Introduction

- Data today is the most valuable resources.
- Database is a collection of interrelated data.
- Database Management System is a software which helps us manage the data.

- Database System = Database + DBMS

- During 1960's first DMS came into picture which used N/W model.
IBM then started using Hierarchical model.
Later E. Codd came up with "Relational Model".

- In Relational Model data is stored in tables, so it is a very natural & intuitive model.

Although many other models have emerged & are used since, Relational Model is still the dominant model after almost 50 years.

- We store our data (small) using File System, but to store large amount of data, file system is inefficient.
One of the major problems with File Systems is Redundancy.
Another is inconsistency. File Systems aren't built for concurrent access, and this can lead to a lot of problems. FS doesn't provide a lot of features relating to restricting data access.
FS can't force constraints on the data stored.

DBMS can take care of all this and more.

* Database Design Process

When a customer comes with a requirement for a DBS, then the steps for its design would be :

(1) Requirement Analysis Process : Gather info.

(2) Convert the req. to some user friendly model (such as ER model) : Conceptual Design.

(3) Logical Design : Convert conceptual design to logical design using some logical model (such as Relational model) (implementation friendly)

(4) Physical Design : Implementation of logical design.

* Just like there are multiple models to represent Regular Languages, each with its own convention, notations, etc.

Similarly data can be represented using various models such as Relational, Object-Oriented, Graph, Document, ER, EER, SQL, etc.

DBMS can be based on many models. We study the most famous : Relational DBMS.

Relational Model & Normalization

- Relation ≡ Table.
- Relational Model: Table / Relation based model to represent the database.

- E.F. Codd in his research paper of "Relational Model", gave:
  1. How to store the data.
  2. Terminology / definition (Keys, 2NF, 3NF, BCNF, constraints, etc).
  3. Query lang. to retrieve data: Relational Algebra & Relational Calculus.

- Later, based on this a practical query language was created by IBM, Structured Query Language (SQL).

Ex. Student

S-no	Name	DoB	Phno
1	Rax	3/8/3	0123
2	Dax	2/2/2	4567

Student is the Name of the relation.
(Sno, Name, DoB, Phno) are the Columns or Attributes or Properties.
Each row is called Data or Tuple.

- Every attribute in a relation has a domain, which specifies the set of values that any tuple can have for that attribute.

* Domain of Attribute

For any attribute its domain is the set of "atomic" (indivisible) values, that the tuples can have for that attribute.

Ex. Say Name attribute is having Domain: String.

...	Name	...
	Amit Kumar	

Can you retrieve just the surname? No. Only the whole name can be retrieved.

Ex. Say PhNo. is Domain: Integer

...	Ph No	...
	357, 312	

Is This valid? No. It's not atomic for the domain of PhN, ie. Integers.

So domain is the set of any & all ~~values~~ atomic values that the attribute can take.

* Schema vs Instance

- Schema is the structure / outline of the table.
- Instance is the snapshot of the table.

Ex. For Student Table:

Student (
S.No: Integer,
Name: String,
DoB: Data,
Ph.No: Integer
)

↑ Schema

Sno.	Name	DOB	PhNo.
01	ABC XYZ	01/1/2	123
02	DEF WXY	3/4/15	456
03	GHI JKL	6/7/18	788.

↑ Instance

So, Schema consists of Table name, column names & column domains. Instance consists of the tuples of the table.

- Degree (or arity) of a relation is the no. of attributes of its relational schema.

- Cardinality: Number of rows in any given instance of a relation.

* Why call a Table as Relation ?

- In mathematics a rel. from set A to B, $R: A \rightarrow B$, is a subset of $A \times B$, i.e., $R \subseteq A \times B$. This is a binary rel^n.

- Similarly a ternery rel^n can be def^n as: $R \subseteq A \times B \times C$.
Eg Say, A is set of all integers, B is set of all countries, C is set of all players of Cricket.

$$A \times B \times C = \{(a,b,c) \mid \forall\, a \in A, b \in B, c \in C\}$$

↳ 3- tuple.

In this all possibilities would be there, say, (01, Rahus, VK). But we only want specific data which correctly represents

reality.

Say reality is: (a,b,c) means 'a' is the rank of highest run scorer in ODI, 'c' & their country 'b'.

So we can def. R: (a,b,c) ∈ R iff

- We can extend this to n-ary relation : $R \subseteq A_1 \times A_2 \times \dots A_n$

$A_1 \times A_2 \times \dots A_n = \{(a_1, a_2, \dots a_n) \mid a_i \in A_i\}$

($a_1, a_2, \dots a_n$) : n-tuple.

