t_R appears in $R(Z)$.

- such that $t_R[Y] = t$
- and with $t_R[X] = t_S$.

for every tuple of t_S in $S(X)$.

Values of Y which is present for every value of X.

$10 \div 4$.
2 times it appears.

R ⇒ Result

Roll, Scode, Score
→ No entry for absent candidate

SUBJECT(SCODE, SNAME, ...)
RESULT(ROLL, SCODE, SCORE).

- Find Roll no. of the students who did not miss any exam.

$\pi_{Roll, Scode}(RESULT) \div \pi_{SCODE}(SUBJECT)$

Division

↳ Derived operation.

$R(Z) \div S(X) = T(Y)$
$X \subset Z$
$Y = Z - X$.

$T_1 \leftarrow \pi_Y(R)$
$T_2 \leftarrow \pi_Y((T_1 \times S) - R)$
$T \leftarrow T_1 - T_2$

π_Y → Y value for which XY combination not in R

XY combinations not in R

eg.
$T_1 \leftarrow \pi_{ROll}(R)$,
$T_2 \leftarrow \pi_{ROll}((T_1 \times S) - R)$,
$T \leftarrow T_1 - T_2$.

Roll nos. for whom all roll, scode combination exists.

Aggregate fn & Grouping

Acts on a collection & returns a single value for the collection

$\Pi_{UPPER(Name)}$ (Student)

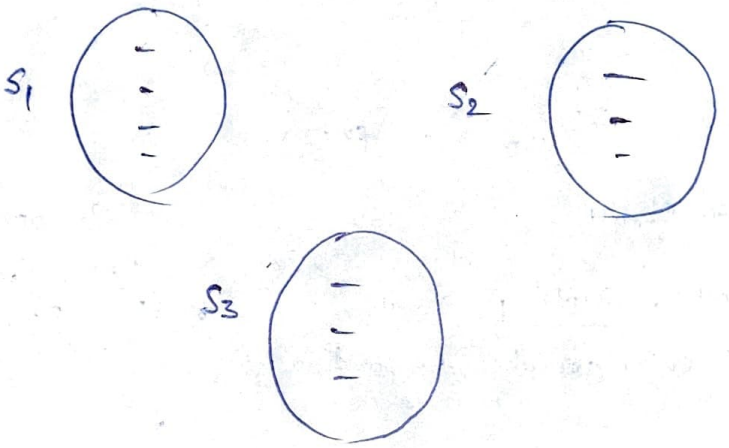
non-aggregate fn.

that acts on individual tuple & generates an output for each tuple.

$\mathcal{F}_{Average(\text{fn expression. eg. Score}),\ STDEV(Score)}$ (Result).

ρ(Avg, Stdev).

Grouping



Result (Roll, Dcode, Scode, Score)

Dept wise, Sub. wise, highest score.

Sum()
← Average()
← Std.dev.

Null values are ignored.

$\mathcal{F}_{DCODE,SCODE}$ (Result).
MAX(Score), MIN(Score), ...

Relational Calculus

⇓

tuple relational calculus

Relⁿ Algebra	Relⁿ Calculus
Procedural	Non-procedural
	What data is required? (Not how).

- find the name of all the students?

$\{ t \mid t \in STUDENT \}$

Range Relⁿ.
The tuple variable ranges over this relⁿ.

⇓

tuple variable that hold a tuple from its range relation.

- find the name of all students living in the city of Kolkata.

$\{ t.name \mid t \in student \text{ and } t.city = \text{"Kolkata"} \}$.

- for every student show name & corresponding dept. name.

$\{ s.name, d.name \mid student(s) \text{ OR } s \in student \text{ AND } DEPT(d) \text{ AND } S.DCODE = D.DCODE \}$

- Show the names of the students studying in dept. named as XYZ.

$\{ s.name \mid Student(s) \text{ AND } (\exists x)(DEPT(x) \text{ AND } x.Dcode = S.DCODE \text{ AND } x.Dcode = \text{"XYZ"}) \}$

s.name → Pre-variable

May have to introduce addⁿ. tuple variable on the right side of |.

unbounded.

Bounded type variable is associated with bounding quantifier.

$(\exists x)$ → Existential quantifier.

$(\forall x)$ → universal quantifier. ∀ (for all) x.

Range relation is universal relation

- find roll no. of student ≥ 50 in all subject.

$\{ st.roll \mid student(st) \wedge (\forall sx)\ (Not\ subject(S_t)$
$OR\ (\exists ro)\ (Result(re)$
$AND\ re.roll = st.roll$
$and\ re.scode = sx.scode$
$and\ re.score \geq 50)))\}$

for every subject in subject x/n

Be very careful

Relational Calculus

for all quantifier.

STUDENT (Roll, Name, ...)
STUDENT (Scode, Sname, Stype,
T => Theory
S => Sessional.
RESULT (Roll, Scode, Score).

Find the students who have scored 50 or more in all subjects.

{ St.name | Student(St) AND (∀Su)(NOT subject(Su) OR ↑
St[name] true for all tuples in universal relation.

NOT
Su.Stype = 'T'
OR
(∃r)(Result(r) AND
r.Roll = St.roll
AND r.Scode = Su.Scode
AND
r.Score ≥ 50))}

Eqv. to,

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

⇓

NOT (exist x that does not satisfy P(x)).

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

↳ Not [All x not satisfying P(x)]
Some x for which P(x)

$(\forall x)(P(x)\ AND\ Q(x))$
$\equiv NOT\ (\exists x)(NOT\ P(x)\ AND\ NOT\ Q(x))$.

$(\forall x)(P(x)) \equiv (\forall x)(P(x) \text{ OR } Q(x))$

$\equiv \text{NOT}(\exists x)(\text{NOT } P(x) \text{ AND NOT } Q(x))$

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

$\equiv \text{NOT}(\forall x)(\text{NOT } P(x) \text{ OR NOT } Q(x))$

$(\exists x)(P(x) \text{ OR } Q(x)) \equiv \text{NOT}(\forall x)(\text{NOT } P(x) \text{ AND NOT } Q(x))$.

IMPLIES

Talking about subset

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

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

there does not exist any x that satisfies $P(x)$ for all x $P(x)$ is false.

Relational Algebra & Calculus ⇒ have same power of expression

Safe Expn

⇓

Returns finite no. of tuples.

$\{x \mid x \notin \text{Student}\}$

⇓ NOT Student(x)

Any tuple from universal rom except : STUDENT.

Entity Relation (ER) Model

↳ Designing the database
↳ looking into the data.
Requirement from conceptual / logical level.

Requirement is captured & presented in the form of entity relation (ER) diagram.

ERD

⇓

Stores the entity types & relation the entity type.

It also presents the description of the entity & relationship

Entity ⇒ Entity is a real life object / concept distinguishable from each other.

Collection of similar type of entities ⇒ entity set

Entity type ⇒ defines the structure of an entity set

(1, "abc", 01-Jan-2000, ...)

STUDENT (Roll, Name, Dob)

⇓

Entity type

Entity type

⇓

Relation Schema

(Extension)

Entity set

⇓

Relation state

(Extension)

Rel^n. Association b/w entities.

⇒ Rel^n set collection of same type of relationship.

⇒ Rel^n type defines a relationship set.

DEPT ⇒ Entity type
STUDENT ⇒ Entity type

$S_1 \longrightarrow D_1$
$S_2 \longrightarrow D_2$
$\vdots$
$S_n \longrightarrow D_n$,

$E_1, E_2, \ldots, E_n$ are the entity types.

A relationship set R $= \{(e_1, e_2, \ldots, e_n) \mid e_i \in E_i\}$.

ATTRIBUTE

- Simple v/s Composite attribute
  - Simple → Atomic cannot be divided.
  - Composite → Consisting of multiple simple attribute
- STORED / Derived attribute
  - Attribute that can be obtained from other attribute → Derived attribute
  - Attribute for which value has to be known only by storing → Stored attribute.
- Single valued v/s multivalued
  - for an entity it can have one value.
  - for any entity → multiple values may be present.
- Null valued attribute
  - ① Null value is allowed.
- Complex attribute
  - Nesting of composite & multivalued attribute.

Simple attr.

X(Y, Z)

Composite attr.

Contact info.

Consisting of address, multiple line.

CONTACT INFO (address, {land-line#}),

CONTACT INFO (address (Home No, street, Pin), {land-line (Area code, No)}}

Does not permit multivalued / Composite values.

Value set.

An attribute A → V

- A → Domain.
- V → Set of values that A can take.

A is a fn rule that maps an entity E to P(v).

[Power set of v]

A: E → P(v),

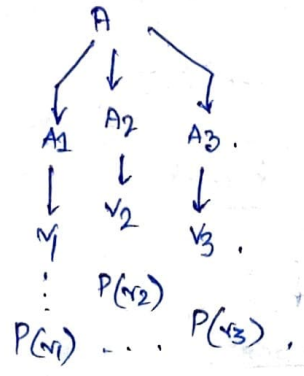
v ⇒ value set of A.

P(v) ⇒ power set of v.

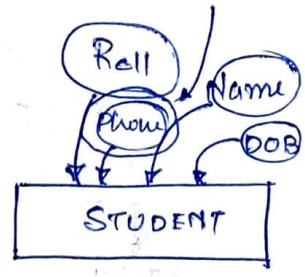
P(v) = set of all possible subsets of v.

Empty $({}^nC_0)$ $+ {}^nC_1 + {}^nC_2 + {}^nC_3 + \dots + {}^nC_n$.

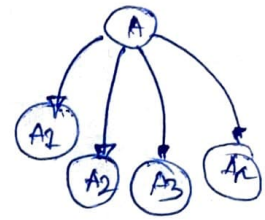
Mandatory
v/s
optional attribute } → roll, name, ..

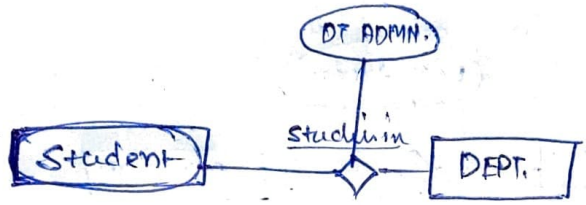


Multivalued



Entity type



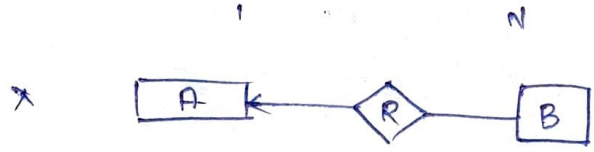


ERD → Degree of a relation.
No. of entity types involved.

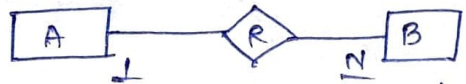
Constraints on Binary Relation

Structural constraint { → Mapping constraint/cardinality ratio
→ Participation constraint.

An entity of type A can have association with atmost one entity of type B and vice versa.



Follow



An element of A can have relationship with multiple element of B. But, an element of B can have relation with atmost one element of A.

Mapping Constraint

⇓

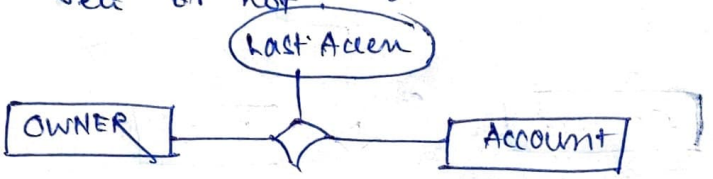
An element of one entity type can have relation with how many elements of the type.

Existential dependency.

Participation Constraint

- Total
- Partion.

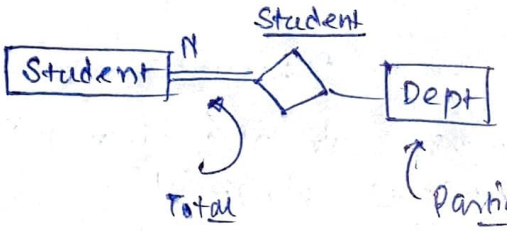
. Whether every entity is bound to take part in the relt or not.



An entity type E totally participates in a relⁿ.

⇓

for each entity of E must take part in relⁿ.



ER Diagram

Entity

Attribute

Relation

Key attribute

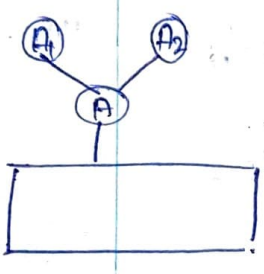
A → E

Binary Rel^n

⇓

Constraints

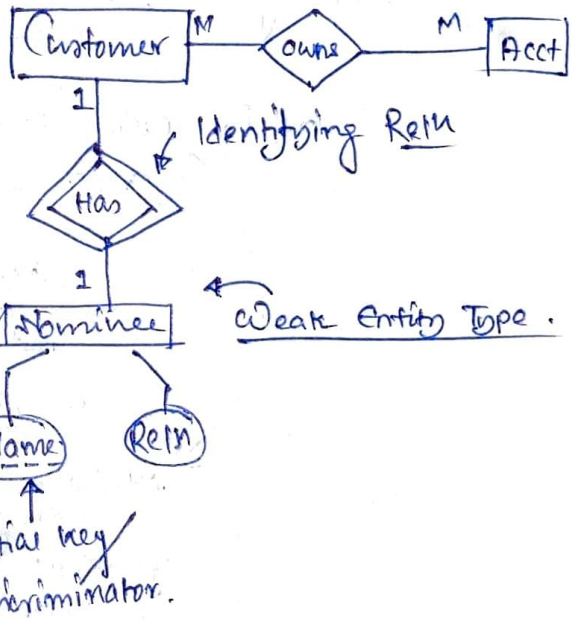
- Mapping Cardinality.
- Participation
  - Total
    - Existence Dependency.
  - Partial.



Entity type

- Strong Entity type
  - ⇓ has its own key
- Weak Entity Type
  - ⇓ do not have its own key

Partial key / Discriminator.



Partial Key

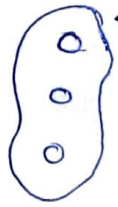
Must participate in the relationship (total) with the entity type on which it depends.

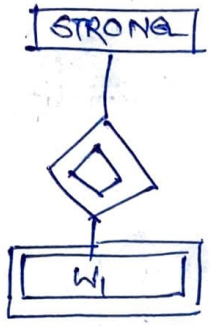
In order to identify an instance of weak entity type, one will have to know the instance of the entity type of which one dependends one that is given traverse.

