DBMS

File Systems

1) data gets duplicated (data redundancy) which results in inconsistency.
   + Same data stored in many places.
   → if value of the same data stored in different places have different values
2) sorting of data is difficult
3) Data retrieval is difficult
4) more storage space
5) more human intervention required
6) sharing files is difficult, ensuring security is difficult

Different DBMS – Oracle
– SQL server
– MS ~~Access~~ Excel
} etc.

DBMS → collection of inter related data and a set of programs to access those data. (software used to create, manipulate and delete DB).
– goal is to provide a way to store and retrieve DB info that is both convenient and efficient.

Major drawbacks of file processing systems

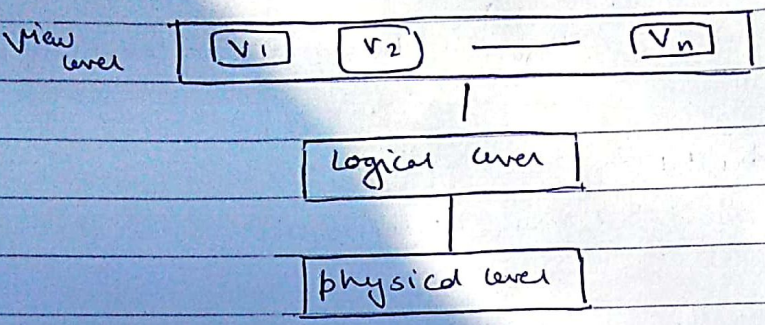
1) data redundancy and inconsistency — Data isolation (data is scattered)
2) difficulty in accessing data
3) integrity problems (some constraints which should not be violated)
   – adding new constraints is difficult
4) Atomicity problem } eg: banking (transaction)
   – all or none } + internet
5) ~~Concurrent~~ Concurrent Access Problem (eg: we are booking tickets at the same time in railways)
6) Security problem — ~~Unauthorised~~ who all can read, or who all can write.

View of Data

1) Physical level – how the data are actually stored
2) Logical level – what data are stored in DB and what relationship

exist among them.

3) View Level – describes only part of the entire DB.
– simplify interaction with the system



Instances and Schema

- Collection of info. stored in a DB at a particular moment – instance
- design of DB is called DB schema
  instance can change, schema can not change
- DB systems have several schemas partitioned according to levels of abstraction.
  1) Physical Schema – physical interpretation and storage
  2) Logical Schema – DB design at logical level
  3) Subschemas – at the view level that describes

Data Models

- Collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints

1) Relational Model – uses collection of tables to represent both data and relationship among those data.
- each table has multiple columns and each columns has a unique name
- table is also called relations
-

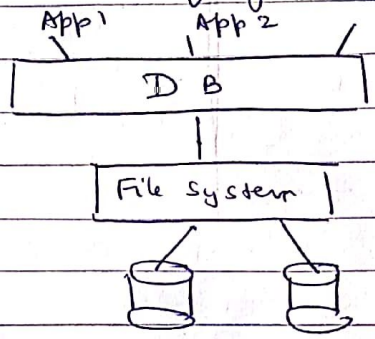
DBMS

2) Entity Relationship model (ER) -

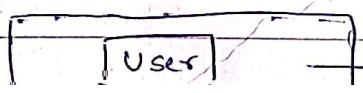
- uses collection of basic objects called entities and relationships among these objects.
- an entity is a thing or object in the real world that is distinguishable from other objects.

Database Architecture

- influenced by the underlying computer system on which DB system runs.
- DB systems can be centralized or client-server where one server machine executes work on behalf of multiple client machines.



Three-Tier Architecture



during termind

2) Designer chose a data model and translates these requirements into a conceptual schema of the DB.

The ER model is used to represent the conceptual design.

3) A fully developed conceptual schema indicates find requirements of the enterprise.

4) Process of moving from an abstract data model to the implementation of the DB proceeds in 2 final design phases.

a) in logical design phase - mapping conceptual schema into a relational schema

b) finally, designs uses the resulting system specific DB schema in the physical design phase.

physical features of DB are specified, index, file organizations, etc.

2 major pitfalls

1) Redundancy -

2) Incompleteness - a bad design may make certain aspects of the enterprise difficult or impossible to model

Entity Relationship Model

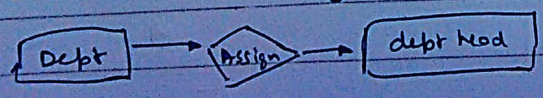
1) entity set - An entity is a thing or object in the real world that is distinguishable from all other objects.

⇒ An entity set is a set of entities of same type that share same properties or attributes

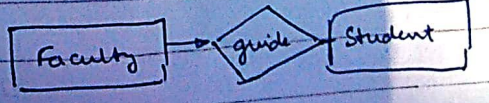
- entity set do not need to be disjoint.
- An entity is represented by a set of attributes eg: id, name, dept, etc.

2) relationship sets - ◇

- association among several entities



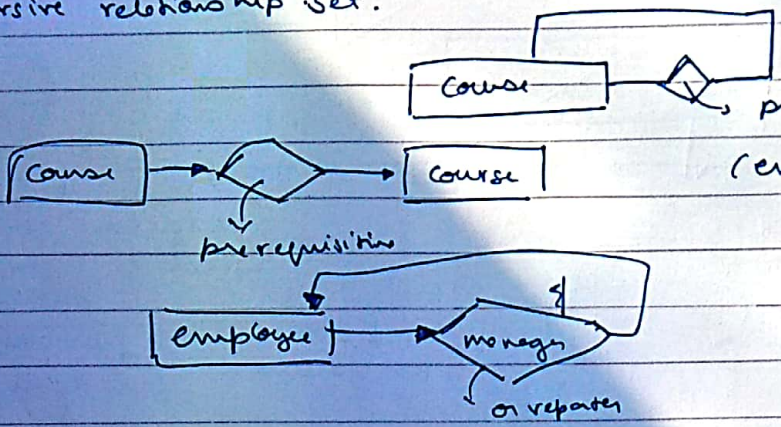
not necessary all are related (some of them are related)



DBMS

- Association between entity sets is referred to as participation
- The function that an entity plays in a relationship is called the entity role. (predefined)

- Recursive Relationship - same entity set participates in a relationship set more than once, in different roles called recursive relationship set.