- So, in a table, say,

Domain:	D_1	D_2	D_3
	Sr. No.	Name	Ph No.
r_i →	01	ABC_EFG	0123

every tuple $r_i \in D_1 \times D_2 \times D_3$

- Hence we call it Relation since it is actually a rel.

① In above example, Student $\subseteq D_1 \times D_2 \times D_3$.

② Every record $r_i \in D_1 \times D_2 \times D_3$.

③ Every instance I (set of records), $I \subseteq D_1 \times D_2 \times D_3$

- So, table is just a layman term we informally use, it actually is a relation.

Ex. Rel. $R(A_1 : D_1, A_2 : D_2, A_3 : D_3)$

$|D_1| = 2 \quad |D_2| = 3 \quad |D_3| = 3$

Max cardinality possible for an instance?

$2 \times 3 \times 3 = 18$. And the instn is $D_1 \times D_2 \times D_3$.

No. of possible instances?

$\equiv$ No. of subsets of $D_1 \times D_2 \times D_3$

$\Rightarrow 2^{18}$

* Null

Ex. Suppose a student relation is to be designed with the schema:

Student (s_name : string, PassportNo : string, Gender : string)

Now, it is very much possible that a student doesn't have Passport no. Then what do we put here?
We put Null.

- So NULL can mean, "not available". (It may/maynot be available in future, but that doesn't matter).

Ex. Say we have Employee schema:

Employee (Id : string, Name : string, PasspNo : Str, SpouseName : string)

It may happen that an employee doesn't have a Spouse.

- So Null can also mean, "Not Applicable".

- So NULL is used to represent:
  1. Not Known
  2. Not applicable
  3. Not recorded.

  etc.

Note: Every attribute's domain contains NULL. It can/may not be allowed always.

* . As is the property of relations, order doesn't matter & no duplicates allowed.

- So in any DB Relation, order of tuples doesn't matter, and no two rows can be exactly same in Relational Model.
- Order of attributes matters in Relational Model.
- Although NULL is part of domain of every attribute, we can restrict its use for attributes.

* Keys

- The purpose of storing data is to retrieve them later. But how do we optimize this. We want something to be an attribute which is unique for every tuple.
  (or set of attributes)

So we can query based on this attribute (set of attributes).

Ex. Identity nos, such as Aadhar, PAN, Social Security No, uniquely identify citizen of a country.

- So, a Key uniquely identifies each tuple in a relation.

- SuperKey: A set of one or more attributes which uniquely identify each record in a relation.

  Say S is superkey, and R is rel.

$$\forall r_i, r_j \in R \quad \exists a \in S \quad a(r_i) \neq a(r_j)$$
$$i \neq j$$

Ex. Student (S.name : String, State : Str, Dist: Str, PIN : Int, HouseNo : Int)

{sname}, {State}, {Dist}, {PIN}, {HouseNo} can't uniquely identify.
{Say House no. is unique in a PIN code}.

So,

S = {PIN, HouseNo.} can uniquely identify.

Then all supersets of S can also uniquely id. rows.

- Superset of a superkey is also a Superkey.

- Can we guarentee that every relation has at least one superkey?

Yes. Since every two record must differ in at least one position. So in worst case set of all attributes will act as Superkey.

Ex. In our previous example suppose the superkey : {PIN, House No, S_name, State}

If we remove S_name, it is still Superkey.
If we remove State, it is still Superkey.

If we now try to remove either PIN or HNo, it won't be Superkey anymore.
So we can say {PIN, HNo} is a minimal superkey.

- Candidate Key: A minimal superkey is called Candidate Key. Say C is a CK, then

  $\forall$ rec C - {a} is not superkey.

- Primary Key: From all possibe Candidate Key, one of the CK is chosen as Primary Key. (It's a design choice, depends on the data stored in the relation)

- Candidate Key [In case of Null Values]: Should identify all non-NULL records uniquely, and must be minimal.

- A candidate Key with NULL value can't be chosen as Primary Key.

If PK comprises of multiple attribute, then no attribute of PK can have NULL values.

Any relation in Relational model, must have exactly one PK at any point of time. It can change over time, but must be exactly one of the possible CKs.

- Simple CK: Single attribute CK.
  Composite CK: Two or more attribute CK.

- Alternate Keys: Out of all CKs one is chosen at a time to be PK. All other remaining CKs are called Alternate Keys.

Superkey	Candidate Key	Primary Key
• Should uniquely identify tuples	• Should uniquely id tuples.	• Should uniquely id tuples.
	• Should be minimal	• Should be minimal
		• No attr. in it should have Null value, for any tuple.

Note: Generally the term Key means Candidate Key.