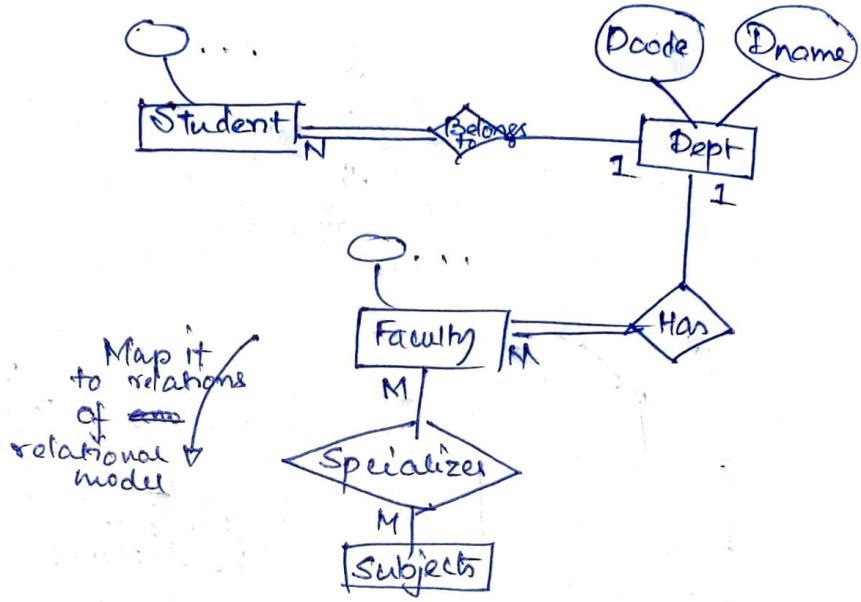
The relation to obtain the related subset from weak entity set we use the partial key to identify specific instance of weak set.

A customer.

CODE → subcee







Mapping entity & Relⁿ of ERD to Relation of Relational Model

File Structure

Table in SQL

- For each strong entity type
  - Create a relation
    - ⇓
    - Attribute of relⁿ same as attribute of entity type.
    - Pk of Relⁿ ⇒ Key attribute of entity type.
  - Attributes must be single valued / atomic.
  - If there is any composite attr. then decompose.

Put multivalued attribute in a seperate Relⁿ.

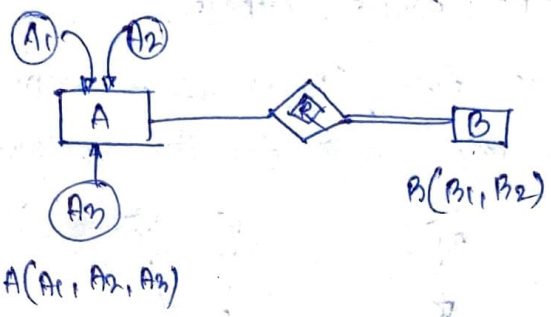
Student (Roll No. , Name ...)

Phone (Student id , PhoneNo)

① Remove m.v. attr. & put in a seperate relⁿ
② Copy the p.k. of original relⁿ to new relⁿ.
③ Pk of new relⁿ ⇒ Pk of original relⁿ ∪ M.V.attr.

In the new relⁿ copied Pk of original relⁿ will be foreign key referring to original relⁿ.

Weak Entity type



Relⁿ schema — p.k. of entity type on which it depends ∪ its own attribute.

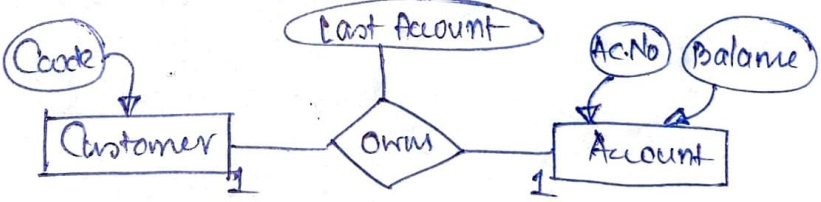
Pk — Pk of entity type on which it depends ∪ partial key.

→ Copied P.K. will be F.K. in weak entity relⁿ.

Binary Relⁿ

→ Mapping Cardinality
→ Participation.

1:1



Customer(Ccode, Cname, ...)
Account(Ac_No, Balance).

Foreign key approach → ① Copy the P.K. of the relⁿ into another & there it will be foreign key.

If one entity type totally participates in the relⁿ then copy the p.k. of other side in this & it will be F.K. here.

② Merged Relⁿ

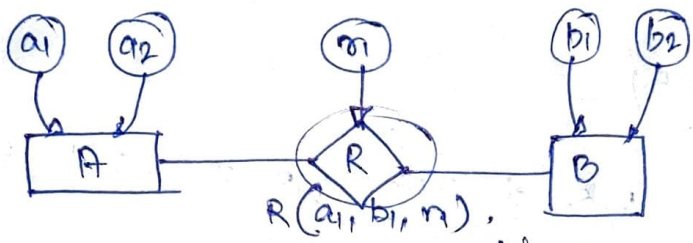
Instead of keeping
→ Both are totally participating instead of
mapping the

In foreign key appraach, attribute of relationship also be copied into the entity type where primary key of other has been copied.

CUSTOMER_ACCOUNT (ECODE, CNAME, ... }.

③ Cross Reference Relⁿ / Relationship Relⁿ

→ Create a separate relⁿ for the relationship in ERD.



R(a1, b1, n1).

→ Both are partially participative.

Primary key of related entity type with attribute

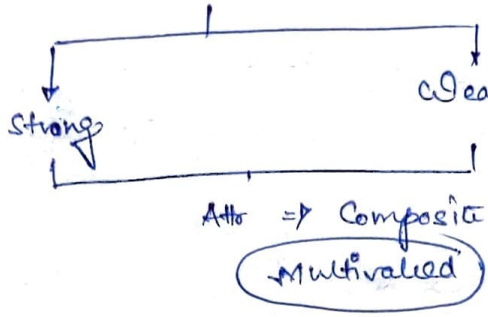
15/3/2021

DBMS

Prof. Sanjay Kumar Saha

ER DIAGRAM = MAPPING to Relations in Relational model.

ENTITY TYPE



Relationship of ERD

⇓ Mapping
CARD
participation
Constr.

1:1

Binary Rel^n.

One to Many / Many to One

Foreign Key Based

In many side copy the primary key of related entity type and it will be a foreign key in many side.

~~Foreign Key~~

Use of Dcode in Student

Rel

Also copy the attribute of relation in Many side.

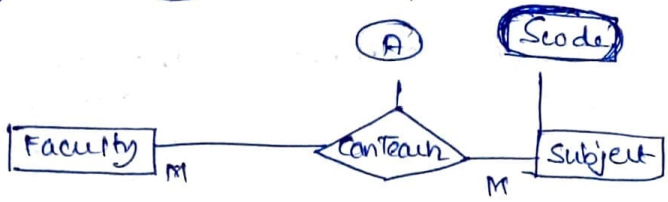
Cross Reference / Relationship Rel^n

Separate relation for Relationship type of ERD

Attr => ⓟPK of related entity types,
Ⓞ Attr. of relationship if any.

Participation of student is partial. In foreign key based approach there will be Null values, but in cross-referencing approach, the row will not be there.

Many to Many Relationship

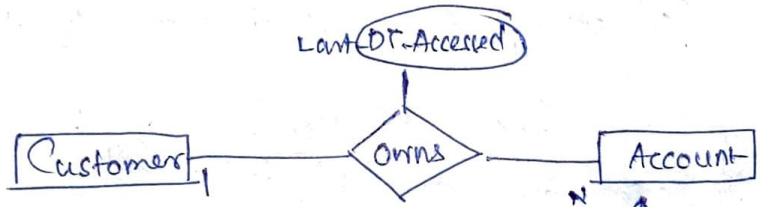


Seperate Relⁿ for Relationship of ERD.

⇓

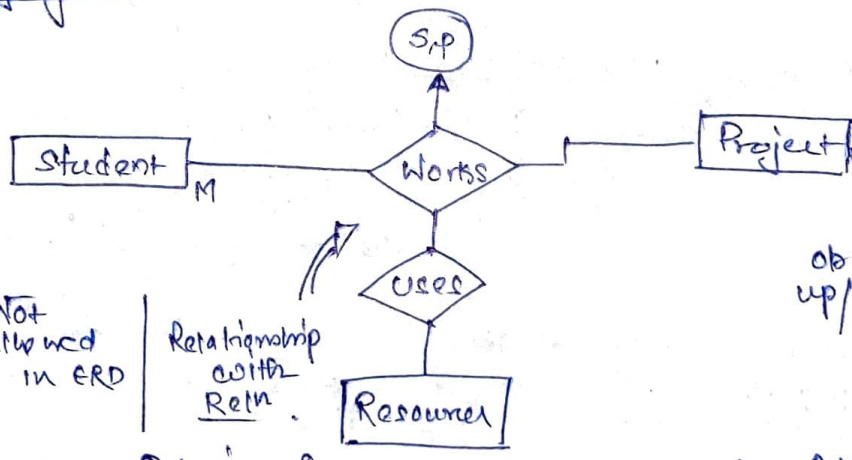
Attr ⇒ U_{PK} of Related entity types.
∪ its own attribute (if any).

F.K → ✗
Merged Relⁿ → ✗
Cross Refⁿ Relⁿ.
Copied PK will also be there.
P.K ⇒ U_{PK} of Related entity types.



Customer Id,
Last_date_Access,
Owns (CustomerId, AC-NO, Last_Dt_Accessed).

Aggregation.



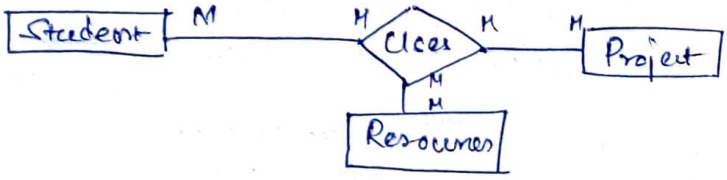
Not allowed in ERD

Relationship with Relⁿ.

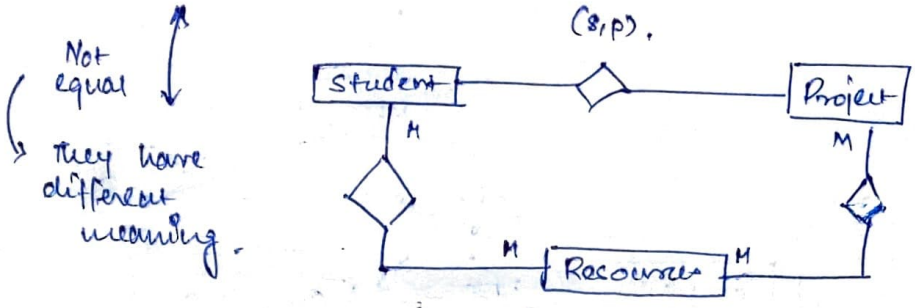
Relationship associated with entity type.

Entity type obtained by summing up/combining other entity types & their associations.

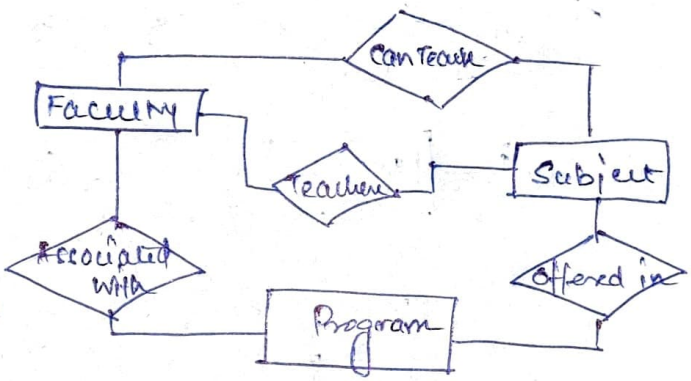
* Relationship b/w aggregated entity type & Resource.



Ternary Relⁿ



Any one binary relation in 1:1, ternary relation can be avoided.



Mapping a ternary relation

↓

A seperate relation for this

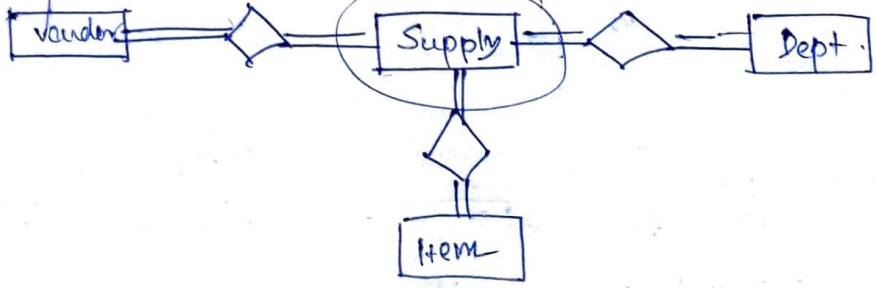
Attr = $\cup$ P.K. of entity types $\cup$ its own attributes

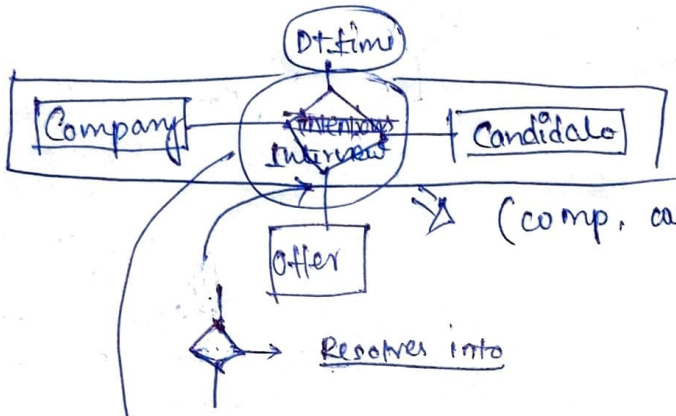
(F.K)

P.K ⇒ $\cup$ P.K of entity types.

~~Given a vendor dept combination~~

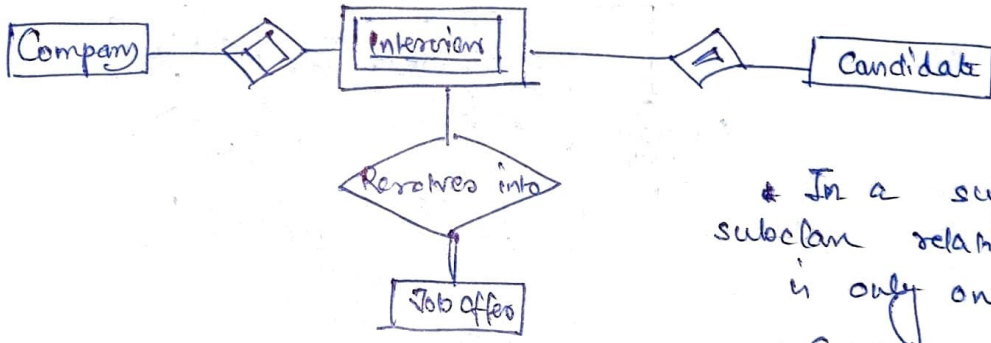
⇓ Union of PK of all entities must be present in all relations.