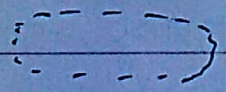


- A relationship may also have attribute called descriptive attribute.

- Single valued and multivalued → eg: phone no, email
  - ↓ only one value possible. eg: id, name, dob


single valued (normal representation)


multivalued } we need this for capturing info

- Derived Attribute eg: ~~date~~ age (can be calculated from date of birth.
  - represented as 
  - ~~values~~ are not stored in the database at all
- value of derived attribute is not stored at all.

- Null values ⇒ attributes can have NULL values if
  1) the entity does not have a value for it. eg: middle name
  2) value is ~~missing~~ unknown

a) missing - exist but does not have information
b) unknown - do not know if information exist.

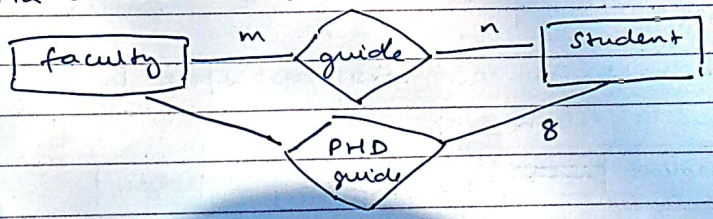
- Simple and Composite Attributes

not divided into subparts
eg: dob, pan no, etc.

divide into subparts → eg: name (middle, first, last)
address

- Constraints - to which contents of a DB must confirm
eg: in an email @ shoul be used., phone no should be numeric and 10 digits

1) Mapping Cardinalities (cardinality ratio) → number
- express no of entities to which another entity can be associated with via a relationship set.

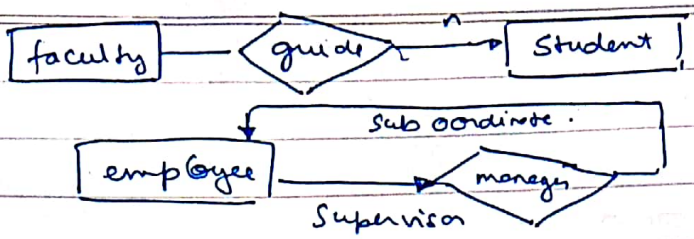


Mapping Cardinalities
- useful in binary relationships
- in a binary relationship, mapping cardinalities must be one of the following:

a) One-to-One

(there may be a new employee, which has not yet been assigned workstation or vice versa) → they do not participate in the relationship.

b) One to many
- student can be guided by a single faculty
- faculty may guide multiple students.

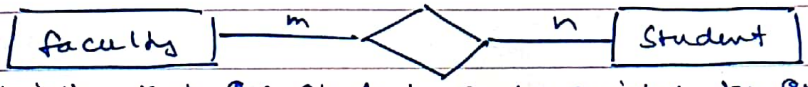


multiple employees can report to a same manager (but they report to just one manager)

c) Many to one

a single state can have many cities, but a city can belong only to one state ~~(if not)~~. (backward representation of one to many)

d) Many to many



without the restriction that one student can be guided by only one faculty

Participation

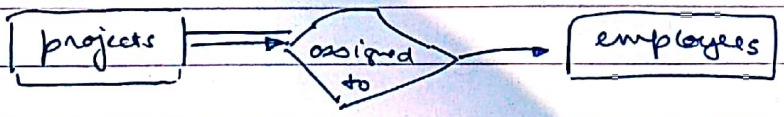
~~partial~~ ~~record~~

total — partial

(participation from the employees side is partial)

: if every entity in the entity set participate in atleast one relationship in relationship set -

eg:



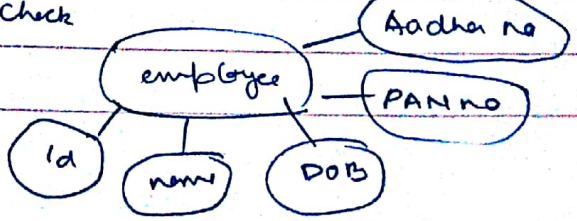
participation from the project side is total (symbolized by ==)

Keys - Keys for an entity is a set of attributes that suffices to distinguish entities from each other.

id	name	DOB	DOJ	PAN NO	Aadhar No
1	ABC	XYZ		A	101
2				B	102
3	ABC			C	103
4		XYZ		D	104

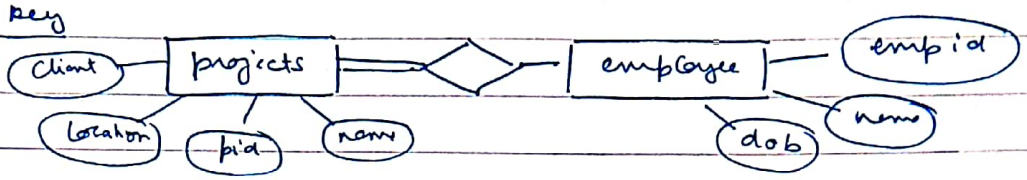
× × × (unique & not null) (can be null)

uniqueness and not null — two of things to check for keys



4 Types of Keys

1) Primary key } unique + not NULL represented by 
2) Candidate keys } which can be the primary keys
   we have to choose one and make them primary key.
3) Alternate key } all the candidate keys except primary key
4) Foreign key



total participation from project side.
when we map ER model to table.

this is a primary key of table projects
⇒ it is a foreign key

empid	name	- ..	pid

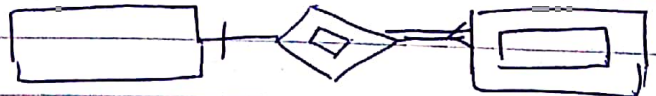
foreign key can have null values and repeated values also.
(it will have only values stored in the original table)

Weak Entity Set

- entity set that does not contain sufficient attribute to form a primary key is termed as a weak entity set.