Ex. Given some rel. R(a,b,c,d,e);

① what is the max. no. of superkeys possible?

$\sum_{i=1}^{n} (-1)^{i+1} \; {}^{n}C_i \times 2^{n-i} \Rightarrow \sum_{i=1}^{n} (-1)^{i+1} \; {}^{n}C_i \times 2^{n-i}$

$\Rightarrow 2^n - 1$ (when every attr is CK)

$\uparrow \phi$ (every non-empty selection of attr is SK)

② What is the max. no. of CK possible?

$\Rightarrow {}^{n}C_{\lfloor n/2 \rfloor}$.

Ex. For R(a,b,c,d,e) CKs = {a, bc, bde}

No. of SKs?

For $a = 2^4$ For $bc = 2^3$ For $bde = 2^2$

For $abc = 2^2$ For $abde = 2$ For $bcde = 2$

For $abcde = 2^0 = 1$

$\therefore$ #SKs $= 2^4 + 2^3 + 2^2 - 2^2 - 2 - 2 + 1$

$\Rightarrow 21$

Ex. R($a_1, a_2, \ldots, a_8$). Assume we know $a_1a_2a_3$ is a CK. Other CKs are unknown.

(a) Max no. of SK possible?

(b) Max. no. of CK possible?

① Since $a_1a_2a_3$ is CK, then none of its proper subset can be key. So no. of possible max. SKs:

$$2^8 - (2^3 - 1) - 1 \quad (\phi)$$

$$\Rightarrow 256 - 8 = 248.$$

② W.K.T. max. no. of CKs is given by ${}^{n}C_{\lfloor n/2 \rfloor}$ but any $\lfloor n/2 \rfloor$ len combination ~~which contains~~ which contains CK $a_1a_2a_3$, can't be CK itself.

So, max possible CKs:

$${}^{8}C_{\lfloor 8/2 \rfloor} - ({}^{5}C_1) + 1$$

($a_1a_2a_3$ is CK already)

(all 4 len comb. with a_1, a_2, a_3 already in it.)

$$\Rightarrow {}^{8}C_4 - {}^{5}C_1 + 1$$

*. Database Instance: Collection of all relation's instance in database.

*. SQL: A language for querying on RDBMS software.
Different vendors have their own variants.
So we follow the standard SQL-92.

SQL has two parts:

Deals w/ Schema ← ① Data Defⁿ language (DDL): Used to define/create/deletion of relation.

Deals w/ Data. ← ② Data Manipulation Language (DML): Used for various purpose, but we only study Query Language which is used for retrieval.

- Although SQL & Relational Model appear to be very similar, Relational Model is a theoretical model, (i.e., it is defined on paper) and it is mathematical model (i.e. it has inflexible set of rules). SQL is loosely based on RM, and is a practical language and can vary a lot from Relational model.

SQL	RM
• Treats table as multiset of row/records, i.e. duplicate rows allowed.	• Treats relation as set of rows/records, i.e. duplicates not allowed.
• Primary Key not mandatory.	• PK mandatory.

* Integrity Constraints over Relations.

- Domain Integrity Constraint: For a attribute in the relation what is the set of all possible values that the attribute can take.

- Key Integrity Constraint: For any two rows in the relation the attribute a_i must be different. (a_i would be valid CK)

- Entity Integrity Constraint: For any row in the relation the attr. a_i must be unique & not null.
  (This would also make a_i a valid Primary Key)

- Ex. Suppose the following rel.

SId	SName	District	State
101	ABC	Kanpur	UP
102	DEF	Bangalore	Karnataka
103	GHI	Hyderabad	Telangana
104	JKL	Kanpur	Bihar

It is only logical to have the unique State for a district. Say for Kanpur, we should only have UP, and not Bihar.
How can we implement such constraint.
None of above constraints talk about multiattr constraints.
We can observe here: District determines state or
District implies State attr or
State is dependent on district.
Although State doesn't determine District.
Ex. Patna being district means state is definitely Bihar.
But Bihar being state doesn't necessarily mean Patna district.

- This type of Integrity Constraint is called Fundional Dependency constraint. This is between attributes (within a table or b/w tables).

Ex. Student-Course

S-id	C-id
101	CS120
102	CS122
103	CS125
104	Null

Course

C-id	C-name	Inst.
CS120	Os	Vishal
CS121	DBMS	Deepak
CS122	Algo	Sachin
CS123	DM	Deepak

Here for Sid 103, CS125 is there but that course isn't available. For SID 104, C-id is Null, which is fine, they might not have enrolled in any course.

So we can say, all the non-null values in Student-Course relation's C-id must come from Course's C-id.