(comp, candidate, offer).

Converting into weak entity type.



* In a superclass subclass relation ther is only one superclass.

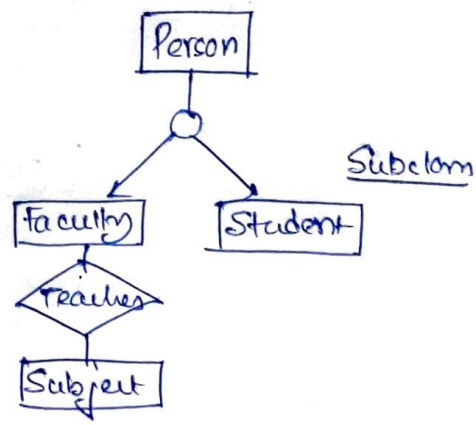
Enhanced ER (EER) Diagram

→ Superclass Subclass Relationship

⇓

* Some attributes may be applicable only for a subset of superclass instance.

* In some relationship only the subclasses may take part



Constraints on superclass-subclass relationship

→ disjointness constraint

↳ Association overlapped in disjoint

Disjoint ⇒ A superclass instance can be a member of atleast one subclass.

Overlapped ⇒ A superclass instance might be a member of multiple subclass instance.

→ Completeness constraint

→ total
→ partial

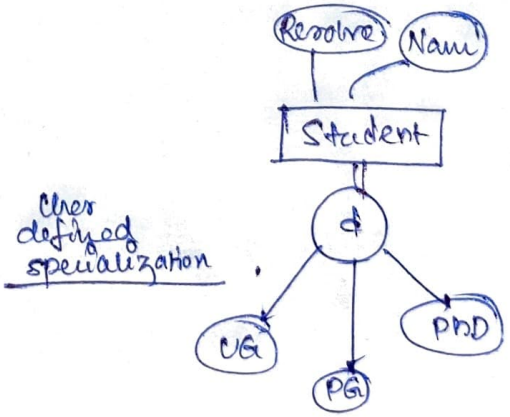
Every superclass instance must belong to atleast one subclass

A superclass instance may not be member of any of the subclass.

A superclass member may be determined as subclass member based on certain condition.

Conditional specification / predicate defined specialization.

based on certain attribute



17/3/2021

DBMS

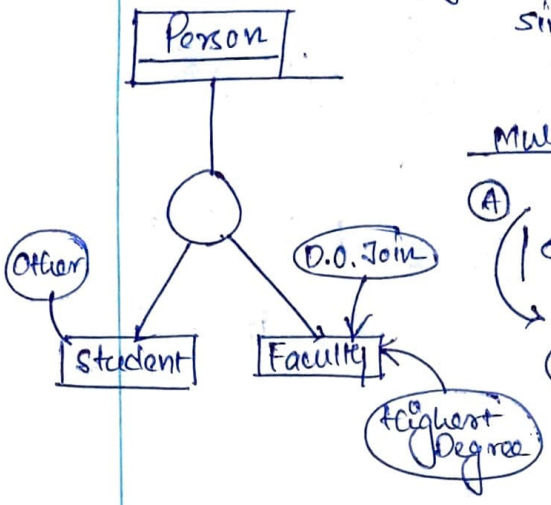
Prof. Sanjay Kumar Saha

① Superclass — Subclass association

⇓

Mapping to relation of relational model.

Single Relⁿ, Multiple Relⁿ



Multiple Relⁿ.

(A) A Relⁿ for superclass.

One relⁿ for each of the sub-classes.

→ Copy of the p.k. of superclass (there will be a foreign key here

* also candidate key/primary key

+ addln attribute

Overlapped/Partial

✓ Model is good enough

~~Total participation~~

No relⁿ for superclass (B) → Disjoint/Total

For each subclass

→ Consider a relation.

Attribute.

Attr. of superclass ∪ Attr. of subclass (it's own attribute)

Student( )

Faculty( )

Total participation ⇒ OK

If partial participation ⇒ LOST

overlapped → Redundancy.

Disjoint → No problem.

(C) One Relⁿ ⇒ Union of attributes of superclass & those of all subclasses.

Type field ⇒ Denote the type of subclass (disjoint specialization).

(D) One relation ⇒ Attribute of superclass ∪ Attribute of all ~~subs~~ subclasses. ∪ No. of type fields. (No of subclasses)

$b_0\ b_1\ b_2 \ldots b_{n-1}$ ⇓ bit position

Type 1 ✓ Type 2 ○ Type 3 ○

$b_j = 1$ means belongs to jth subclass

Multiple { (A) Superclass + Subclasses
(B) Subclasses

Person, Student, Faculty.
X Addln Attribute.

Single attr. { (C)
(D)

For a subclass
⇓
We also want to know corresponding superclass information.

Person ⋈ Student

Person.Person_ID = Student.Person_ID

(A) ⟶ carry out Equijoin
(B) No equijoin needed
(A) ∨ (D)

[All superclass information] ⇒ (A).
⇓
(B)
Outerunion

(C) & (D) ⇒ No outer union is required.

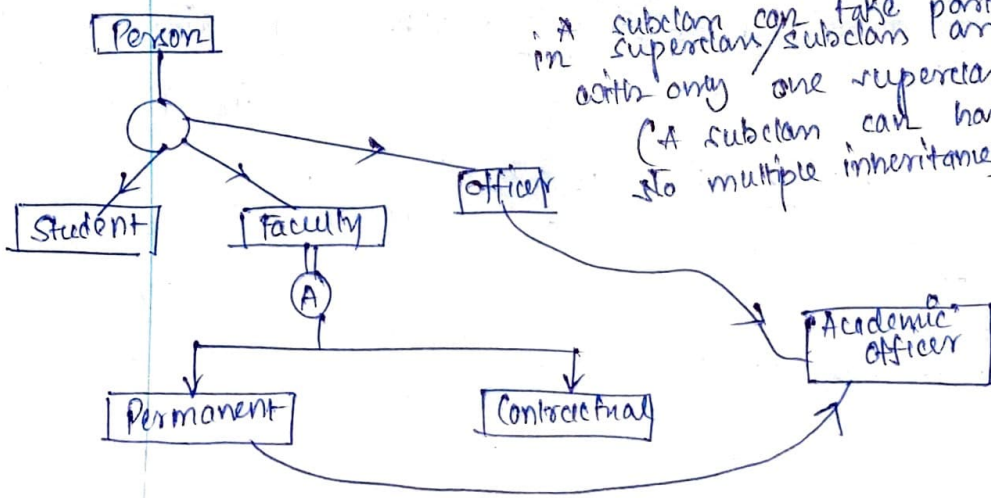
Subclasses are no. of attributes
⇓
A superclass instance may belong to Coco no. of subclasses.
⇒ Cost of more value in imp attributes

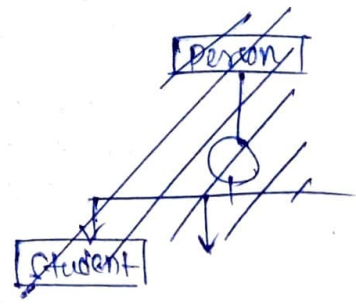
Not a large no. of subclasses & they are not having large no. of attributes
C/D

Specialisation Hierarchy ∨ Lattice

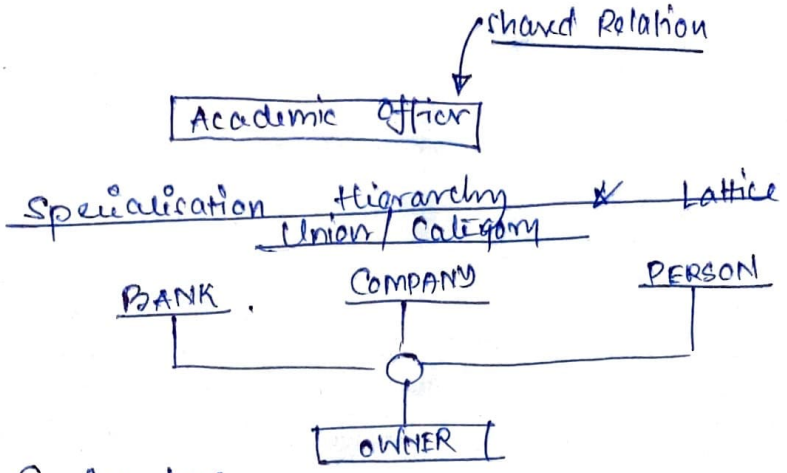
Superclass — Subclass.

A subclass can take part in superclass/subclass association with only one superclass (A subclass can have only one parent / No multiple inheritance).





In a lattice, a subclass can have multiple parents-



Superclasses

Special type of superclass - subclass association.

Subclasses

Multiple superclasses for the subclass

Keys are also different types.

Different type of entities.

Shared class

instances of shared class will be the subset of intersection of the superclasses.

Subclass/Category of union

instances are subset - instances in superclasses

superclass of

~~In a~~ category may not have a common primary key

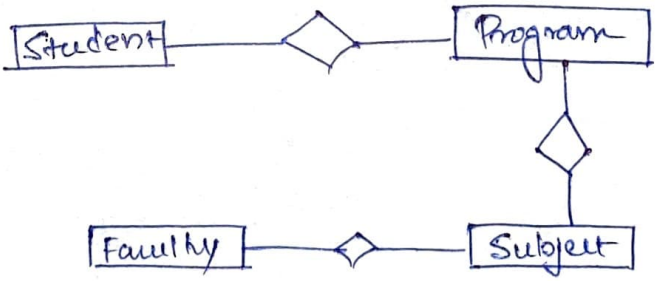
So, a primary key in category has to be introduced

surrogate key

~~so~~ it will be copied to all super - class.

Given a system ⇒ study interact with the users

→ Capture data requirement and their relation.



Structured Query Language (SQL)

→ DDL
→ DML

Statements → DDL
→ DML

Relational Algebra (Proq).
Relational Calculus (Non-procedural).

SQL Statement ⇒ Similarity with Relational Algebra
⇓
Non-procedural

End user programming language.

Programming Language vs SQL

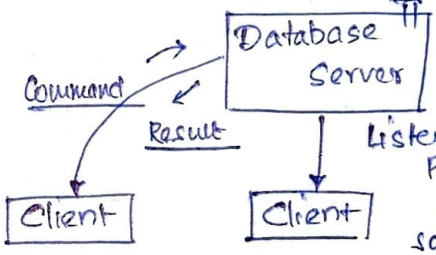
SQL → Oracle
DB2
MySQL

Logically same
Syntax may vary a bit.

SQL Plus ⇓ Editing Environment

Oracle Provides

DataBase



Prompt.

SQL>

Q. Where is the server?
Q. Which service?
→ Encapsulated in the host string.

Data Defn Language

DDL

→ Metadata is stored in a data dictionary

DEPT (DCODE, DNAME)
STUDENT (ROLL, NAME, DOB, SCORE, DCODE)

```
CREATE TABLE DEPT
( DCODE CHAR(5) PRIMARY KEY
  ----
  ----
  DNAME CHAR(10) ~~NOT~~ NOT NULL );
  ....
```

Column description
Column Name type & size Constraint

eg. Number (3,1)
Number (3,0) Number (M,N)
total no. of digits
↳ No. of digits after decimal.

; End of Command

→ If prem an enter ~~after~~ at a blank line it will not Execute but will store into a buffer.

if we writ SQL> RUN / OR → Command in the buffer will be executed.

Pro tip : Use a text editor like Notepad to type in the commands.

SYS_00-50 → Name given

Column Type & Size | Primary Key | Constraint

PK_dept
Constraint Name

```
SQL> CREATE TABLE DEPT
     ( DCODE CHAR(5) PRIMARY KEY CONSTRAINT
       DNAME CHAR(10) NOT NULL )

SQL> CREATE TABLE STUDENT
     ( ROLL NUMBER(3,0) PRIMARY KEY CONSTRAINT PKSTUDEN
       NAME CHAR(20)
       DOB DATE
       SCORE NUMBER (5,2)
       DCODE CHAR(5) REFERENCES DEPT(DCODE)
                     CONSTRAINT FK_STUDENT )

SCORE NUMBER (5,2) CHECK (SCORE >= 0)
```

| PRIMARY KEY
| REFERENCING
| NOT NULL
| UNIQUE
| CHECK (Cond.)

Column Constraint

⇓

Specified for individual column along with its description

RESULT (ROLL, SCODE, SCORE)

Composite Primary Key

```
CREATE TABLE RESULT
( ROLL NUMBER (3,0)
  SCODE CHAR(5)
  SCORE NUMBER (5,2)
  PRIMARY KEY (ROLL, SCODE) )
```

As a whole it must refer

BACKLOG (ROLL, SCODE)

ATTENDANCE (ROLL, SCODE, DT_EMP)

```
CREATE TABLE ATTEDANCE ( ROLL NUMBER(3,0) ,
                         REFERENCES STUDENT (ROLL)
                         SCODE CHAR(5) REFERENCES SUBJECT
                         DT_EXAM DATE .
                         PRIMARY KEY (ROLL, SCODE) )
```

Primary key of Subject

```
CREATE TABLE RESULT
(  ROLL NUMBER(3,0) ,
   SCODE CHAR(4),
   SCORE NUMBER(5,2) ,
→  PRIMARY KEY (ROLL_NO, SCODE) ) .
                                  . . .
```

TO ADD TUPLES/ROWS

DEPT (DCODE, DNAME)

STUDENT (ROLL, NAME, DOB, SCORE, DCODE)

INSERT INTO DEPT VALUES ('D1', 'ABC')

tablename

Value of the columns following their order of specific in CREATE_TABLE)

SQLPLUS COMMAND

SQL> DESC DEPT

NAME	TYPE	NOTNULL
DCODE		
DN	—	— . . .

TO Display DATA

→ List of Expressions

SQL> SELECT * FROM STUDENT ;

Displays all the tuples/rows from STUDENT .

Col. Names

```
SELECT ROLL, NAME
FROM STUDENT;

SELECT ROLL, SCORE+5
FROM STUDENT;
```

SELECT 'ABC' FROM STUDENT ;

SQL > SELECT *
FROM STUDENT
WHERE DCODE = 'D1' ;

takes a table
checks the where clause if satisfied
⇓
take action

Condition

AND
OR
NOT

>
<
>=
<=
==
!=

SELECT *
FROM STUDENT
WHERE SCORE IS NULL

To Compute with NULL
-> IS NULL
-> IS NOT NULL

SELECT *
FROM STUDENT
WHERE DCODE IN ('D1', 'D5', 'D7')

DCODE = 'D1' OR DCODE = 'D5' OR DCODE = 'D7'

WHERE SCORE BETWEEN 70 AND 90
SCORE >= 70 AND SCORE <= 90.