Entities belonging to a weak entity set are identified by being related to specific entities from another entity set in combination with one of their attribute values.

even if a key contains all the components of the collection, they will not be uniquely identified.



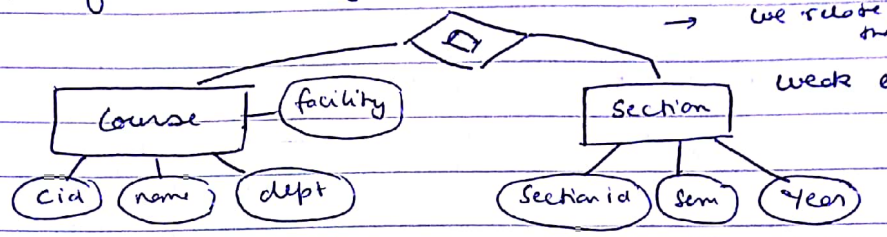
Strong entity set (identifying entity set) — weak entity set

- This entity set is called the identifying or owner entity set.

DBMS

- Weak entity set must be associated with an identifying set.
- Identifying relationship set is many-to-one from weak entity set to the identifying entity set.
- Participation of the weak entity set in the relationship is total.

eg:



→ we relate it with the course.

	A	VI	2019	} of two different branch
multiple rows with	A	VI	2019	
some identical data	B	II	2018	

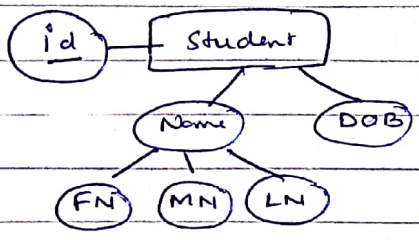
Cid will be used for identifying uniquely data of one set

primary keys. (alternate keys) can not be used in another entity set.

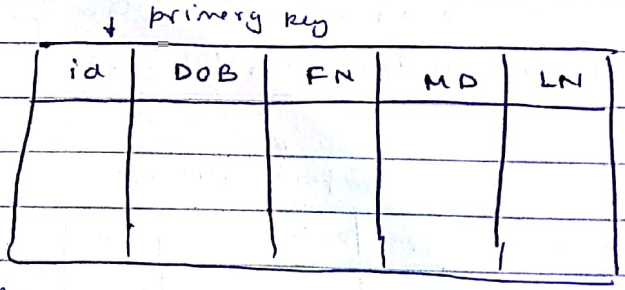
Reduction to Relational Schema

- Representation of strong entity sets with simple attributes
- SES → table with attributes as columns

primary key of entity set → primary key of relationship / table.

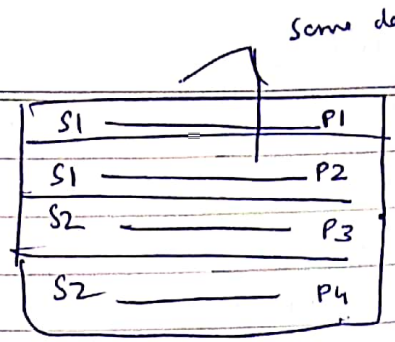


~~for complex attributes~~



- For composite attributes → Create seperate attributes for each of the component attributes (eg: name)
- derived attributes → not stored explicitly (eg: age)
- multivalued attribute. → (eg: phone no)

two possibilities



but here data redundancy is there.

~~but~~ to We will make another table as.

for each multivalue ~~data~~ attribute.

Sid	Phone
S1	P1
S1	P2
S2	P4
S2	P5

null's will not be here

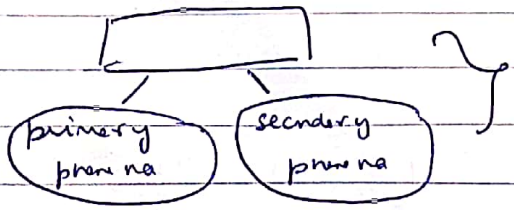
If max no of phone no's is 3

and if we have the maximum no of phone no.

for eg:

data redundancy will not be there but here the problem is NULL

Sid	Phone 1	Phone 2	Phone 3



this is not multivalued as each will only have 1 value.

Representation of WES

A (WES) = {a1, a2, ..., am}. → attributes of the weak entity set.

B (IES) = { b1, b2, ... , bn, ... }

↓ Composite primary key of the identifying entity set.

to

a1	a2	- - -	am	b1	b2	- - - bn

Representation of Relationship sets

- R = {a1, a2, — am} ← union of primary keys of each of the entity sets participating in R.

{ b1, b2, - - . bn } — descriptive attribute.

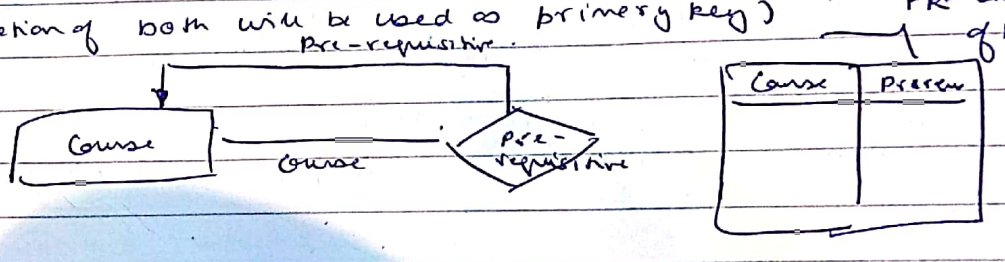
- For Binary Relationship (primary key

1) many-to-many
   - union of primary key of both the entity set.