- This type of IC is called Referential IC.
(Foreign Key constraint).

It can be within a table or within two tables.

* Domain Integrity Constraint

The relational schema specifies the domain of each attribute in the relation. Each instance of the relation for the attributes must have values from the specified domains.

Sometimes additional constraints are required on the domain.

Ex.
```
Student (
        S_id   int,
        Age    int
)
```

But this doesn't prevent neg. value, and erroneous age value.
SQL-92 does allow this using CHECKs.
(In Relational Model, this is implicit)

* Key Integrity Constraint (Uniqueness Constraint)

If α is CK then two tuples can't have same non-null value of α.
(In SQL this is acheived using "Unique" keyword)

* Entity Integrity Constraint (Primary Key Constraint)

In Relational Model every table must have PK.
If α is PK then no tuple can have null or duplicate value for α.
With this constraint the aim is to identify every entity uniquely.

* Referential Integrity Constraint (Foreign Key Constraint)

An attribute x referring to some other attribute y must have values which are currently there in y.

Ex. Student(S-id, Name, Cid)
Course (Cid, Cname, Instructor)

Student.Cid is referencing Course.Cid.
So, it should only take values which Course.Cid currently has.

Student here is called Child table & Course is Parent table based on Student.Cid attribute.
Student.Cid is foreign key in Student table, because it is pointing to a key in a foreign table.

Ex. student(Cid, name, cid) Course(id, name, monitor)

This is also possible.

- Ex. Employee (id, name, manager)

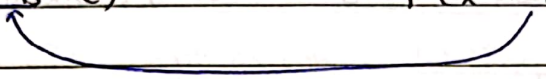
- We can't allow a non-Key attribute to a be referenced by some foreign key.

So, foreign key always references CK.

- Foreign key may/maynot be CK in its table but it always references a CK.

- It may/maynot be null. Depends on design, and requirement.
- FK can consist of more than one attr. as well.

Ex. R(A B C) T(X Y Z)



So, YZ is the FK refering to CK AB.
(It's not same as Y refering to A & Z refering to B)

- FK can also point to CK within same table.

Ex. Suppose the following DB instance:

A:	a	b	c
r_0	2	1	3
r_1	1	2	3
r_2	9	2	2

B:	a	y	z
r_0	2	1	3
r_1	1	2	1
r_2	3	2	1

c references x

- If A.r_0 is deleted, is there problem? No.
- If B.r_2 is deleted, is there problem? Yes. A.r_0 & A.r_1 would be referencing to invalid value 3. So violates Ref. IC.
- If a row (5, 1, 9) is added to A? Yes. 9 is not in B.x. Violates Ref. I.C.
- If a row (5,1,9) is added to B? No problem.

In SQL-92, FK can be specified while creating table.

Ex. Enrolled (sid CHAR(20), cid CHAR(20),
grade CHAR(10),
PRIMARY KEY (sid, cid),
FOREIGN KEY (sid) REFERENCES student)

{ Suppose Studend (Sid: CHAR(20), Sname....)

Since we haven't specified which CK from Student table is to be referenced, by default PK would be used, ie. Student.Sid.

Note: In SQL 92, all such things as, PK, FK, domain constraint, Key constraint can be specified using CONSTRAINT Keyword.

Ex.

```
CREATE TABLE User ( id: INT,
                    name: VARCHAR(30),
                    email: VARCHAR(50),
                    group: INT,
    CONSTRAINT user-pk PRIMARY KEY(id),
    CONSTRAINT group-fk FOREIGN KEY(group) REFERENCES Group,
    CONSTRAINT Email-key UNIQUE (email),
    CONSTRAINT id-domain CHECK (id >0),
)
```

Keyword — name — definition

If constraint is violated, error would show will constraint name.

- Violation of Referential IC happens when:
  Suppose R.x references S.y

  ① Insertion to R.
  ② Deletion from S
  ③ Updation to R
  ④ Updation to S

- If violation happens when delcting/updating in referenced table then the possible actions are:

(1) Cascade: Delete all such records in referencing table which refered to the particular record

(2) Set Null: Set the referencing field to null, when referenced value is deleted.

(3) Set Default: Set some default value.

(4) Restrict: Prohibit the violating operation.
("No Action" in SQL-92)

In SQL-92 we can specify ON DELETE <ACTION>, for each reference.

Ex. FOREIGN KEY(Sid) references Student ON DELETE CASCADE

Note: In Schema representation of a relation attribute underline means PK.

Ex. Enrollment (Sid, Cid, Grade)

So, Sid Cid is PK.

* Violation of IC by operations.