SCORE >= ANY (50, 65, 80)
List of values

>= ALL (50, 65, 80)

any relational operator

SQL > SELECT AVG(SCORE)
FROM STUDENT

BUILT IN FUNCTIONS

Non-Aggregate FN
⇓
Acts on non-tuple
& for every tuple gives in output

AGGREGATE FN
⇓
Acts on a collection of tuple/rows
For each connection gives one output

Aggregate →
- SUM( ), COUNT(*), COUNT(DISTINCT COL.)
  - COUNT(*) → Count of all tuples.
- COUNT(DISTINCT DCODE)
- MAX() AND MIN()
- VARIANCE(), STDDEV()

Non-Aggregate →
- UPPER()
- LOWER()

WHERE DCODE = 'D1'
UPPER(DCODE) = 'D1'

INIT CAP
⇓
Multiword string
1st character of each word in UPPERCASE
+ Rest in Lowercase.

SUBSTR(String, n, m)
- n → from form in 1st parameter ⇒ 1.
- m → No. of character to extract

INSTR(String1, String2, n, n)
- finding string2 in string2
- start position.
- nth pos. of string2.

NUMERIC

ROUND(value, n)

ROUND(172.752)

ROUND(172.752, 1)
⇓
172.8

ROUND(172.452, 0)
⇓
172

ROUND(172.452, -1)
⇒ 170.
177.452
⇒ 180.

TRUNCATE(Value, No. of places)

DATE FN

MONTHS_BETWEEN(d+1, d+2)
⇓
Real value.

ADD_MONTHS(d+).

Numeric String
To Number
⇓
TO_CHAR(Numeric value)

STUDENT (ROLL, NAME, DOB, SCORE, DCODE)

INSERT INTO STUDENT VALUES

(1, 'ABC', '01-Jan-2001', 95, '01'),

Char to be converted to Date

SYSDATE → Oracle → System Date

CEIL(17.5) = 18.

SELECT CEIL(17.1)
FROM DUAL ;

TO_DATE ('01-JAN-2001', 'DD-MM-YYYY')

from .

DD → Day of Month
MM → Month
MON → 3 character Month Name
MONTH → full month Name
YYYY → 4 digit Year

SQL> UPDATE STUDENT
SET SCORE = SCORE + 5
WHERE DEPCODE = 'D5' ;

SQL> DROP TABLE TABLENAME

Syllabus → Till SELECT, CREATE DDL statement.

SQL

SUBJECT (SCODE, _ , FM, PM)
STUDENT (ROLL, NAME, DOB)
RESULT (ROLL, SCODE , SCORE)

SQL> SELECT *
FROM STUDENT
WHERE DOB >= '01-Jan-2000'
AND DOB <= '31-Dec-2000'

DOB >= TO_DATE ('01-Jan-2000', 'DD-MM-YYYY')
AND DOB >= ____ .

SELECT *
FROM STUDENT
WHERE DOB >= '01-Jan-2000'

⇓

where clause is checked with every tuple.

Cartesian Product/Join.

SQL> SELECT FROM RESULT, Student => Cartesian Product/ Join.

Students born in 2000

JOIN CAN BE AVOIDED

```
SELECT ROLL, SCORE
FROM RESULT R, SUBJECT S
WHERE R.SCODE = S.SCODE AND SCORE < PM .
```

~~SELF JOIN~~ SELF JOIN .

(Please check slide examples).

SQL > SELECT *
FROM Result ——— WHERE SCORE >= 50
ORDER BY SCODE, ROLL/SCORE DESC.

SCODE ASC, SCORE DESC.

To group by use GROUP BY ROLL.
ORDER BY SUM(SCORE) DESC
HAVING SUM(BASIC) >= 400
↓
Cannot be used without groupby.

→ FROM
→ WHERE
→ GROUP BY
→ HAVING
→ ORDER BY

ROLL, NAME, TOTAL SCORE ...

```
SELECT   STUDENT.ROLL, NAME,
                       SUM(SCORE)
FROM     STUDENT, RESULT
WHERE    STUDENT.ROLL = RESULT.ROLL
GROUP BY STUDENT.ROLL, NAME.
```

SQL

Simple Subquery.

Store the student who have scored >= 70 in subject named OOPS.

```
SELECT *
  FROM RESULT
WHERE   SCORE >= 70  AND  SCODE =
        [ ( SELECT SCODE
              FROM SUBJECT
              WHERE UPPER(INDEX) = 'OOPS' ).
```

↑

* Inner query is executed only one, its evaluated & its result is used.

* Inner query must return a single value.

* If inner query returns no value / multiple values then special measures have to be taken.

```
SELECT NAME
  FROM STUDENTS
    WHERE UPPER(CITY) =
      ( SELECT UPPER(CITY) FROM STUDENT WHERE ROLL=10 )
```

Find the subjects in which no student has scored >= 80.

```
SQL> SELECT *
     FROM SUBJECT
     WHERE SCODE NOT IN
          ( SELECT SCODE
            FROM RESULT
            WHERE SCORE >= 80 )
```

Max score in SCODE S1

```
SQL> SELECT SCORE
     FROM RESULT
     WHERE SCODE='S1'
     AND SCODE >= ALL
        (SELECT SCODE
           FROM RESULT
             WHERE SCODE='S1').
```

Example HW
→ Roll of the students who have not scored minimum score

```
SELECT   SCODE
FROM     SUBJECT
WHERE    SCORE IN
   ( SELECT DISTINCT SCORE
       FROM ...
       ...)
```

→ Simple subquery correlated subquery where the table in outer query is refered in inner query.

Correlated subquery.

(Please go through slide examples).

DBMS

Prof. Sanjoy Kumar Saha

SQL

Correlated Subquery.

Find the students & subject who have scored more than avg. in the corresponding subject.

```
SELECT   ROLL, SCODE
FROM     RESULT R1
WHEE     SCORE >
   ( SELECT  AVG(SCORE)
     FROM    RESULT R2
     WHERE   R2.SCODE = R1.SCODE )
```

→ Take a tuple from the table in outer query
· Candidate tuple.

For each candidate tuple
→ Execute inner query.
→ Use the outcome to decide the action on and tuple.

Inner query is executed no. of times

Correlated query | Outer Query & Inner Query

Same relⁿ
⇒ Self correlation.

Find the subject where nobody has failed (among those who had appeared)

```
SELECT   SCODE, SNAME
FROM     SUBJECT
WHERE    NOT EXISTS
(     SELECT *
      FROM   RESULT
      WHERE  SCORE != NULL
             AND SCORE < SUBJECT.DM)
```

DDL Statement

→ Create TABLE
→ Drop TABLE ____

TO MODIFY THE SCHEMA

SQL> ALTER TABLE tablename

[ADD (Col, datatype,
        Col, datatype, ...)

[MODIFY (col datatype, col datatype)
  Existing columns.

⇒ if certain tuples are already present the constraint not null cannot be used.

Existing Column

→ Increasing the size ✓
→ Decreasing the size ✓

⇓

Change of type
Avoided only if there are no. of tuples.

DROP constraint ConstraintName.

Create Table Tablename.

⇓

As query will have the schema same as that of the query output & it will contain the tuples produced by the query.

Result (Roll, Scode, Score)

① Absent (Roll, Scode)
② Create Table absent (X, Y)
~~As~~
~~Select Roll, Scode~~
~~From~~

Backlog (Roll, Scode)

Score is Null
& Score < PM

Insert into Backlog Values (Select Roll, Scode From Result where score is null or Score < (Select PM from Subject where S.Scode = R.Scode))

View

Student (Roll, Name, Dcode, ...) | for all students

HOD CSE ⇒ Students of CSE
HOD civil ⇒ " " Civil

Create View CSE_STUDENT
As
SELECT * FROM student
WHERE DCODE = 'CSE'

→ Once a view is created the defn is stored into data dictionary

→ Its not a physically separate tal

Single Table Based View

Affects the physical table

→ Select
→ Insert ⇒ If schema of view is subset of physical table for attribute not in view ⇒ Null / Default values are considered.
→ Modify
→ Delete

May lead to constraint violation

```
CREATE VIEW CSE_STUDENT
AS
SELECT *
FROM STUDENT
WHERE DCODE = ____ .
WITH CHECK OPTION CHECKNAME .
```

ORACLE SPECIFIC .

⇓

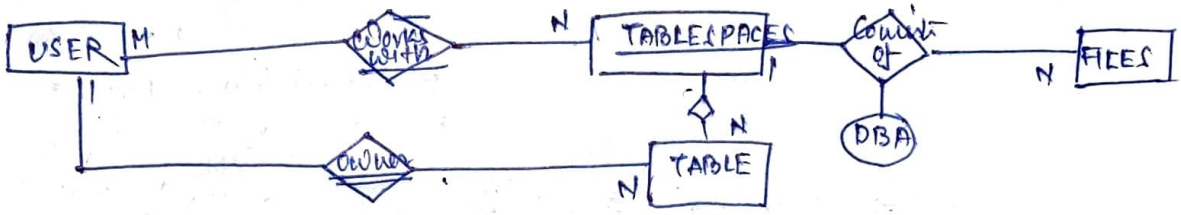
DATABASE ⟶ LOGICAL .

PARTITIONED INTO NO. OF TABLESPACES

⇓

Contains the tables, indexes .

Create user account ⇓ default TS also can spend resources



CREATE TABESPACES

⇓

FILE NAME

SYSTEM ⇓ holds data dict

[DATABASE Design .]

Functional Dependency / Normalization .

⇓

Formal way to analyse the database design based on key & functional dependency .

⇒ Different levels of normal form .

Database satisfies a particular normal form

Attributes are grouped in a relational schema .

Certain meaning is associated with the attribute & there exists an interpretation & based on this attribute are grouped .

Semantics of attribute

Intirated based on meaning & interpretation .

STUDENT (ROLL NAME, DOB, E-MAIL, PHONE, DCODE, ..., BUDGET)

⇓

Combined everything.

① A schema should correspond to one type of entity/relationship.

In a Dept. ⇒ No. of student are there every student, same dept info. is dept, so, Redundancy.

UPDATE Anomalies

→ Insert Anomaly → A new dept has been introduced into student is there cannot be stored, → Primary key roll is NULL, * student has come. Dept. info is not known. lots of null values to be allowed.

→ Deletion Anomaly. → Suppose A student tuple is to be deleted & he/she is the only student in the dept, then dept. info is also lost.

→ Modification Anomaly. → Suppose, HOD of a dept. is changed multiple student tuples for the dept are to be modified the inconsistency.

② Relationship design must avoid Redundancy & update anomalies

③ Avoid atleast frequent occuring of null values.

STUDENT (ROLL, NAME, DOB, PH, E-MAIL, DCODE, HOSTEL-ID, HOSTEL-NAME, ...).

⇓

Not applicable

STUDENT (ROLL, ..., YEAR-EST)

DEPT (DCODE, YEAR-EST, DNAME, ...)

EQUIJOIN BASED IN

⇓

May Generate unwanted tuples. ⇒ Spurious tuples

(3) Design must be such that it is possible to go for equijoin based on primary key & foreign keys [to avoid spurious tuples].

~~Univsal~~ Universal Relation ⇒ Single relation with all attr.

Let R be a schema X & Y are two subsets of attributes.

$X \subset R$ & $Y \subset R$.

A functional dependency $X \longrightarrow Y$ is a constraint on two set of attributes. F.D. specified on R means all tuples in $r(R)$ must satisfy it.

for any two tuples,

$t_1, t_2 \in r(R)$.

if $t_1[X] = t_2[X]$

then $t_1[Y] = t_2[Y]$

$\frac{R}{r(R)}$ ⇒ A set of F.D. specified on Score → Grade.

Score → Grade

70 → B

70 → A.

- if $X \to Y$ holds on $r(R)$ specified on R does not mean $Y \to X$.
- if X is a (Cand Key).
  then $X \to R$
  if ($t_1[X] = t_2[X]$
  then $t_1[R] = t_2[R]$).

$r(R)$

Roll	Name
1	XYZ
2	ABC
⋮	⋮
7	XYZ

Prof. Sanjay Kumar Saha

Functional Dependency

$X \rightarrow Y$

Given :- A F.D. SET F specified on Relation R
From this some other F.D. can be inferred — Reference Rule.

Armstrong's Axioms

IR1 Reflexivity Rule $X \rightarrow Y$ if $Y \subset X$ ← AKA. ~~Transitive~~ Trivial fn. dependency.

IR2. Augmentation Rule. if $X \rightarrow Y$ then $WX \rightarrow WY$.

IR3. Transitive Rule if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

IR4. Decomposition Rule if $X \rightarrow YZ$ then $X \rightarrow Y$

IR5. Union Rule if $X \rightarrow Y$ & $X \rightarrow Z$ then $X \rightarrow YZ$.

IR6 Pseudo-transitive Rule

if $X \rightarrow Y$ AND $WY \rightarrow Z$ then $WX \rightarrow Z$.

Proof.

IR1 $t_1, t_2 \in r(R)$.

if $t_1[X] = t_2[X]$ then $t_1[Y]$ must be same as $t_2[Y]$.

As $t_1[X] = t_2[X]$ AND $Y \subset X$

then $t_1[Y] = t_2[Y]$.

IR2 $X \rightarrow Y$ Holds, $\begin{cases} t_1[X] = t_2[X] & (A) \\ t_1[Y] = t_2[Y] & (D) \end{cases}$

Say $t_1[WX] = t_2[WX]$, now We have to show that $t_1[WY] = t_2[WY]$

Let $WX \rightarrow WY$ does not hold

$\Downarrow$

$t_1[WX] = t_2[WX]$ (B)

But $t_1[WY] \neq t_2[WY]$

(A) & (B) $\Rightarrow$ $t_1[W] = t_2[W]$ (C)

(C) & (D) $\Rightarrow$ $t_1[WY] = t_2[WY]$

IR3

$X \to Y$
$Y \to Z$ | Holds.

$t_1[X] = t_2[X]$ (A) $t_1[Y] = t_2[Y]$

$t_1[X] = t_2[Z]$ (B)

(A) & (B) $\Rightarrow$ $X \to Z$

IR4

$X \to YZ$
$YZ \to Y$ (IR1)

IR3

$X \to Y$

IR5

$X \to Y$
$X \to Z$ $\Rightarrow$ IR2.
$XY \to YZ$

X

$X \to XY$ (IR2) — IR3.

$X \to XY$
$XY \to YZ$) IR3.
$X \to YZ$

IR6

$X \to Y$
$WY \to Z$

IR1

$WX \to WY$

$WX \to WY$
$WY \to Z$ | IR3.