2) one-to-one
   - primary key of either Entity set can be chosen (as the primary key)

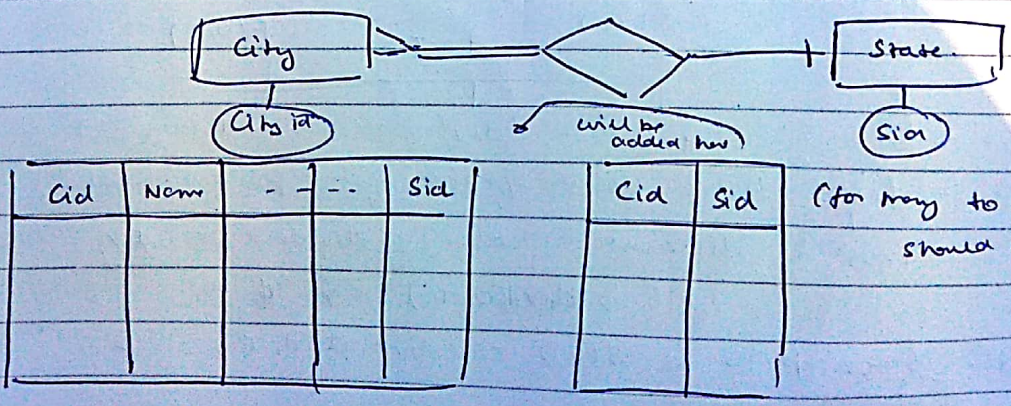
3) many-to-1 or 1-to-many relationships
   - primary key on the ES on the many side of the relationship set serves as the primary key.

- For n-ary relationships, union of primary key from participating Entity set is taken as the primary key of the table.
- For recursive relationship set, pre-required - roles indicators associated with the relationships are used as attribute name.
  (Combination of both will be used as primary key)



Identifying relationship => no scheme is created.

- Combination of schemes => • for many-to-1 relationship and participation of A (many side) is total.
  => Combine A and AB to form a single scheme.



(for many to one this should be)

and this is how it is stored ← will be rejected.

DBMS

UNIT-2

Relational Algebra and Relational Calculus

- RA and RC were developed before SQL.
- The basic set of operations for the formal relational model is called the relational algebra.
- These operations enable a user to specify basic retrieval requests as relational algebra expressions.
- The result is a new relation.
- The relational calculus provides a higher level declarative language for specifying relational queries
- In relational calculus, there is no order of operations to specify how to retrieve the query result - only what info the result should contain.

Relational algebra operations can be divided into 2 groups =>

1) include set operations - union, intersection, set difference, cartesian product
2) operations specifically for relational database, ex: SELECT, PROJECT, JOIN, etc.

Unary Relational Operations -

1) SELECT (σ) - to chose a subset of tuples (rows) from a relation that satisfies a selection condition.

$\sigma_{D.NO=5}(emp)$ emp = table or relation

↳ selection condition.

↓ result in all the employees from the emp table whose department no is 5.

In SQL =>

```
SELECT *
FROM EMP
WHERE D.NO = 5
```

* (all the columns)

- Selection condition is a boolean expression on the attributes of relation R (can be true or false)
- Clauses can be connected by standard boolean operations (and, or, not)

attributes ↓ columns

In case of select, all rows will be distinct

to form a general selection condition.

eg: $\sigma_{(D.NO=5 \wedge Manager='Smith')}(emp)$ (∧ = and)

↳ we want all employees whose dept no is 5 and manager is Smith.

- The selection condition is applied independently to each ~~individual~~ independently to each individual tuple in R
- Select operator is unary - applied to a single relation
- degree (no of columns) of resulting from a select operation = degree of the original relation

- The ~~fraction~~ fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.
- $\sigma_{<Cond1>}(\sigma_{<Cond2>}(R)) = \sigma_{<Cond2>}(\sigma_{<Cond1>}(R))$
  σ Operator is commutative.

2) Project operation (π)

- selects certain columns from the table and discards other cols.
  - $\pi_{<name, salary>}(emp)$
    ↳ it will give all the rows, but only name and salary.
  - $\pi_{<name, salary>}(\sigma_{(DNO=5)}(emp))$ — this will pull the entire data of employees who D.NO = 5
    ↳ name and salary of those employees whose d.NO = 5
    ↳ final outcome.
- Removes any duplicate tuples
- If duplicates are not eliminated, result would be a multi set (DO NOT CONFUSE WITH SQL => it is different)
- Commutativity does not hold on PROJECT.

$\pi_{name}(\pi_{name, DOB}(emp))$

↳ it will result in name of all the employees.

In case of select, all rows will be distinct

to form a general selection condition.

eg: $\sigma_{(D.NO=5 \wedge Manager = 'Smith')}(emp)$ (and)

↳ we want all employees whose dept no is 5 and manager is Smith.

The selection condition is applied independently to each ~~individual~~ independently to each individual tuple in R

Select operator is unary - applied to a single relation

- degree (no of columns) of resulting from a select operation = degree of the original relation

- The ~~fraction~~ fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.
- $\sigma_{<Cond1>}(\sigma_{<Cond2>}(R)) = \sigma_{<Cond2>}(\sigma_{<Cond1>}(R))$

σ Operator is commutative.

2.) Project operation (Π)

- selects certain columns from the table and discards other cols.

- $\Pi_{<name, salary>}(emp)$

↳ it will give all the rows, but only names and salary.

- $\Pi_{<name, salary>}(\sigma_{(DNO=5)}(emp))$ ↳ this will pull the entire data of employees who D.NO = 5

↳ name and salary of those employees whose d.NO = 5

↳ final outcome.

- Removes any duplicate tuples

If duplicates are not eliminated, result would be a multi set

(DO NOT CONFUSE WITH SQL => it is different)

Commutativity does not hold on PROJECT.

$\Pi_{name}(\Pi_{name, DOB}(emp))$

↳ it will result in name of all the employees.

$\pi_{Name, \cancel{Emp}\ DOB}(\pi_{name}(emp))$ is not defined.