Constraint	Operation	Violated	Remark
Domain	Read	No	
	Add	Yes	Out of domain value added
	Delete	No	
	Update	Yes	—"—
Key	Read	No	
	Add	Yes	Duplicate value.
	Delete	No.	
	Update	Yes	—"—
Entity	Read	No	
	Add	Yes	Null or Dupl. Value
	Delete	No	
	Update	Yes	—"—
Referential	Read	No	
	Add	Yes	In Referencing Relation
	Delete	Yes	In Referenced Relation
	Update	Yes	In both relation.

* Functional Dependencies

Ex. We already saw an example earlier where:
District determined State OR
State was dependent on District.
i.e.

District ⟶ State.
(similar to implication in Logic.

- The functional dependency $\alpha \longrightarrow \beta$ on a relation R means.

$$\forall_{\substack{T_1, T_2 \in R \\ T_1 \neq T_2}} \quad T_1.\alpha = T_2.\alpha \longrightarrow T_1.\beta = T_2.\beta$$

i.e. for all rows T_1, T_2 in R ($T_1 \neq T_2$) if value of α is same for them then β value must also be same.
where, $\alpha, \beta \subseteq$ R.Attributes & $\alpha, \beta \neq \phi$

- Similar to other constraints the DB owner would specify the FDs in the DB.
For example the owner may want District to determine State. or may want otherwise.
Ex. {Aadhar No} ⟶ {PAN No.}
also {PAN ⟶ Aadhar}
Ex. {Grade} ⟶ {IsPass}
but {IsPass} ⟶ {Grade}

(Note: Think of $\alpha \rightarrow \beta$, as, if you know α, you should uniquely know β automatically)

Ex. {Class, Roll} → {StudentName}

(or CR → N)

Here CR → Name, but

Class by itself doesn't imply Name

& Roll by itself also —"— ——

(by design & also logically)

Ex. R:

A	B	C	D
2	2	5	1
2	2	5	1
2	3	4	2
2	3	4	1
4	3	6	1

In this instance / snapshot:

(A) AB → C (C) A → C

(B) AB → D (D) B → C.

A only.

In relation R:

(A) AB → C (C) A → C

(B) AB → D (D) B → C.

B, C, D are false.

A may/maynot be true. For an ~~Functional~~ FD to hold, it must hold for all instances of relation. For it to violate, it only needs one tuple.

- $\forall \alpha \subseteq_{R} .\text{Attr}, \alpha \neq \phi \quad \alpha \rightarrow \alpha$ (Trivial).
- FD is defined on a relation schema R, (not on instance), so for all instances r(R) of R, it must hold.

For an FD to be violated a single counterexample row is enough.

(So from a given instance r(R) of relation R, we can comment on the FDs that don't hold for R, but not the ones that hold for R. except trivial ones.)

Ex. Consider R(A, B, C, D, E) with FDs:

$A, B \rightarrow C$

$C, D \rightarrow E$

Suppose there are atmost 3 diff. values for each A, B & D. What is max. possible no. of diff. values of E?

Suppose for 3x3 comb. of AB we have all unique C.

a_1 b_1 c_{11}

a_1 b_2 c_{12} So 9 unique Cs.

:

a_3 b_3 c_{33}

So, 9x3 comb. of CD, & hence we'll have 27 unique Es.

So, 27 is max possible no. of diff. values of E.

Note: The concept of FDs were originally defined with no Null values.

- $\forall_{X,Y,Z \subseteq R, \wedge Hr \atop X,Y,Z \neq \emptyset} \quad (X \to Y \wedge Y \to Z) \longrightarrow (X \to Z)$

[Transitive Rule]

- $\forall_{X,Y \subseteq R, \wedge Hr}$ Say $X = \{x_1, x_2, \ldots x_n\}$, $Y = \{y_1, y_2, \ldots y_n\}$

① If $X \to Y$
then, $X \to y_1 \wedge$
$X \to y_2 \wedge$
$X \to y_3 \wedge$
$\vdots$
$X \to y_n \wedge$

② If $X \to y_1 \wedge X \to y_2 \ldots \wedge X \to y_n$
then $X \to \{y_1, y_2, \ldots y_n\}$.

[Splitting / Combination Rule]

Basically on RHS of FD we can split (when more than one attr. on RHS) and combine (if LHS same) as well.

- We can't split on LHS.

Counterexample:

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2

Here, $AB \to C$ but neither $A \to C$, nor $B \to C$.

We can combine on LHS.

i.e, $(A \to C \wedge B \to C) \longrightarrow (AB \to C)$

Proof: Say $A \to C$ & $B \to C$, but $AB \not\to C$

A	B	C
a	b	c_1
a	b	c_2

but then $A \not\to C$ & $B \not\to C$ here

∴ Contradiction.