Set of F.D.s F specified on R

F+ is closure of F

⇓

Set of all FDs.
that includes F and F.D.s that
can be infered from F.

Armstrong's Axioms

Sound & Complete

Fn dependency set is specified on relation R armstrong's axioms that holds on R any F.D. that is infered from F using the

Any valid tuple satisfies the fn dependency.

By successive appln of armstrong's axioms on F a complete set of FDs can be infered.

F

$X \to Y$ $Y \to Z$

F+

$X \to Y$
$Y \to Z$
$X \to Y$
$X \to Z$
$Y \to Z$

Relⁿ R

F.D. set F } Given.

X^+ Closure of set of attr. X

Set of all attribute that can be determined by X.

$X \to X$ (X)

repeat { old $X^+ = X$

For each functional dependency $A \to B$ in F

{ if $A \subseteq X^+$ then $X^+ = X^+ \cup \{B\}$

} until (old $X^+ = X^+$).

$X \to Z \to$ FD1
$X \to Y$ FD2

$X^+ = (X)$
$Y \subseteq X^+ (X)$
$X^+ = XY$

$X^+ = X$

old $X^+ = X$

$X^+ = XY$

old $X^+ = XY$

$X^+ = XYZ$

old $X^+ = XYZ$

$X^+ = XYZ$

if $(X^+ = R)$

$\Downarrow$

X is a candidate key.

can be.

$R(A_1, A_2, \ldots, A_n)$.

A_1^+

A_2^+

$\vdots$

A_n^+

Given two F.D. set E & F specified on R.

F covers E,

if each F.D. in E is present in F^+

F

$X \to Y$

$X \to Z$

F'

$X \to Y$

$Y \to Z$

$X \to Z$

E

$X \to Y$

$X \to Z$

→ Cannot infer anything else. Reverse is not true.

$E \subseteq F$

Two F.D. sets are equivalent if F covers E and vice-versa.

$\equiv$ $F^+ = E^+$

Minimal cover.

→ Replace each F.D. of the form

$X \to A_2, A_2, \ldots, A_n$

with set of FDs.

$X \to A_1, X \to A_2, \ldots, X \to A_n$

Set in F

↓

Fmin : For each FD $X \to Y$

if $(\{F - \{X \to Y\}\} \cup \{(X - \{B\}) \to Y\})$

Name, Roll

↓

Score

B C X

- For each functional dependency $X \to Y$ in F
  If $F \to \{X \to Y\}$ is equivalent to F
  then $F = F - \{X \to Y\}$.

  → Any reduadant F.D. ?

Normalization

⇓

Formal way of analysing the schema subjected to Key & Functional dependencies

Key ⇓ Cand. key

Functional dependencies ⇓ Domain Knowledge

It is a process of refining a complex/FAT schema into simpler ones such that →

- Redundancy is avoided
- Updation anomaly is avoided.
- Occurence of null value is restricted.
- Dependency preserving

Prime attribute of a relation

Attribute of which is part of any candidate key.

STUDENT(ROLL, REG.NO, ...)

Result(...).

→ Rest are non-prime attr.

A relⁿ is in a normal form satisfies 1st, 2nd, ..., Knf.

1 NF
2 NF
3 NF

Always highest level of normalisation may not be done

Certain redundancies may be allowed by not going for higher form.

→ **Denormalisation**

05/04/2021

DBMS

Prof. Sanjay Kumar Saha

①

NORMALIZATION ⇒ F.D. set is Given (Minimal Cover).

SCHEMA

For every student, need to store information.
Every student has unique roll ⇒ Roll is the p.k.

→ Roll No.
→ Name → Phone No.
→ City → {SCODE, SNAME, ... }
→ DCODE
→ DNAME
→ Y_EST

1st Normal Form

→ Every attribute in the schema must be atomic & single valued.

If we have Non-atomic attributes (Atomic/Non-atomic) ⇒ Requirement specific.
↳ Replace it by the components

If there is multi-valued attribute ⇒ Remove these attributes from original schema & put into a new schema. Also copy the p.k. in new schema.

↓

M.V. attribute that repeat together & related with each other, group them & remove them from original schema to form a new one. Copy p.k. of original relation.

Student
Roll (P.K.)
Name
City
Phone No.
{Scode, ... }
Dcode
. . . .

↓

1NF.

Student
Roll (P.K.).
Name
City
Ph.No.
Dcode
Dname
YR_EST
OFFICE PH.

ER

Result
Roll
Scodl
Sname
FH
PH
Score

DPHONE
ROLL
PH_NO

2NF

Result
Roll
Scode
Score

Subject
→ Sname
→ FM
→ PM

ROLL → NAME, CITY, PH-NO, DCODE
DCODE → DNAME, YR-EST, OFF-PH
SCODE → SNAME, FN, PM
ROLL, SCODE → SCORE

2ND NORMAL FORM (2NF)

A schema is in 2NF if every non-prime attribute is fully dependent on Primary key.

Fully dependent ⇒ Functional dependency.

Set of attributes

Full/Partial dependency

$X \longrightarrow Y$

Full functional dependency

$X - \{A\} \longrightarrow Y$ is not possible

$A \subset X$

$X - \{A\} \longrightarrow Y$ is possible

then Y depends on X partially.

X is simple

If P.K. is atomic ⇒ that schema is in 2NF.

X

Roll, Scode → (Name, FN, PM, Score)

A = Roll

$X - \{A\}$.

Violation for 2NF

X is p.K.
$A \subset X$
$X - \{A\} \rightarrow Y$
Y: Non-prime

→ Remove Y from original schema & put them into new Rel^n.

Copy $X - \{A\}$ (Part on which Y depends) in new schema

SCODE ⇒ SNAME, FN, PM

In original schema
$X - \{A\}$ (Scode) will be a F.K. referring to new schema.

~~Remove~~

3NF (3rd Normal Form)

→ To remove transitive dependency

Schema. R

$X \to Y$ holds on R

Suppose $X \to Z$ & $Z \to Y$ holds on R.

& Z is not a candidate key & Non-prime.

Say, a non-trivial F.D. $X \to Y$ violates

=> X is not a S.K. & Y is non-prime.

$X \to Y$ violates 3NF $\Rightarrow$ (X is S.K) OR (Y is prime).

= (X is not a S.K) AND (Y is non prime)

= (X is part of S.K.) OR (X is non-prime)

AND (Y is non-prime).

Violation of 2NF

X is part of candidate key & Y is non-prime

$X \to Y$

X: Non-prime, Y: .

If $X \to Y$ violates 3NF then remove Y from original schema & put in a new one. Copy X into new schema & it will P.K. in original, it will be F.K.

New schema (DEPT).

→ DNAME
→ YR-EST
→ OFF_PH.
→ DCODE

STUDENT
...
DCODE — F.K.

→ STUDENT (ROLL, NAME, CITY, PH-NO, DCODE)
→ DEPT (DCODE, DEPT, YR_ET, ...)
→ SUBJECT (SCODE, SNAME, FM, PM).
→ RESULT (ROLL, SCODE, SCORE).

1NF
2NF
TRANS DEP. 3NF. BCNF.

(ROLL, SCODE, FACULTY_ID)

ROLL, SCODE → FACULTY_ID
ROLL, FACULTY_ID → SCODE

X → Y
X is S.K.
OR
Y is prime attribute

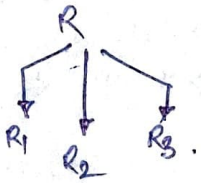
R(A,B,C)
A C is Key
AB is Key
C → B.

BCNF (stricter than 3NF).

BCNF
⇓
A schema is in BCNF
If for every X → Y trivial F.D. that holds on the schema X is a S.K.

X → Y violation
Remove Y, put in new schema.
Copy X, X is P.K.
& F.K in OLA.

- To remove redundancy, minize the occurrence values.
- Remove update anomaly.



R(A1, A2, ..., An)
= R1 ∪ R2 ∪ ... Rn.

→ Decomposition must be loss-less.

Natural join of $\sigma_i(R_i)$ = nat. join of $\pi_{R_i}(\sigma(R_i))$

$\sigma_1, \sigma_2, \ldots, \sigma_n = \pi_{R_i}(\sigma(R)) \cdot \pi_{R_2}(\sigma(R)) \cdot \ldots \cdot \pi_{R_k}(\sigma(R))$

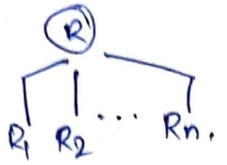
Either $R_1 \cap R_2 \longrightarrow R_1$.
OR
$R_1 \cap R_2 \longrightarrow R_2$.

Decomposition should be
→ loss-less
→ Dependency preserving

$E = \cup E_i$
if $E^+ = F^+$

F.D. let F on R

Say. E_i be the set of F.D. on R_i



for any F.d. $X \to Y \in E_i$
$X \cup Y \subseteq R_i$

$(X \cup Y) \subset R_i$

E_i is the restriction of F on R_i

$X \to A_1, A_2, \ldots, A_n$
$Y \to B_1, B_2 \ldots B_n.$

3NF

- Removes Redundancy
- Loss-less decomposition
- Dependency preserving.

BCNF

- Removes Redundancy.
- Loss-less decomposition
- May not be dep. pres.

06/04/2021 . DBMS . Prof. Sangoy Kumar Saha

NORMALIZATION .

1NF
↓
2NF
↓
3NF BCNF

→ Loss less Decomposition
→ Dep. pres. Decomposition.

Decomposition .

ECODE ENAME BASIC PHONE DP. IN

DCODE DNAME
↓
DEPT. PH

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

DEPT .

DCODE	DNAME
D1	.
D2	.
D3	.
⋮	.

EMPLOYEE .

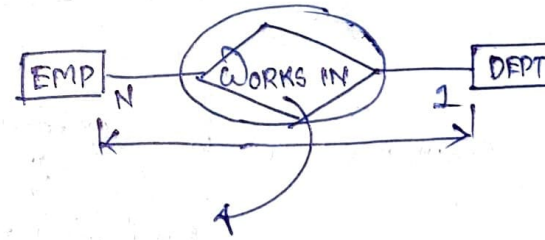
ECODE	ENAME	....	DEPT
E1			D1
E2			D2
E3			⋮
.			.
.			.
E19			NULL

~~EMP ⋈~~

EMP OR DEPT

EMP ⋈ DEPT
(EMP.DCODE = DEPT.DCODE).

Decomposition .
⇓
Be aware of the null values could joining.



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

WORKS IN (ECODE, DCODE).

WORKS IN

ECODE	DCODE
E_1	D_1
E_2	D_2
E_3	D_3
⋮	⋮
	NULL
E_{19}	⋮
⋮	
E_{50}	NULL

If DCODE is not known, No entry can be made.

EMP ⋈ WORKS_IN

⇓

EMPLOYEE TUPLES NOT IN WORKS IN CALL NOT APPEAR.

(EMP ——— DCODE)
NULL

(EMP NOT WORKS_IN)

⇓

Join not Allowed.

EMP(ECODE, ENAME, ... , DCODE)

⇓

All employees

EMP(ECODE, ENAME, ...)
WORKS_IN(ECODE, DCODE)

EMP ⋈ WORKS IN — NO NULL IS ALLOWED

Decomposition is extrinically made.

⇓

Equi join (inner) may not give all tuples.

Multi°-valued dependency

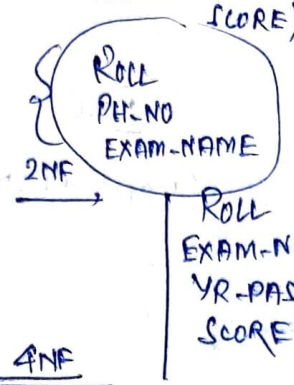
STUDENT (ROLL, NAME, DOB, {PHONE NO}, {EXAM NAME, YR_PASS, SCORE}).

⇓ INF.

1. ROLL
2. PHONE_NO
3. EXAM NAME

t_1	...	1	P1	10th	... YR_PASS, SCORE
t_2	...	1	P1	12th	
t_3	...	1	P2	10th	
t_4	...	1	P2	12th	

2NF →



4NF ⇓

Fallout of INF

$X \longrightarrow\!\!> Y$

X multi°-determines Y

MVD (Multi°valued dependency)
ON SCHEMA R $\quad \alpha \longrightarrow\!\!> \beta$

SAY, $Z = R - \{\alpha \cup \beta\}$ HOLDS.

$\alpha \longrightarrow\!\!> \beta$ means
if there exists two tuples
t_1 & $t_2 \in r(R)$ with $t_1[\alpha] = t_2[\alpha]$
then there exists two other tuples t_3 & t_4 such that :.

$\rightarrow t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$

$\rightarrow t_1[\beta] = t_2[\beta]$ AND $t_3[\beta] = t_4[\beta]$

$\rightarrow t_1[Z] = t_3[Z]$
AND $t_2[Z] = t_4[Z]$

If a MVD $\alpha \longrightarrow\!\!> \beta$ holds on R
then if it is not a trivial dependency,
then R is not in 4TH normal form.

$\alpha \longrightarrow\!\!> \beta$ violates

F.D.
$A \rightarrow B$.
BCA.
↑
Trivial

MVD $\quad \alpha \longrightarrow\!\!> \beta$ is trivial if
β is a subset of α OR
$\alpha \cup \beta = R$

ROLL $\longrightarrow\!\!>$ PHONE NO.

$X, Y \subset R$
$Z = R - \{X \cup Y\}$
$X \longrightarrow\!\!> Y$
$X \longrightarrow\!\!> Z$ | $X \longrightarrow\!\!> Y|Z$

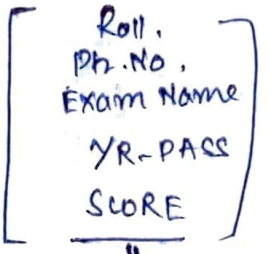
[Remove Y in a seperate relation and copy X.

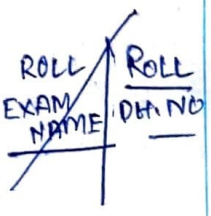
ROLL PH-NO

(ROLL PH-NO)

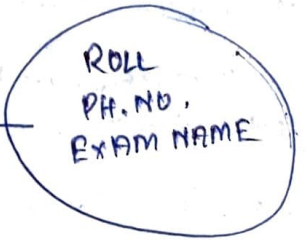
PH-NO

(ROLL. EXAM-NAME)





4NF



ROLL
EXAM-NAME
YR-PASS
SCORE.

22/04/2021

FILE - ORGANIZATION

Prof. Sanjoy Kumar Saha

Disk ⇒ File

File ⇒ Collection of Records.

Record ⇒ Collection of related values/fields. - value/data.

Records

⇓

Variable length why?