- In SQL, PROJECT attribute list is specified in the SELECT clause of the querry.
- If we remove the keyword DISTINCT from SQL query, duplicates will not be eliminated.

```
SELECT Salary
FROM   Emp
```
↳ duplicates will not be removed

```
SELECT DISTINCT Salary
FROM   Emp
```
if we want non-duplicate values, DISTINCT is used

Inline expressions

$\pi_{Fname, Name, Sal}(\sigma_{D.NO=5}(emp))$

another option

but the output of this expression in a table and then apply the other operation on this table.

- Or Explicitly show sequence of operations giving a name to each intermediate relations and using assignment operation ($\leftarrow$)

$Demp5 \leftarrow \sigma_{DNO=5}(emp)$

$Result \leftarrow \pi_{Fname, Name, Sal}(Demp5)$

attribute names will be same as the original one i.e Fname, Name, Sal.

If we want to change the attribute values do

$Result(First\ name, Last\ name, Salary) \leftarrow \pi_{Fname, Lname, Sal}(Demp5)$

Rename operation (ρ)

$\rho_{S(B_1, B_2, \dots B_n)}(R)$

S => new relational name

$B_1, B_2, \dots B_n$ => new attribute names in order

we can not explicitly change the values of attributes in between, we will have to specify the ~~alone~~ attribute names before it also.

- In SQL, use keyword AS

```
SELECT   E.Fname   AS   First-Name,
         E.Lname   AS   Last-Name,
         E.Sal     AS   Salary-of-emp
FROM     Emp  AS E
WHERE    E.D.NO = 5
```

Relational Algebra Operations from Set Theory

- UNION, INTERSECTION, SET DIFF
- binary operations
- Union Compatability =>

2 relations must have same type of tuples

1) no of columns should be same
2) data type of each corresponding columns should be same.

ENO	ENAME	DOJ	PAN

Emp

S.NO	SNAME	DOB	PAN

Student

both these tables are union compatible.
(even though it is ~~union~~ not useful)

UNION (RUS)

- includes tuples that are either in R or in S or in both
- Duplicates are eliminated (if the entire row is same)
- Names of the column will be the names of the column

Union of Emp table and Student table.

ENO	ENAME	DOJ	PAN

} Data

INTERSECTION R∩S (Common between the two tables)

SET-DIFF (R-S) all the rows that are in R but not in S

(Emp - Student) will result in all the employees which are exclusively employees (i.e they are not students)

CARTESIAN PRODUCT (no concept of union compatibility)

(Cross product) Cross Join ×

Emp × Student =>

ENO	ENAME	DOJ	PAN	SNO	SNAME	DOB	PAN
E1				S1			
E1				S2			
E1				S,N			
E2				S1			
E3							

R (A1, A2 —— An) × S (B1, B2 —— Bm)

= Φ (A1, A2 —— An, B1 —— Bm)

no of columns of both the table ~~should~~ may not be same

Join (|X|) Combines related tuples from 2 relations into single larger tuple.

Join ≃ Cartesian product (selection condition)

Emp |X| Dept
Emp.D.NO = Dept.D.NO

First it will create a cartesian product and then choose all those values where d.NO = dept d.NO.

R |X| <join condition> S

Result of join is a relation Q with n+m attributes

INTERSECTION R∩S (Common between the two tables)

SET-DIFF (R-S) all the rows that are in R but not in S

(Emp - Student) will result in all the employees which are exclusively employees (i.e they are not students)

CARTESIAN PRODUCT (no concept of union compatibility)

(Cross product) Cross Join ×

Emp × Student =>

ENO	ENAME	DOJ	PAN-1	SNO	SNAME	DOB	PAN
E1				S1			
E1				S2			
E1				S N			
E2				S1			
E3							

R (A1, A2 —— An) × S (B1, B2 —— Bm)

= Q (A1, A2 —— An, B1 —— Bm)

no of columns of both the table ~~should~~ may not be same

Join (⋈) Combines related tuples from 2 relations into single larger tuple.

Join ≃ Cartesian Product (Selection condition)

Emp ⋈ Dept

Emp.D.NO = Dept.D.NO] First it will create a cartesian product and then choose for all those values where d.NO = dept d.NO.

R ⋈ <join condition> S

Result of join is a relation Q with n+m attributes

R(n) S(m)

- Q has 1 tuple for each combination of tuples
- A join with such a general condition is called theta join.
- Tuples whose join attributes are NULL or for which the join condition is false do not appear in the result.

Equi join or Natural Join

- "only = is used
- have one or more pairs of attributes that have identical values in every tuple.
  ∴ one of each pair of attribute with identical value is superfluous
- Natural Join = equi join + removal of superfluous attribute
- Natural Join requires that the two join attributes have the same name in both relations.

- Join Selectivity - rate of expected size of join result divided by the maximum size $n_R \times n_S$

Inner Join - match and combine operation defined as a combination of Cartesian product and selection.

$R \bowtie_{condition} S = \sigma_{<condition>}(R \times S)$

Division Operation (÷)

- Result of division is a relation T(Y) that includes a tuple t if tuples t_R appears in R with $t_R[Y] = t$ and $t_R[X] = t$ $\forall$ tuples in S

Retrieve names of employees who work on all the projects that 'John Smith' works on.

SMITH ← $\sigma_{FName = 'John', LName = 'Smith'}(Emp)$ → all details of employee named Smith

SMITH_PNOS ← π_{PNo} (WORKS-ON ⋈ SMITH)

it will create join of table work-on and Smith. Emp.NO = ENo.

↑ this picks all the project nos when John Smith is working on.

Emp table

E.NO	E.Name	DOB	DOJ	PAN
E1				
E11	John Smith			
En				

this data will be produced by query 1 resulting in table smith

E11	John Smith	.	-	-	-

SMITH

works on table

Emp NO	P.NO	-
E1	P1	
E1	P6	
⋮		
E		
E11	P24	
E11	P12	
E11	P6	

these two would now be joined when E.NO = Emp NO

Output no-w will be.

PNO
P24
P12
P6

∈ SMITH - PNOS

b) Emp_Smith (ENO) ← Works_On ÷ Smith_PNOS

↓

it will pull all the data of all employees (one by one) and it will divide the value by the entire table Smith_PNOS (if it exists completely or not)

Result ← $\Pi_{fname, lname}$ (Emp_Smith * Emp)

- SQL - exists → True / False

1) Generalized Projections

extends projection operation by allowing functions of attributes to included in the projection list.