- Trivial FDs: FDs that always hold irrespective of the Relation.

For $\alpha, \beta \subseteq R.Attrs$, $\alpha, \beta \neq \phi$

$$\alpha \longrightarrow \beta$$

always holds if $\beta \subseteq \alpha$

- $\forall_{X,Y,Z \subseteq R.Attrs,\ X,Y,Z \neq \phi}$ $\quad (X \to Y) \longrightarrow (XZ \longrightarrow YZ)$

$XZ = X \cup Z$, $\quad YZ = Y \cup Z$

Proof: Say $X \to Y$ holds & $XZ \to YZ$ doesn't hold.

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_1

but then $X \to Y$ doesn't hold.

∴ Contradiction.

- If $X \to Y$; $WY \to Z$ then $WX \to Z$.
- If $X \to Y$; $W \to Z$ then $XW \to YZ$

} Proof by Contradiction }

(Trick to remember, Think of $X \to Y$ as X beats Y or X is more powerful than Y)

Closure of Set of Attributes:
(Attribute Closure)

For a set of attributes X, its closure is given as:

$$X^+ = \{ w \mid w \in R.Attr,\ X \rightarrow w \}$$

Ex. Suppose R (A, B, C, D), with
FD = { A → B, B → C, D → B }

$A^+ = \{A, B, C\}$ $\quad (AD)^+ = (AD)^+ = \{A, D, B, C\}$

$B^+ = \{B, C\}$ $\quad (AB)^+ = \{A, B, C\}$

$C^+ = \{C\}$ $\quad (AC)^+ = \{A, B, C\}$

$D^+ = \{D, B, C\}$ $\quad (BC)^+ = \{B, C\}$

$(BD)^+ = \{B, D, C\}$ $\quad (CD)^+ = \{C, D, B\}$

:

Ex. If for a R of $X^+ = R.Attr$ (or R), then what can be said about X?

(A) SuperKey
(B) Primary Key
(C) Candidate Key
(D) Foreign Key.

$X^+ = R$ means $X \rightarrow R$, i.e
if I know X, we can uniquely identify R.
So, X must be SK.
It might / might not be CK & PK.

∴ if $X \subseteq R$ & $X^+ = R$, then X is SK of R.

(1) For a relation R, if $X \subseteq R$ & $X^+ = R$, then X is SK of R.

(2) For a relation R, if $X \subseteq R$ & $X^+ = R$ & $\forall Y \subset X$ $Y^+ \neq R$ then X is CK of R. (Key)

($\subset$: proper subset.)

Ex. R(A,B,C,D) FD = {A → B, B → C} Find all CKs of R.

$A^+ = ABC$

$B^+ = BC$

:

We can iterate, or we can observe.

Ob: A & D never appear on RHS of any FD, so for them to be determined they must be on LHS (trivial). And hence must also be in every CK.

$(AD)^+ = ADBC$.

$\therefore (AD)^+ = R$.

$\therefore$ AD is CK.

Now all superset of AD would be SK.

$\therefore$ Only AD is CK.

Ex. {AB → C, AE → D, D → B} R(A,B,C,D,E)

A,E must be in every Key since they aren't on any RHS.

$(AE)^+ = AE, D, B, C$. $\therefore$ AE is the only CK.

(∵ All its superset are SK)

- Inferring FDs

Ex. R(A, B, C) FD = {A → B, B → C}
which other FDs can be inferred?

A → A, B → B, C → C.
A → B, B → C ⇒ A → C.
A → C, B → C ⇒ AB → C.
AC → C, AC → A, ... etc.

Ex. R(A, B, C, D, E, F)
FD = {AB → C, BC → AD, D → E, CF → B}
which FD can be inferred from above set
(A) A → A ✓ (Trivial)
(B) A → BC ✗
Check A^+. $A^+ = A$. ∴ A ↛ BC.
(C) AB → BDE ✓
$(AB)^+ = ABCDE$ ∴ AB → BDE

- Closure of FD set: For a FD set 'F', its closure F^+ is the set of all FDs that can be inferred from it.

Ex. F = {A → B, B → C} R(A, B, C)
$A^+ = ABC$
So, A → A, A → B, A → C, A → AB, A → AC, A → BC, A → ABC
∴ $2^3 - 1 = 7$

$B^+ = BC$ $\quad C^+ = C$

#fds for $B = 2^2 - 1 = 3$ $\quad$ #fds $= 2^1 - 1 = 1$

$(AB)^+ = ABC$ $\quad (AC)^+ = ABC$ $\quad (BC)^+ = BC$

#fds $= 2^3 - 1 = 7$ $\quad 2^3 - 1 = 7$ $\quad 2^2 - 1 = 3$

$(ABC)^+ = ABC$

#fds $= 2^3 - 1 = 7$.