File

Holding diff. types of records in the same field STUDENT Record

Dept. Record diff types of record.

Holding records of single type.

Some fields are of varying length (actual length of data varies).

→ Wastage of Space

Force max. length for such fields.

Some fields which are not applicable to all.

Suppose student

if applicable for the

- Hostel No
- Location

make those mandatory and allow special values say NULL.

→ Some fields are multivalued.

→ Allow max no. of values for all.

Management of fixed length record is easier

→ For each record certain space is reserved, so editing within size limit is not a problem.

→ ~~For each~~

→ Variable length record ⇒ New length > old length.

— < old length.

Add in length!

Variable length Record

→ End of Record.
Record terminator symbol.

Record Terminator

→ Large no. of attributes but most are optional.
<tuplename, value> pairs.

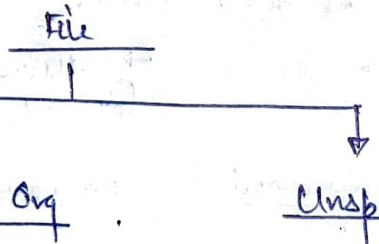
→ FieldName seperator (#)

↓

seperator cannot happen as Actual data.

File
⇓
Σ Records.

Physically.
File ⇒ Σ disk blocks.



→ A record can b/c cross block boundary. Can span over multiple boundaries.

→ ~~A record cannot span over multi~~

bfr ⇒ Blocking factor.

Fixed length record ⇒ Block size B Bytes.
Record size R Bytes.

Unspanned Org.

$$bfr = \lfloor B/R \rfloor \quad \lfloor 512/100 \rfloor$$

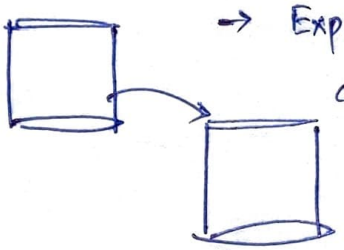
Say there are r no. of records stored in b no. of blocks

$$b = \lceil r/bfr \rceil$$

Alloc. of disk block to a file.

Closed alloc. → Continuous alloc.

Blocks are allocated to a file in contiguous manner.



→ Expansion is easier but already all the records will be shifted.

Advantage: Reading all the records will be faster for all the blocks in a track ⇒ [Only one seek time + latency for the first block.]

→ Clustered Allocation.

Indexed alloc.
↳ for the file an index holding the pointers to the data block is maintained.

File
|
- Unordered File
  Records are not stored in an ordered order as to any particular field.
- Ordered File
  → Records are stored maintaining an order on certain field, (ordering field). If ordering field is a key field ⇒ Ordering Key.

Unordered

→ Insertion of record is easy.

→ Searching : Linear search b no. of blocks in worst ~~case~~ case, ~~the~~ all the b blocks are to be read. On an avg. b/2 block access.

Ordered

→ Insertion is difficult. Find the position

→ Searching : • Based on ordering field. Binary search. $\lceil \log_2 b \rceil$ ⇐ no. of block access.

→ We want to access all record acc. to the same order.

Order ⇒ Easier.

→ If search is based on criteria other than ordering field ⇒ linear search.

Editing the record

Read it
Modify it
Write.

Deleting a record.

⇒ Reuse the space.

Logical Deletion

⇒ check a flag set it if deleted while accessing, check the flag before working with it.

Periodically ⇒ go for physical deletion.

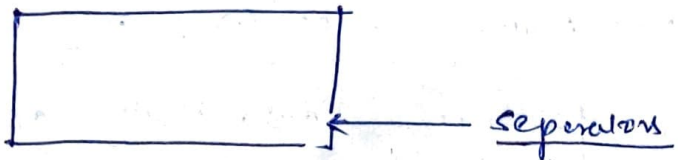
Count exceeds a limit

OR

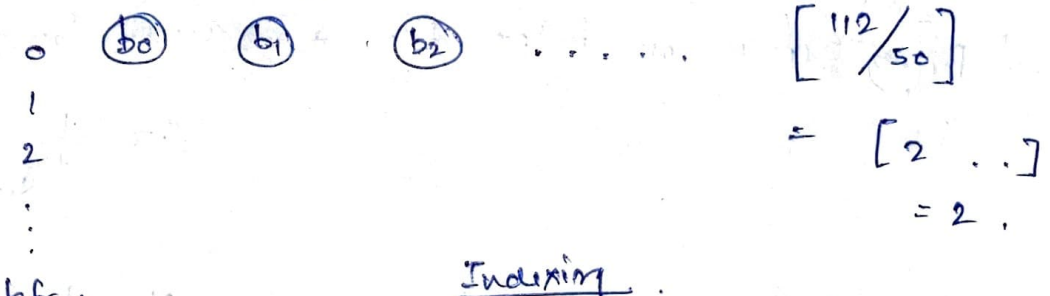
Unordered — A new record will occupy the space of logically deleted one.

Header of a file

→ Info that helps to interpret the data



fixed length record



Indexing

→ To make search in a file fast

Any index file is an ordered file of fixed length record (very often so) whose records are of the form.

< index field value , Record/Block pt. of correspondig record >

Index file

Root Dense

Non-Root Non-Dense

Index file

Primary Index (Non-Dense)

Cluster Index (Non-Dense)

Secondary Index

P.I. Entry → One entry /data block. 1st record of each data block is called block anchor or anchoring record of the block.

Corresponding to each block anchor there is an entry into index file.

Searching for a value:

→ Indexfield value.

- Carry out the binary search on index file to find the entry such that $Key(i) \leq K \leq Key(i+1)$.
- go to the particular data block as provided by matching index entry.

P.I

→ Binary search on index file. + access particular data block.

→

Size of index file << Size of data file.

No. of index records = No. of data blocks (No. of data records

↑ Size of index record.

Data blocks => 50000 / 5 = 10,000

$\lceil \log_2 10{,}000 \rceil$

Data Record added

⇓

blocks anchor may change → P.I needed to be created again

Size of index block => 10 Bytes.

No. of Index Records = No. of data block = 10,000.

No. of index block = $\left\lceil \frac{10{,}000}{50} \right\rceil$

Binary search on Index = $\lceil \log_2 200 \rceil$

14/04/2021

DBMS

Prof. Sanjay Kumar Saha

Indexing

↙

Hashing

Internal Hashing.

hash(key) ⟶ Index to a (table in memory)

hash field. equality checking on hash key.

Collision.

multiple key value ⇒ may give same index

Linear probing

h(a) ⇒ (L) ⟶ deleted.

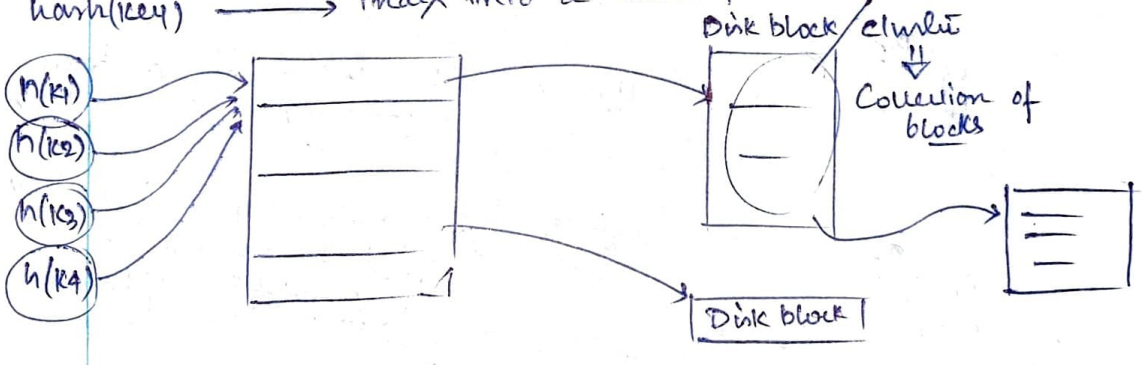
h(K)

hash fn

hash field.

External Hashing

hash(key) ⟶ index into a table



A block/cluster

→ Hold multiple records so collision problem is less severe.

Indexing

- Dense
- Non-Dense Indexing
  - PRI (Non-Dense): Data file is sorted on key → Indexing field.
  - CLUST: Data file is ordered. ~~Order~~ field is Non-key → Ordering.
  - SPC

Records of cluster index file are of fixed length & the form <index field value, Record Ptr>

Index File Access + Data ~~Si~~ der Access.

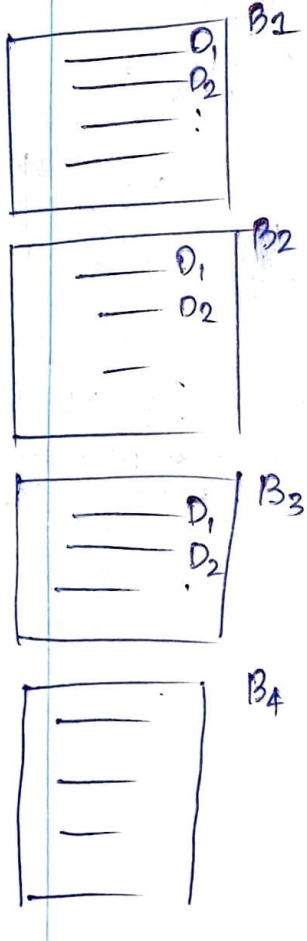
Index on ordering key-field.

Student
- Roll
- Name
- Score
- DoB
- :
- (Dcode)

Sorted on Dcode

In Index File

→ For every distinct value of indexing field an entry is made



No distinct value of ordering/indexing.

Non-key field.

D1 → Address of disk block where the value appears first

D2 →

D3 →

:

D100 →

Search for a part-index value

→ An index file can carry out binary search to find the block containing the desired entry.

Insert/Delete operation

⇒ May lead to changes in data file.

Secondary Index

If Data file is not sorted on indexing field.

P.I. } Index Field
C.I. }
⇓
Data file is sorted on index field.

See Index on Key Field
⇓
Dense Index

Indexing Field
Key Field | Non-Key field.

~~For~~

Data File
⇓
→ Contains 50,000 words
100 Byte/record
Block size → 512 Byte.
bfr = [512/100] = 5 records/block
No. of data blocks ⇒ 5000/5

Linear search on Data File on an average : 5000/2 = 2500 block access

Given a key, find the data record.
⇓
Carry out binary search on index key. Get the record and access the data block in the disk.

Index Record Size ⇒ 10 Byte

bfr = [512/10] ≈ 50

~~No of index records~~
~~⇒ No. of data records~~
=

Secondary Index
⇓
On non-key field

Dense Index
for every data record an Index entry
<Index field name, block pointer>

Non-Dense Index

D_2 —
D_1 —
⋮
D_1 —
D_2

Secondary Index
⇓
on non key field

Non-Dense

→ Set index field value.
→ Go to first level of index file to obtain comp. data block pointer
→ Access the data block.

Variable length record for each distinct value of indexing field. ⇒ index entries.
100 index entry

⟨ Indexed field value , Set of Record pointer ⟩

Data File

Data block
Data block

⟨ P_1 , $P_1, P_2, P_3, \ldots, P_{50}$ ⟩
⟨ D_2 , $P_1, P_2, P_3, \ldots, P_{50}$ ⟩

data block

D_1
D_2
⋮
D_{100}

holding data block problem

Non-Dense

Dependency on the ~~situation~~
→ PRI
→ Cluster
→ Sec.

Index
Fixed length
Ordered File
⇓

→ if it is quite rare

→ Data Block

Multilevel Indexes n-blocks

n-blocks

Base Index

Consider base Index on data file.

⇓

Sorted on Index Field

Index entry / block size | Create Pri Index.

n-entries

$\left[n/bfr \right]$ $\left[n/bfr^t \right] = 1$.

t levels + 1

Search

16/4/2021

DBMS

Prof. Sanjay Kumar Saha

Indexing

Multi-level Indexing

⇓

No. of disk accesses

= No. of levels in Index + 1 (Data Block)

~~Problem → Insertion~~

B/B+T Tree Based Indexing

Multi-way Search Tree of order N.

⇒ N children

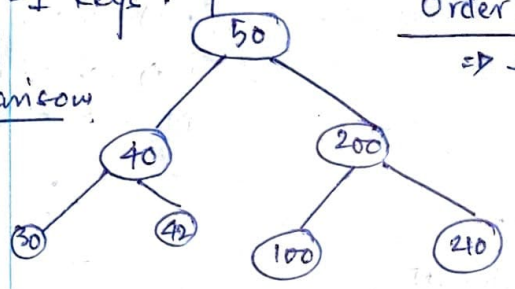
⇒ N-1 keys.

⇒ Sorted Order

Order (2)

⇒ No. of children that a node can have.

$\lceil \log_2 N \rceil$ comparisons



Less Nodes ⇒ less block access

Binary Search Tree | Multiway Search Tree.

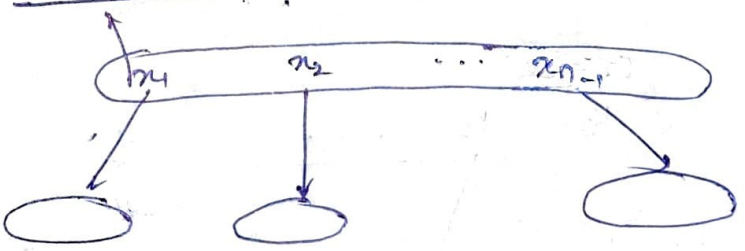
⇓

Depth will be less.

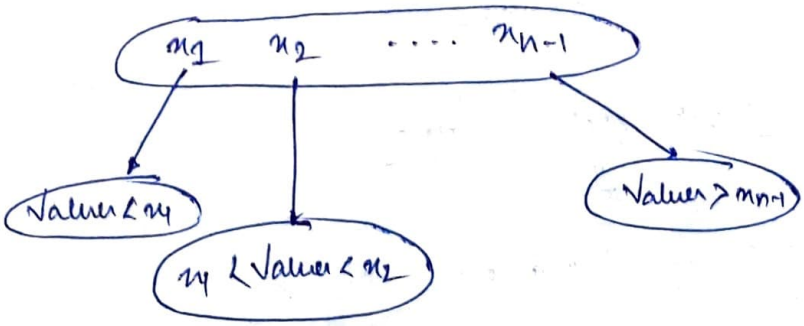
Index → Disk Resident

One node corresponds to a block.

Disk address



B-Tree of order N



1st children => Key value $< n_1$
last children => Key value $> n_{n-1}$

$n_i <$ Key value $< n_{i+1}$

for other its children.