$\Pi_{ename, sal}$ (employee)

$\Pi_{ename, sal - 0.1 * sal}$ (emp)

↓ operations can take place in generalized projections

DBMS

2) Aggregate functions

- Common functions include SUM, AVERAGE, MAXIMUM, MINIMUM
- for SUM to function properly, data type must be integer.

display all those departments where average salary is greater than 50,000

Emp (eno, ename, Dno, sal)
Dept (dno, dname, loc)

o/p => Average (dno, dname) > 50,000

SELECT $\Pi_{AVERAGE(sal)}$ (Emp) } it will compute the average of all the employees of every department

Π_{dno} (dept no) } contains all the department no

3) GROUPING

- grouping tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.

$$\rho_R(\text{DNO}, \text{no of emp}, \text{avg sal})\ (\text{D.NO}\ \mathcal{F}\ \text{count}(\text{eid}^{*})\ ,\ \text{average}(\text{sal}))\ (\text{emp})$$

[CR]

<grouping attribute> <f^n list>

- ρ_R : rename operation
- D.NO : this will give the answer for the above question.
- it will result in a table.

D.NO.	no-of-employees	av.sal

- In general, duplicates are not eliminated when an aggregate function is applied.
- NULL values are not considered in the aggregation.

Recursive Closure Operations

- applied to recursive relationship between tuples of same type.
  eg: relationship between employee and manager

- Report all reportees of an employee at all levels
  i.e all emp e' directly supervised by e
  and all emp e'' directly supervised by e' and so on.

Eid	Ename	Dno	Sid
E1			E10
E10			E34
E34			E107
E107			X
E2			E34
E202			E34

direct
E_{34} direct reportees
E10, E2, E202
and then it will pull all the direct reportees of E10, E2, E202

- easy to specify all employees supervised by e at a specific level by joining table with itself one or more time.

However, difficult to specify all supervisors at all levels.

Jones_id ← π_{eid} ($\sigma_{name='Jones'}$ (emp))

Supervision (eid1, eid2) ← $\pi_{eid,sid}$ (emp)

Result (eid) ← (Supervision $\bowtie_{sid=eid2}$ (Jones))

Supervision

EID1	EID2
E1	E10
E2	E34
E10	E34
E34	E107
E107	—
E202	E34

Jones_id

eid
E34

For second level

RESULT2 (eid) ← $\pi_{(eid1)}$ (Supervision $\bowtie_{eid=eid}$ Result)

Outer Join Operations

- Join Operation match tuples that satisfy the join condition
- Type of join where tuples (rows) with no match are eliminated

is known as inner join.

Outer join → user wants to keep all the tuples in R or all those in S, or all those in both relations in the result of the JOIN regardless of whether or not they have matching tuples in other relations.

→ R ⟕ S left outer join ⇒ all the rows of R must be included irrespective of whether they satisfy the condition or not.

→ R ⟖ S ↳ right outer join (all the rows of S are included)

→ R ⟗ S ⇒ full outer join

- List of all emp names as well as names of the department they manage if they happen to manage a dept. If they do not manage one, we can indicate with a NULL value.

Emp

Eid	ename	–
e1	–	
e2	–	
e34	–	

Dept

DNO	Dname	Mg'id
D1		e2
D2		e3
D106		e2

Empl / ~~IX~~ ~~Dept~~ ~~%~~ / ~~first~~ ~~part~~

$\Pi_{ename,\ deptname}$ (Emp ⋈ Dept) (eid = mgrid) → for all the managers of the department.

$\Pi_{ename,\ deptname}$ (Emp ⟗ Dept) (eid = mgrid)

Outer union operation

→ to take union of tuples from 2 relations that have some common attribute but are not union compatible.

R(x, y) and S(x, z) R and S are tables

↳ Columns of table R

- Tuples are partially compatible

- attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation.

T(x, y, z)

SQL (Structured query Language)

- non procedural language (declerative language)
- SQL is classified as

1) Data Definition Language (DDL) - Create, Alter, ~~Attribute~~, drop, truncate
2) Data Modification Language (DML) - Insert, Update, delete
3) Data Control Language (DCL)
4) Transaction Control Language (TCL)

DDL operations are auto commit (operations can not be rolled back) table once created can be dropped but the CREATE TABLE command can not be rolled back.

DROP table deletes the entire table, whereas in TRUNCATE deletes the entire contents of the table, the table still remains.

Select is not ~~count~~ a DDL or DML & it is just used for query

DML statements are not auto commit.

Even delete operations can be undone. (it can be rolled back)

DBMS

- process of efficiently organizing data in a DB.
- a 2 step process =>
  1) puts data into tabular form by removing repeating groups
  2) removes duplicated data from the relational table.

Anomalies =>

1. insertion anomaly
2. updation anomaly
3. deletion anomaly

Insertion anomaly (leads to updation anomaly)

emp

Eid	Fname	Lname	pid	pname	Ploc
Ex	—	-	NULL	NULL	NULL

→ split it into two tables

Suppose an employee x has been recruited but has not yet been assigned some project

the pid, pname, ploc will all be NULL in this case

1 or 2 columns are acceptable, but here we have many columns with NULL value.

Updation anomaly => happens when data is redundant. (values are repeated many times)

Eid	Fname	Lname	City	Pincode	
—			Bombay		
—			Bombay		

↓

if name of city changes from Bombay to Mumbai

Some one has changed the city from Bombay to Mumbai

Other have not → Updation anomaly

Deletion anomaly

emp	fname	lname	Did	Dname
~~612~~			D2	

↳ if there is only one employee of a department and we delete that, the information pertaining to the department is also lost forever.

Normal forms

A table is said to be in a normal form if it satisfies a set of constraints.