$\therefore$ #fds in $F^+ = 7 \times 4 + 3 \times 2 + 1 = 35$.

- FD Cover : A FD set F is said to cover another FD set G iff

$$G^+ \subseteq F^+$$

- FD set Equivalence: Two FD set F & G are said to be equivalent iff

F Covers G, i.e $\quad G^+ \subseteq F^+$

AND G Covers F, i.e. $\quad F^+ \subseteq G^+$

Ex. R(A, B, C, D) $\quad F = \{A \to B, B \to C\}$ $\quad G = \{A \to B, A \to C\}$

Which set covers which set

Checking for $G^+ \subseteq F^+$ (if F covers G):

$A^+ = ABC$ {Found A^+ using F, & all FDs of G are covered}

$\therefore \quad G^+ \subseteq F^+$

Checking for $F^+ \subseteq G^+$

$A^+ = ABC$.

$B^+ = B$.

$\therefore$ $B \to C$ not found So, $F^+ \not\subseteq G^+$.

$\therefore$ F covers G.

Ex. $F = \{A \to B, B \to ACD, CD \to B\}$ $R(A,B,C,D)$

$G = \{A \to CD, B \to CD, CD \to AB\}$

Checking $G^+ \subseteq F^+$

$A^+ = ABCD$

$B^+ = BACD$ ∴ F covers G.

$(CD)^+ = CDBA$

Checking $F^+ \subseteq G^+$

$A^+ = ACDB$

$B^+ = BCDA$ ∴ G covers F.

$(CD)^+ = CDAB$

∴ $F \equiv G$.

* • Prime Attribute: In a relation R, an attribute is called Prime if it is member of some Candidate Key.

• Non-Prime Attribute: In a relation R, an attr. which isn't part of any CK is called Non-Prime.

Ex. R(A,B,C) FD = {A → B, B → C}

A isn't on any RHS, so only A can determine A.
So, A is present in every Key.

$A^+ = ABC$ So A is SK & since its minimal its CK.

∴ Only 1 CK ⇒ A.

∴ Prime Attr: A

Non Prime Attr: B, C

* • Trivial: Something which always holds or never holds.

• Trivial FD: In a relation R, a FD $\alpha \rightarrow \beta$, $\alpha, \beta \subseteq R$ is called trivial iff $\beta \subseteq \alpha$, i.e. β is subset of α. (since it always hold)

• Non-Trivial FD: An FD is Non-Trivial, iff its non-trivial.

Ex. $AB \rightarrow BCD$

split RHS

$AB \rightarrow B$ Trivial

$AB \rightarrow CD$ Non-Trivial

∴ $AB \rightarrow BCD \equiv AB \rightarrow CD$.

Note: In further conversation about Normal forms, we'll only consider Non-trivial FDs split on RHS with each FD having single attr. on RHS.

* Fully-Dependent FD: A FD $\alpha \to \beta$ is fully-FD ~~dependent~~, iff $\forall_{A \in \alpha}\ \alpha - \{A\} \not\to \beta$.

OR

β is fully dependent on α iff any proper subset α' ($\alpha' \subset \alpha$) of α can't determine β.

* Partial FD: A FD $\alpha \to \beta$ is partial FD iff there is some proper subset of α can determine β, i.e.

$$\exists \gamma \subset \alpha \quad \gamma \to \beta$$

Ex. R(A, B, C, D, E) FD = {CD → E; D → A; A → E}

① Is CD → E partial?

$C^+ = C$ $D^+ = DAE$

So D can determine E

∴ CD → E is partial because we don't need entire CD to determine E.

② Is D → E partial?

No. We can't remove anything from LHS

③ $BD \rightarrow C$. Partial or full?

$BD \rightarrow C$ is not a valid FD on R, since $(BD)^+ = \{B, D, A, E\}$ & $C \notin (BD)$.

④ Find CKs.

BCD must be part of every CK, as they don't appear on any RHS.

$(BCD)^+ = \{B, C, D, A, E\}$

∴ Only $\{B, C, D\}$ is CK.