(N-1) → Key Value
(N-1) → Record pointer
N → child pointer

fits into a disk Block

Key	Record ptr

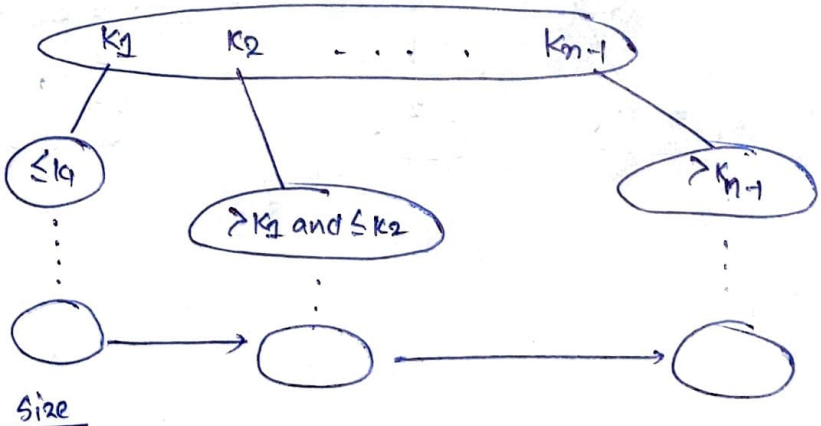
Order N

B+ Tree

All the values are repeated in the leaves. $N_2 > N_1$

Non-leaf node.
N-1 key values
N child ptr.

LEAF Node
N-1 key Value
N-1 Record pointer
1 Next node ptr



Block Size (B).

PL/SQL . (specific to oracle)

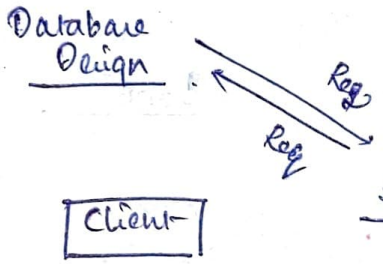
→ To interact with database ⇒ SQL (Non-procedural).

- Provides procedural feature
- While PL/sql block is passed to server in one shot ⇒ No control structure, No Loop etc.

Oracle ⇒ PL/SQL
→ It has the procedural feature

Application ⇒ Procedural aspect is also required.

Host lang ⇒ C/C++/Java
⇓
Procedural aspect



- Each DML statement goes from client to server & response come back
- for each DML statement (Net work traffic increase)

Declare
:
Begin
:
Exception
:
End

Optional

To declare PL/SQL Variables & Constants

Each statement ⇒ Terminated by semi-colon.
⇓
Executable.

X.SQL . SQL-Plus > @x

```
SQL > SET SERVEROUTPUT ON
SQL > :

SET SERVEROUTPUT OFF
```

X	Number (5,2) := 0
Y	Char (5)
Z	Date ;
B	Boolean := true ;
A	Constant Number (3,2) := 3.14 .

Dynamic Datatype Declaration

Salary EMP. BASIC %TYPE
↓
table/view

Conditional statement

If cond. then
=
=
optional { Else
Exit

Loop

Loop
If cond. then
exit.
END IF
END LOOP

can add REVERSE(...)

FOR I IN 1...10
LOOP
END LOOP.

Exit when Condition.

```
for(i=0; i<=10; i++)
{
...
}
```

Must Return a single row else error.

Single Row select statement
If it does not return any row
=> No data_found.
Multiple rows are returned
=> Too many Rows.

EMP (Ecode, Dcode, EName, Basic)

```
DECLARE
   EC    EMP.ECODE%TYPE
   B     EMP.BASIC%TYPE
BEGIN
   EC := '&EC'
SELECT BASIC INTO B
   FROM EMP WHERE ECODE = EC.
END DBMS_OUTPUT.PUTLINE(BASIC || B);
```

Insert
Update
Delete | May not affect any row or affect multiple rows.

Cursor → Is a waiting place in memory to store multiple rows from one at a time. Rows can be fetched.

- Implicit Cursor
- Explicit cursor.

Explicit Cursor

Where one at a time rows can be fetched & can work on it.

Defn.

```
WHERE
CURSOR C1 IS
SELECT *
FROM EMP
WHERE BASIC > 50000 AND DCODE IS NULL;
```

```
BEGIN
  G = '&G'
  OPEN C1;
```

→ Opening the cursor, defn statement is executed & related rows from the active set.

```
  Fetch C1 ;
```

→ To fetch one row (current row) from cursor.

Cursorname %. FOUND
Cursorname %. NOTFOUND.

→ SELECT → INSERT
→ DELETE → UPDATE | → SQL Implicit cursor

%ISOPEN ⟶ TRUE if the cursor is open.
[for implicit cursor, SQL %ISOPEN ⇒ Always False]

%Found → Explicit cursor ⇒ if last fetch operand is successful then it gives true else false.

Implicit cursor ⇒ If last select/insert/update/delete affect at least one row.
[for single row select statement no row ⇒ no data found ⇒ multiple rows ⇒ too many rows].

%Not found.

→ Explicit cursor.
⇒ True if last fetch fail else false.

Implicit Cursor.
→ if last DML statement, affect no row.

For select statement.
→ No rows exception

%Rowcount ⇒ Implicit cursor.
⇒ No. of rows affected.
n = SQL %Rowcount

Explicit cursor.
⇒ Consecutive count of rows fetched so far. ⇒ Cursor is open.

~~Declare~~ Cursor C1, C2 ..

Cursor is open.

OPEN
FETCH
CHECKING
CLOSE

C1%ROWTYPE

FOR (R) INTO C1
LOOP
R.BASIC
END LOOP

PL/SQL

CURSOR

```
DECLARE
   CURSOR C IS
      SELECT *
         FROM RESULT
         WHERE SCODE = 'CS001'
         CITY ROWTYPE

BEGIN
      FOR R ~~IN~~ IN C
      LOOP
         IF R.SCORE < 40 THEN
            UPDATE RESULT
            SET SCORE = 40 ──→ All Rows will be affected.
         END IF                WHERE ~~CURR~~ CURRENT OF C ──→ To prevent
      END LOOP
   END;
```

Using the cursor if we want to go for modification & that will be done on the row in table corresponding to the one fetched from cursor then cursor declaration have an intimation.

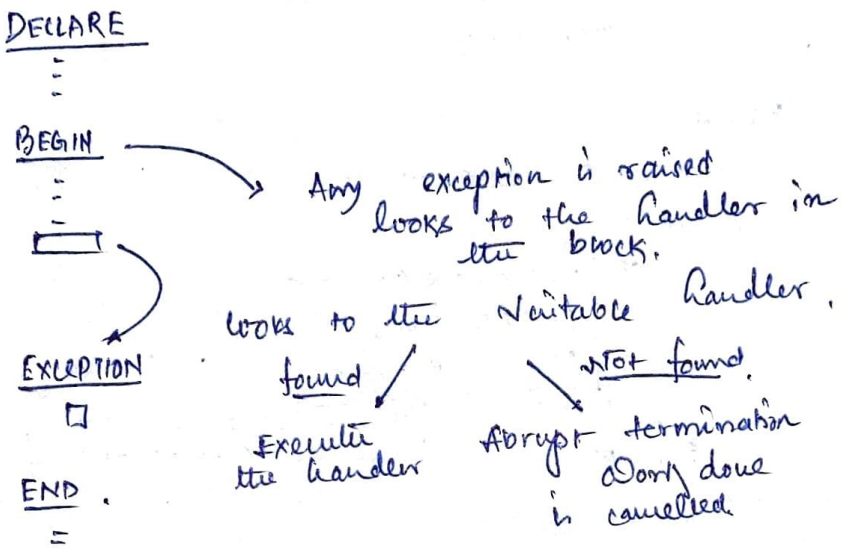
```
      FOR SCODE => S1
        find the row of top5 scores.

DECLARE
  CURSOR MERIT_LIST IS
     SELECT *
     FROM RESULT
     WHERE SCODE = 'S1'
     ORDER BY SCORE DESC ;

BEGIN
   FOR I IN MERIT_LIST
   LOOP
      DBMS_OUTPUT ~~PUTLINE~~.PUTLINE(I.ROLL) ;
   ENDLOOP
```

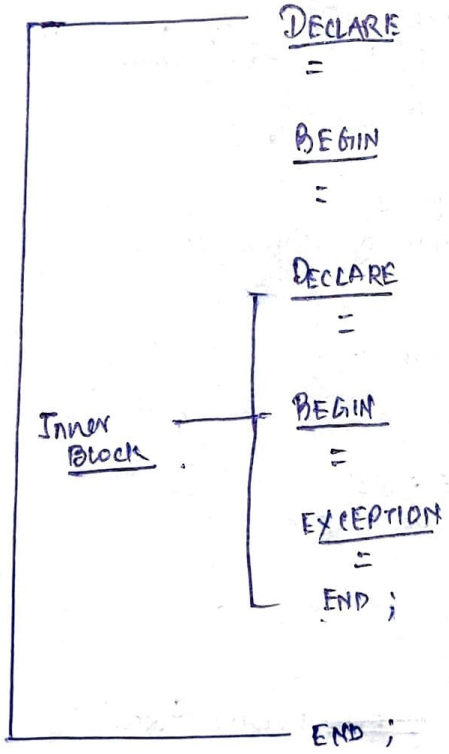
SQL => IMPLICIT CURSOR
SELECT / INSERT / DELETE / UPDATE

n = SQL %. ROW_COUNT

MERITLILT %. ROW_COUNT

PL/SQL => EXCEPTION_HANDLING





~~Variables of inner block is finite of them~~

Not available outside. Outer Block Variables are available to inner block.

EXCEPTION RAISED

↓

EXCEPTION HANDLER AVAILABLE IN THE BLOCK

- Yes
  - Is this suitable handler?
    - Yes
      - Executes & comes out of the block.
    - No
      - Is there any enclosing block handler?
        - Yes
        - No
- No

BUILT-IN EXCEPTIONS

→ No-data found.

A SELECT STATEMENT NOT RETURNING ANY ROW.

[FOR DELETE/UPDATE IT DOES NOT OCCUR] [Aggregate fn returns some value]

→
```
SELECT  ENAME  INTO EN
    EC  EMP  ECODE%TYPE
    EN  EMP  ECODE%.TYPE

EC = '&EC'
SELECT  ENAME  INTO  EN
  FROM  EMP
  WHERE  ECODE = 'EC'
```

(Non-existing employee code. → NO DATA FOUND.)

```
EXCEPTION
WHEN  NO-DATA-FOUND  THEN
  |==
WHEN
  |==
END ;
```

```
SELECT  COUNT(*) INTO C
WHERE  FROM EMP
WHERE  ECODE =
```

(Will not give No data found exception.)

→ Too many Rows,

IF SELECT STATEMENT RETURNS MULTIPLE ROW.

RESULT (ROLL, SCODE, SCORE)

→ NUMBER

→ ZERO_DIVIDE
→ div. by zero.

→ INVALID_NUMBER
IN SQL STATEMENT
IF Conversion of a string to number fails.

INSERT INTO RESULT VALUES ('S1', 'S2', 'ABC').
⇓
Variables.

VALUE_ERROR
⇓
WHENEVER ANY TYPE CONVERSION FAILS EXCEPT THE CASE FOR INVALID NUMBER. VALUE_ERROR

NON SQL STATEMENT
X = 'ABC'

TO A CHAR TYPE.
COLUMN ⇒ DATA LEADS TO TRUNCATION.

INVALID CURSOR
⇓
INVALID OPERATION WITH A CURSOR.
A CURSOR C is not opened. close C.

WHEN OTHERS THEN
⇓
An exception for which handler has not been defined.

RESULT (ROLL, SCODE, SCORE)

EXPLICIT EXCEPTION

DECLARE
R → RESULT ROLL %TYPE
S → RESULT SCODE %TYPE
M → RESULT SCORE %TYPE.
INVALID_MARKS EXCEPTION,

>=0 AND <100

```
IF M<0 OR M>100 THEN
    RAISE INVALID_MARKS
END IF
    INSERT INTO RESULT VALUES (R,S,M)
EXCEPTION
    WHEN INVALID_MARKS THEN
    =
    =
    END.
```

EMP (ECODE , . . . , DCODE , .)

⇓ Must exist in dept.

DC

DC = ' &DC'

SELECT DCODE INTO DC
FROM EMP
WHERE DCODE = 'DC'

→ Valid dept.

→ Exception
Invalid dept.

SELECT COUNT(DCODE)
INTO C2
FROM EMP
WHERE ECODE = EC

TRIGGER

DML statement

→ CONSTRAINTS

Validation | SCHEMA LEVEL CONSTRAINTS.

PL/SQL ⇒ LOGIC.

All constraints
⇓
One cannot deal ~~with~~ at schema level.

→ Application.

→ Is a program that is stored as a part of database.

→ Cannot be invoked explicitly.

→ Trigger is associated with event when the event occurs is called automatically.

← TRIGGER.

SUBJECT (— FM —).

RESULT (— SOURCE).

SCORE <= FM
⇓
SCHEMA LEVEL

→ PL/SQL

→ APPLN (HOST LANG.).

SQL> CREATE OR REPLACE TRIGGER triggername.

↓

optional.

BEFORE / AFTER

INSERT, DELETE, UPDATE OF ROLL, ...

ON TABLENAME

Event

DML statement

May affect multiple rows.

Whether the trigger code could be executed for each affected row / only one for the event.

One or More.

For each row

PL/SQL Block

Declare

Begin

Exception

End

- BEFORE/AFTER TRIGGER
- STATEMENT LEVEL/ROW LEVEL

```
CREATE TRIGGER EXAM_SCORE
BEFORE
INSERT
INTO RESULT
FOR EACH ROW
BEGIN
```

INSERT INTO RESULT VALUES (R, SC, M)

R → Roll

SC → Scode

New → Insert

old → Delete

OLD } - Update
NEW }

Event has to be cancelled.

```
SELECT EM INTO F
    FROM SUBJECT
    WHERE SCODE = :NEW.SCODE

IF :NEW.SCORE > F THEN
|
END IF.
```

PROCEDURE

RAISE_APPLICATION_ERROR (error-code, description)

-20000 to -20999

```
IF :NEW.SCORE > F THEN
    RAISE_APPLICATION_ERROR(-20100, 'SCORE_EXCEEDS
END END IF.
```

SECURITY

→ Only valid user can work
⇓ userid & pwd,

1) DBA → create user account
2) SQL > CREATE USER userid
IDENTIFIED BY pwd
DEFAULT TABLESPACE tablespace.

ORACLE → ORACLE
⇓
DATABASE
⇓
TABLESPACE
(Logical)

CREATE TABLE —
=
=
-
Default tablespace.

CREATE TABLE —

TABLESPACE tablespace name.

SECURITY

WHAT A VALID USER CAN DO IN DBMS?

PRIVELEGES

System Level
(Granted by DBA)
Lowest → Connect (only can connect to the database)
→ Resource (can create my resources)
Highest → DBA (Database administrator).