6 Normal forms

1) First Normal Form (1NF)
2) Second " " (2NF)
3) Third " " (3NF)
4) BCNF
5) Fourth (4NF)
6) ~~Fifth~~ 5NF

item no	item-desc	price	manufacturer name	distributor id	distributor name	order id	date executed	qty
watches								

There may be multiple order of watches from the same distributor

First Normal Form — No repeating groups (multiple values / composite values) and all attributes are dependent on PK

When we have a multivalued attribute we create another table.

Eg:

Roll NO	Name	Aadhar Card	Phone no

→ multivalued.

Roll NO	Name	Aadhar Card

Roll NO	Phone NO

for composite form => we make different columns
Eg: name is broken into first name, second name, middle name.

we go by the assumption that tables are in first NF.

↓ primary key

Pid	Pname	Loc	Rev.	Mg. id	Client id

all attributes must be dependent on primary key. (by default satisfied)

if we write
```
SELECT *
FROM PROJECT
WHERE Pid = 'P01'
```
we will get a single value.

if value of project id (primary key) is known, all other values can be found out, and ~~key~~ it will be unique. → functionally dependent

Second Normal form (2NF)

if a table is in 1NF and all attributes are fully functionally dependent on all the candidate keys.

↗ non key (not part of even candidate keys)

Fully functionally dependent

for a given relation R, attribute B of R is fully functionally dependent on attribute A of R, if it is functionally dependent on A and not on any subset of A.

Order - details

Oid	Prid	Prname	Manufacturer date	Price
01	P1	Soap		
01	P_{10}	Shampoo		
01	P_6	Bread.		

M'fg date and price are fully functionally dependent

primery key — non key attribute.

Prname is not dependend on Oid and Prid.

it only needs Prid. to determine it's value uniquely

it is not fully dependent. (it is dependend only on a subset of primery key)

functionally

is not in

We need to split this table, because, this table in 2NF.

(data is redundent over here)

~~to bring it in~~

now 2NF of this table is.

Oid	Prid	Mfg date	Price

and

Prid	Prname

Oid — primery key (now primery key is both is not composite)

Prid — primery key.

now they are in 2 NF

Transaction

C#	Cname	C City	Cphone	P#	Date and time	Quantity

C => customer P => product.

Cname, C City, Cphone depends on C#

C# + Date and time => primery key =>

we will get all other columns (name, city, cphone, P#

Scanned with CamScanner

2nd normal form of the table

C#	Cname	Cphone	Ccity

C#	P#	Date and time	quantity

Transitive Dependency (column)

- value in a non-key field is determined by value in another non-key field and that field is not a candidate key.

$(a,b)\ (b,c) \Rightarrow (a,c)$

non-key → non-key

$A \rightarrow B$

primary key →

$A \rightarrow B$

$PK \rightarrow A$

$\Rightarrow PK \rightarrow B$

↑ primary key.

Project No	Project Title	Project Manager	Phone

→ a project can have only one manager.

Project Mgr → Phone

Project No → Project Title

Project No → Project Mgr

Project No → Phone

∴ Project No is a primary key.

∴ there is transitive dependency.

able is in 2 NF and

all it's attributes are dependent on ~~key~~ only on the key attributes (primary key / candidate key)

ct Mgr is a non key attribute. ⇒ there is transitive dependency.

⇒ the table is not in 3NF.

key attributes should be mutually independent.

the table.

ject No	Project Title	Project Mgr

Project Mgr	Phone

2nd normal form of the table

C#	Cname	Cphone	Ccity

C#	P#	Date and time	quantity

Transitive Dependency

value in a non-key field is determined by value in another non-key field and that field is not a candidate key.

(a, b) (b, c) => (a, c)

non-key → non-key

A → B

~~primary key~~ →

A → B

PK → A

=> PK → B

↑ primary key.

Project No	Project Title	Project Manager	Phone

a project can have only one manager.

Project Mgr → Phone

Project No → Project Title

Project No → Project Mgr

Project No → Phone

} as Project No is a primary key.

then is transitive dependency.

table is in 2 NF and all it's attributes are dependent on ~~key~~ only on the key attributes (primary key / candidate key)

Project Mgr is a non key attribute. => there is transitive dependency.

=> the table is not in 3NF.

non key attributes should be mutually independent.

the table.

Project No	Project Title	Project Mgr

Project Mgr	Phone

Normalization questions

(1) a) We have composite branch address which can be broken ~~into~~ into.

B.NO	B. A Line 1	B. A Line 2	B. Add L 3	City	State	Pincode

↓ primary key

Telephone is a multivalued column, we create one more table.

B.NO.	B.Tel

→ primary key

it is in first normal form

Since primary key is not composite.

=> no case of functional dependency

it is in 2nd normal form

now for 3rd normal form => check for transitivity ⇒ dependency

A → B, (B) → C => A → C

↓ if B is a non-key attribute.

~~City~~ Pincode → City

Pincode → State

now to bring it into 3rd normal form table is now ~~one~~ =>

↓ primary key ↓ foreign key

B.NO.	B A L1	B A L2	B A L3	Pincode	~~City~~ ~~State~~

↓ primary key

and

Pincode	City	State

B.NO	B.Tel

→ primary key

↓ foreign key

(2) ~~Primary key~~ Staff no # branch no => Hour / week

↓ primary key

B.NO → B Add

Staff no → Name, Position (when position of staff does not change irrespective of branch)

↓

=> ~~functional~~ partial dependency on primary key

not in 2nd normal form

B.NO	B Add

pk

Staff no	Name	~~Name~~	Position

pk

Staff NO	B.NO	Hours/week

pk

transitive dependency => all 3 are =
=> it is not in 3rd normal case.

Now: one staff is working in different branches at different position.

B NO → B Add

Staff NO → Name

Staff NO, B NO → hours/week, position

B NO	B Add

(3) ~~Branch No → B Add, Tel No, Mgr Staff No, Name.~~

Manager can supervise many branches

B NO is the primary key.

B NO → B Add, Tel NO, Mgr Staff No, Name.