Object Level
(Owner of the object can grant)

Creator of a table is the owner of the table.

SQL> SELECT CONNECT, Resource
TO
user_id1, user2.

SQL> Revoke privilege.
From
userid1, userid2

SQL> GRANT privilege1, privilege2 | ALL.
ON tablename (objectname)
TO userid, ... | PUBLIC.
→ With GRANT OPTION
⇓
Being given the privilege to grant others.

Revoke privilege1, ...
ON Resourcename
FROM user_id1

DBA

SQL> GRANT RESOURCE (TOM)
ON Tablespace name
TO userid1, ...

FROM

TS1
TS2
TS3

Query Processing

To store/retrieve data from database.

⇓

DML Statement
- Now procedural
- Query
- SELECT
- DELETE
- UPDATE
- INSERT

To EXECUTE ⇒ Detailed planning is required

Functional unit of DBMS

⇓

Query processor.

User Given Query

⇓

Can be done in no. of ways.

⇓

Query processor is responsible.

...

User Given Query

⇓

1) Translation → Translated into an equivalent & efficient form into an intermediate form

2) Optimization ⇒ Acts on translated version.

Translation

- Takes the query as input & as output presents an intermediate form (Relational Algebra)
- Query is parsed into tokens.
- Verifies the syntax
- Verifies the relation name, attributes, etc. → Data Dictionary

Optimisation / Expression level optimization

→ First Phase → Relational Algebra Expression (obtained after translation) is converted into equivalent but more optimised form).

→ Detailed plan of execution (like, index to be used or not, joining strategy, etc).

Expression Level Optimisation

URBAN/RURAL

→ EMP TABLE (DCODE, DNAME, LOCTYPE)

DEPT TABLE (ECODE, ENAME, BASIC, ..., DCODE)

$\sigma_{BASIC>50000 \text{ AND } LOCTYPE='R'}$ (EMP ⋈ DEPT)

EMP → 20000 × DEPT → 100

⇓

$\sigma_{BASIC>50000}(EMP) \bowtie \sigma_{LOCTYPE='R'}(DEPT)$

$\sigma_{BASIC>50000}(EMP)$ < 20,000 ; $\sigma_{LOCTYPE='R'}(DEPT)$ < 100

$R_1 \bowtie R_2$

⇓

Cartesian product of $R_1 \bowtie R_2$

⇓

Filtering based on join criteria.

→ If no. of tuples are large in R_1, R_2

Accessing them ⇒ ~~More Data~~ Be

Intermediate Memory Requirement is Large. (EMP & DEPT)

$\sigma_{BASIC>50000 \text{ AND } LOCTYPE='R'}$

⇓

APPLY SELECT OPERN EARLIER ⇒ $\sigma_{BASIC>50000}(EMP) \bowtie \sigma_{LOCTYPE='R'}(DEPT)$

At subsequent operation cost will be handling less no. of tuples.

$\pi_{A_1, B_1}(R_1 \bowtie R_2)$ ← ~~DEPT (DCODE~~

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

$R_2(B_1, B_2, \ldots, B_m, A_1)$

If the schema of participating rels are large ⇒ tuple after cartesian projection will also be large.

Cartesian Product ⇒ $n_1 \times n_2$ tuples

$R_1 \rightarrow n_1$ tuples

$R_2 \rightarrow n_2$ tuples

Schema ⇒ Schema of R_1 ∪ Schema of R_2

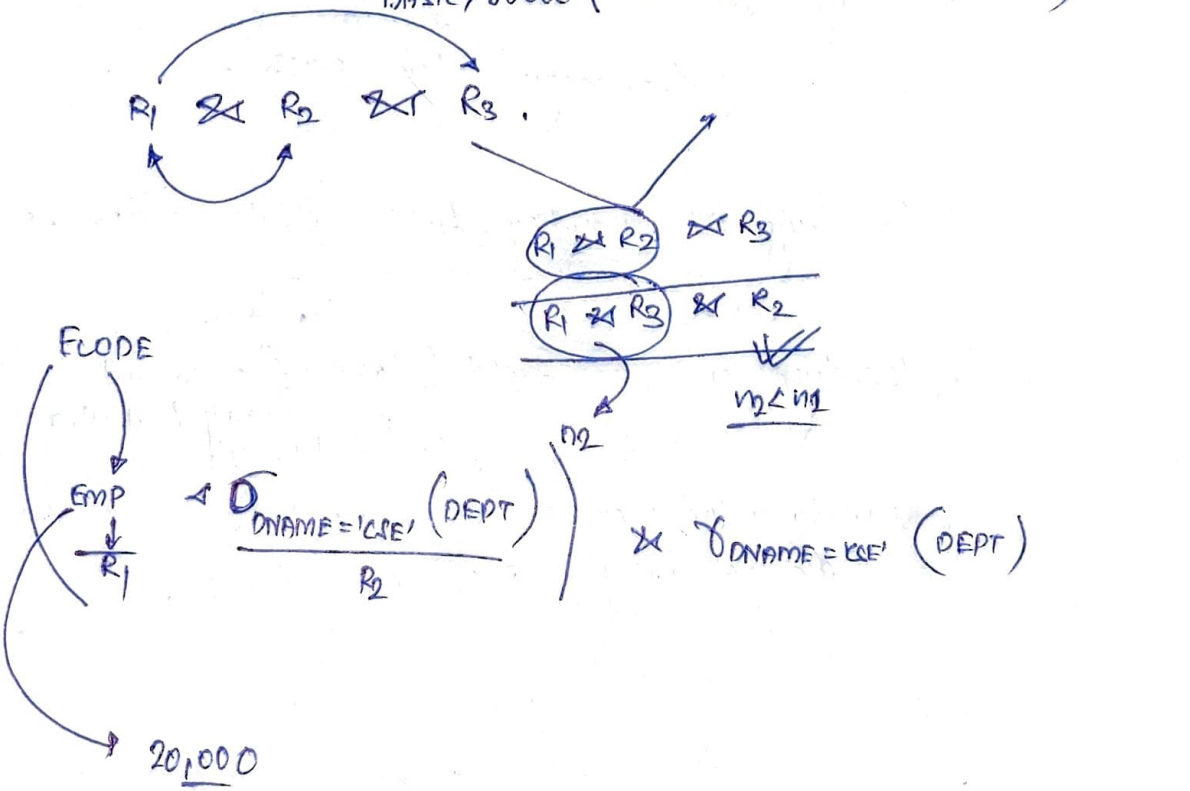
$\Pi_{A_1}(R_1) \bowtie \Pi_{B_1}(R_2)$

- Apply Projection earlier to make the size of tuple smaller for next level of processing.
- TAKE THE PROJECTION OF ATTRIBUTES Required in final output & also the attributes required for subsequent processing.

$\Pi_{ENAME, DNAME}\left(\sigma_{BASIC>50000 \text{ AND } LOCTYPE='R'}(EMP \bowtie DEPT)\right)$

$\Pi_{ENAME, DNAME}\left(\sigma_{BASIC>50000}(EMP)\right) \bowtie \Pi_{DCODE, DNAME}\left(\sigma_{LOCTYPE='R'}(DEPT)\right)$

$\Pi_{ENAME, DNAME}\left(\sigma_{BASIC>50000}\left(\Pi_{ENAME, DCODE, BASIC}(EMP)\right)\right)$



→ SELECT ✓

→ PROJECT $(R_1 \bowtie R_2 \bowtie R_3)$

→ JOIN OPERATION , $(R_2 \bowtie R_3)$.

$R_1 \bowtie R_2 \bowtie R_3$

More than one order of join operation is important.

Detailed strategy level optimization

SELECT OPERATION

→ BASED ON SIMPLE CRITERIA (ie single attribute based).

- Linear search (can be used always)
- Search Criteria involves an attribute & a data file is sorted on that attribute carry out binary search on data file
- Looking for a particular record based on key field & primary index exists. Use primary index for searching.

Single Record
Roll=10

- Looking for a set of records specified by key field & primary index exists. multiple records.
- Searching on non-key field, cluster index exists ⇓ Use it.
- Searching on a field (data file is not sorted no primary / cluster index is available)
- Secondary index exists → use it
  Range Query ≥ 20 & ≤ 20.

Query Processing

Translation Optimization → Exp. Level
→ 2nd phase Detailed Strategy.

→ How to deal with Select operation?

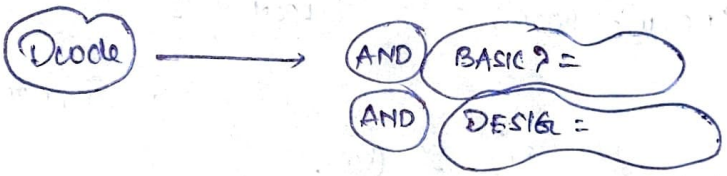
- Linear Search
- Binary Search

Primary Index → Single Record
→ Multiple Record.

Clustering Index → Multiple Record.

Secondary Index → Single Record
→ Range Query.

→ Select criteria involves multiple condn involving different attributes



- **Using Single Index**

Suppose Index on one particular attribute is available. Use it to get a set of tuples & on those check other condn.

- **Using Composite Index**

Secondary Index on (Dcode * Desig)

Use it & on retrieved tuples. Check the other condition if any.

- Multiple indices exist

Dcode →

Deliq: →

Use index tuples to join records/record pointers satisfying the individual condn take the intersection → satisfy those condn if any, addln criteria is there. Check with the intersection.

Join

$R_1 \bowtie R_2$ equijoin on common attribute or predicate

$\sigma_{Pred}(R_1 \times R_2)$

Emp ⋈ Dept

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

$\sigma_{BASIC >= 5000 \text{ AND } LOCN='XYZ'}(EMP \times DEPT)$

Join.

$R_1 \times R_2$.

Suppose it contains n_1 tuples

n_2 tuples.

Simple Iteration

```
for each tuple t1 in R1
{
    for each tuple t2 in R2
    {
        check(t1, t2) whether they should
    }   appear in output or not
}
```

Worst case.

Each tuple may be in different disk block.

Each tuple → one disk block read.

No. of disk block access

All tuples of R_1 are read only once.
→ n_1 block access,

→ All tuples of R_2 are to be read n_1 times,

to read all tuples of R_2 once. ⇒ n_2 block access.

n_1 times ⇒ $n_1 * n_2$ block access.

Total no. of block access : $\boxed{n_1 + n_1 * n_2}$

for R_2

n_2 times.

$\boxed{n_2 + n_1 * n_2}$ → Better to keep smaller one in outer loop.

Smaller one is so small that all the tuples can be put into memory then strategy will be different.

Say, R_2 is smaller

→ Read all tuples of R_2 into buffer → All tuples of t_2 read in memory: n_2 block access.

→ for each tuple t_1 in R_1 → Once all tuples of R_1 are read.

{

check with each tuple t_2 of R_2 (already in buffer).

~~else~~ decide

}.

Blocks iteration

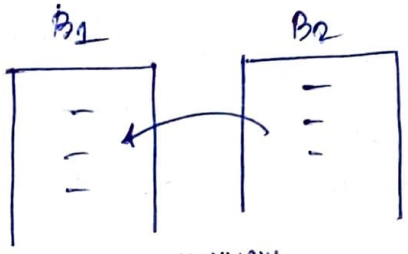
Suppose tuples of each relation are physically stored together. A block holds tuples of one particular relⁿ.

A reln R_1 ⇒ n_1 tuples:

Block B_1 ⇒ data of R_1

Block size.

bfr_1



Disk access

→ for each block B_1 of R_1 ⇒ All the blocks of R_1 read once
⇒ N_1 block access.

memory

→ for each block B_2 of R_2 ⇒ All the blocks of R_2 read N_1 times. One such read (all block of R_2)
⇓
N_2 block access.

$N_1 * N_2$.

No disk access.

```
{
    for each tuple t1 in B1
    {
        for each tuple t2 in B2
        {
            check (t1, t2) & decide.
        }
    }
}
```

Total $N_1 + N_1 * N_2$.

↳ $\left[n_1/bfr_1\right] + \left[n_1/bfr_1\right] \cdot \left[n_2/bfr_2\right]$

$R_1 \Rightarrow$ n_1 tuples, bfr_1 } = No. of disk blocks $= \left[n_1/bfr_1\right] = N_1$.

n_2 tuples bfr_2 } = No. of disk blocks $= \left[n_2/bfr_2\right] = N_2$.

$R_1 * R_2$

Both the Relations are Sorted.
⇓
Also tuples of a reln are physically stored together.

~~Joins~~

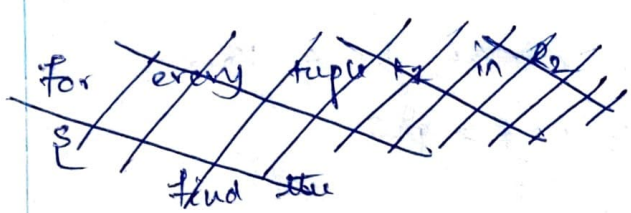
Joining Attribute

Merge Join

R_1 & R_2 points to first tuple of R_1 & R_2 Repeat

```
while (p1 != NULL && p2 != NULL)
{
    t1 = *p1;
    S1 = {R1};
    p1++;
    while (p1 != NULL && *p1[i] == t1[i])
    {
        S1 = S1 ∪ {*p1};
    }
```

```
for each element e1 ∈ S1
for each element e2 ∈ S2
{
    Put e1 × e2 in the output
}
}
```



Based on joining attribute of hash function is applied on the tuples of $R_1 \times R_2$

K buckets

$h(A) \rightarrow 1, 2, \ldots, K$.

$H_{1i} \Rightarrow$ } Buckets for R_1 & R_2 respectively.
$H_{2i} \Rightarrow$ }

Relations ⇓ Not stored together. Not sorted.

Hash Join.

```
for(i = 1 to K)
{
    H1i = {} ;
    H2i = {} ;
}

for each tuple t1 in R1
{
    i = h(t1[A])
    H1i = H1i ∪ { Record pointer corresponding to t1 }
}
For each tuple t2 in R2
{   i = h(t2[A])
    H2i = H2i ∪ { Record pointer corresponding to t2 }.
}
```

Disk Read/write. Rec. acc.

$S_1 = \{\}$
$S_2 = \{\}$

For ($i = 1$ ~~to~~ K)
{
for each element $p_1 \in H_{1i}$
$S_1 = S_1 \cup \{ * p_1 \}$.

for each element $p_2 \in H_{2i}$
$S_2 = S_2 \cup \{ * p_2 \}$.

for each $t_1 \in S_1$
for each $t_2 \in S_2$
{ ~~otuic~~ check(t_1, t_2) ✗
decide
} }

Internal memory to hold the record pointers of both the relations.