Mgr Staff NO → Name

=> it is not in 3rd normal form

B.NO	B Add	Tel No	SMS NO.

SMS NO	Name

DBMS

18/02

Integrity Constraints

- primary key
- not null
- unique
- default
- check

```
(create table emp
(   name  varcher  not null,
    PAN,  varcher  unique
    active . varcher default 'Y',   check in ("Y", "N")
)
```

all the values with default should be towards the right end, otherwise there will be confusion as to which the value has to be mapped.
otherwise we have to do explicit mapping.

insert into emp values ('ABC', 'AXL123IP')
↓ name ↓ PAN and active by default 'Y'.

Check can be used to check whether the ~~tuple values~~ values of the columns should be from the specified one. (example for active ~~check~~ the column can contain only 'Y' and 'N'.

Referential Integrity (for foreign key)

D.NO.	D.Name	Loc.

Dept

ENo	EName	D.NO (F.K).

Emp

If Department nos is D1 to D10 in Dept table
=> Dept No of Emp table should also be between D1 to D10
Any other dept is not valid.
Dept nos should be from the Dept table only. (but we can have NULL in the D.NO of Emp table)
↑ in the Emp table

If there are many ~~dept~~ values of D2 in the Emp table.
and if D2 is deleted from Dept table
all the values with D2 department in Emp table will not be valid anymore.
1 option is to assign department value to be NULL or (in Emp table)
other option is to delete all the values with department D2.

DBMS

18/02

Integrity Constraints

• primary key • not null • unique • default • check.

```
(create table emp
 (  name varcher not null,
    PAN , varcher unique
    active . varcher default 'Y',   check in ("Y", "N")
 )
```

all the values with default should be towards the right end, otherwise there will be confusion as to which the value has to be mapped.
otherwise we have to do explicit mapping.

insert into emp values ('ABC', 'AXL123IP')
name → PAN and active by default 'Y'.

Check can be used to check whether the ~~tuple values~~ values of the columns should be from the specified one. (example for active ~~check~~ the column can contain only 'Y' and 'N'.

Referential Integrity (for foreign key)

D.NO.	D.Name	Loc.

Dept

ENo	EName	D.NO (F.K).

Emp

If Department nos is D1 to D10 in Dept table
=> Dept No of Emp table should also be between D1 to D10
Any other dept is not valid.
Dept nos should be from the Dept table only. (but we can have NULL in the D.NO of Emp table)
↑ in the Emp table

If there are no ~~to~~ values of D2 in the Emp table.
from Dept table
in Emp table will not be valid anymore.
to be NULL or (in Emp table)
values with

Triggers (action leads to immediate reaction)

If D_6 is changed to D_{24} in the Dept table,
then the same should be changed for all rows with D_6 department in Emp table also

- Create Trigger => used to implement actions in SQL

create Trigger <Trigger_name>
Before / or after insert or update of Salary, supervison_id, on Emp
update Salary = 0 where supervison_id = 134

Trigger is regarded as event-condition-action (ECA) has 3 components.

1) Event(s) - usually DB update operations that are explicitly applied to the DB (insertion, deletion or updation)
- specified after keyword Before / After

2) Condition - determines whether the rule action should be executed

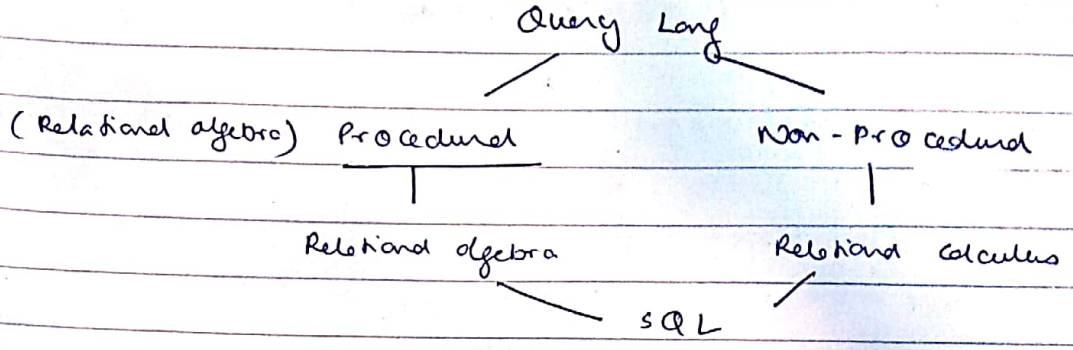
Once the triggering event has occured, an optional Condition may be evaluated. If no condition is specified, action will be executed once the event occurs.

- If a condition is specified it is evaluated first, and only if it evaluates to true will the rule action be executed.

The condition is specified in the where clause of the trigger

3) Action to be taken, usually a sequence of SQL statements.

Tuple Relational Calculus



Relational Calculus

Tuple Relational Calculus

Domain Relational Calculus

- user is concerned with the details of how to obtain the end results
- uses mathematical predicate calculus
- a predicate is a truth valued function with arguments.
- when we replace with values for the arguments, the function yields an expression called a proposition, which will be either true / false.

Tuple Relational Calculus

- write one declarative expression to specify a retrieval request.
- The expressive power of Relational algebra and relational calculus is identical

Tuple variables and Range relations

- TRC is based on specifying a no of tuple variables
- each tuple variable usually ranges over a particular DB relation.
- the variable may take as it's value any ~~instance~~ individual tuple from that relation.

→ {t | COND(t)} => t variable should satisfy the condition.

{t | Emp(t)} ~~for all the~~ all those rows which belong to Emp table.

{t | Emp(t) AND t.sal > 50000}

all those employees (entire row) of the employees whose salary is greater than 50000

=> t.sal ≈ t[sal]

You need to specify the following information in a TRC expression =>

1) to each tuple variable t, the range relation R of t. R(t)

because if we specify only {t | t.sal > 50000}, it will span all ~~the info about salary~~ of the entire database.

2) a condition to select particular combination of tuples.

3) Requested attributes. (we could retrieve only the specified columns)