Ex. R(A, B, C, D, E, F)

FDs = { $ABC \rightarrow DE$, $DE \rightarrow ABC$, $ABC \rightarrow F$, $DE \rightarrow F$, $AB \rightarrow D$, $E \rightarrow C$ }

① $DE \rightarrow C$. Full / Partial

$D^+ = D$ $\quad E^+ = EC$

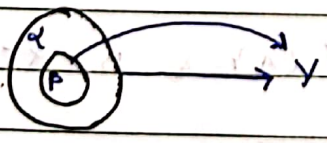
∴ Partial.

② $ABC \rightarrow D$

$A^+ = A$ $\quad B^+ = B$ $\quad C^+ = C$ $\quad (AB)^+ = ABD$

∴ Partial.

- Diagram of PD



* Transitive FD : A FD $X \to A$ $\{X \subseteq R, A \in A\}$ in relation R, iff

$A \notin X$

& $\exists_{Y \subseteq R} (X \to Y) \wedge (Y \to A) \wedge (Y \not\to X) \wedge (A \notin Y)$

$\{ X \to Y \to A \}$

- Some observations :

① $A \not\to X$.

Proof : Assume $A \to X$.

$Y \to A \to X \Rightarrow Y \to X$

But $Y \not\to X$

∴ Contradiction.

② ~~Every Partial FD is also transitive FD.~~

Ex. R(A, B, C) FD = $\{A \to C\}$

① Is $AB \to C$ a TD?

Yes. $AB \to A$ & $A \to C$ & $A \not\to AB$ & $C \notin \{A\}$ & $C \notin \{A, B\}$

(Also Partial FD)

Ex. R(A, B, C, D) FD = $\{A \to B, B \to C, C \to D\}$

① Is $A \to D$ partial?

No.

② Is $A \to D$ transitive?

$A \to C \wedge C \to D \wedge C \not\to A \wedge D \notin \{A\} \wedge D \notin \{C\}$

OR

$A \to C \to D$ & $D \notin \{A\}$ & $D \notin \{C\}$.

∴ Yes. $A \to D$ is transitive.

* Normal Forms:

A relation (relational schema) R, can be in the following Normal forms:

- 1NF } Defⁿ w.r.t. Domain
- 2 NF } Defⁿ w.r.t Non-Trival
- 3 NF } Functional Dependencies.
- BCNF } w/ RHS containing single attr.

- First Normal Form: A relation R is in 1NF iff the domains of all attributes of R are atomic.
  In Relational Model every relation is in 1NF, since we consider domain of every attr. as atomic.
  (So trivial for 1NF. Defined w.r.t Domain)

- Second Normal Form: A relation R is in 2NF iff every non-prime attr. is fully dependent on every Candidate Key.

OR R is in 2NF iff no non-prime attr is partialy dependent on some Candidate Key(or just Key)
So. FD of the form: Some CK ⟶ Non-prime attr, must not be partial.

Ex. R(A,B,C,D,E). FDs = {A → B, BC → E, ED → A}
CD not on any RHS. ∴ CKs = {ACD, BCD, ECD}
$(CD)^+ = \{C, D\}$
$(ACD)^+ = \{A, C, D, B, E\}$. ∴ CK
$(BCD)^+ = \{B, C, D, E, A\}$ ∴ CK
$(ECD)^+ = \{E, C, D, A, B\}$ ∴ CK

∴ Prime attr = {A, B, C, D, E}

Non Prime attr = {}.

∵ Non prime attr are not there

CK → NPA

type of partial dependency doesn't exist

∴ R is in 2NF.

Ex. R(A, B, C, D) FD = { A → C, B → A, A → D, AD → C }

B not on RHS.

B^+ = {B, A, C, D}

∴ CKs = {B}.

Prime Attrs = {B}

Non Prime Attrs = {A, C, D}

All FDs of form CK → NPA :

B → A, B → C, B → D.

None of them are partial FDs.

∴ R is in 2NF.

Ex. R(A, B, C, D) FD = {C → D, CD → A, AB → C, BD → A}

B not on RHS.

B^+ = {B} $(BC)^+$ = {B, C, D, A}

$(BA)^+$ = {B, A, C, D} $(BD)^+$ = {B, D, A, C}

∴ CKs = {AB, BC, BD}.

∴ No non-prime attr.

Hence no CK → NPA partial F.D.

∴ R is in 2NF.

Ex. R(A,B,C,D) FD = {BD→C, AB→D, AC→B, BD→A}

$(AB)^+ = \{A,B,D,C\}$ ✓

$(AC)^+ = \{A,C,B,D\}$ ✓

$(AD)^+ = \{A,D\}$

$(BC)^+ = \{B,C\}$

$(BD)^+ = \{B,D,A,C\}$ ✓

$(CD)^+ = \{C,D\}$

For any other comb. there is a proper subset which is Key.

∴ CKs = {AB, AC, BD}

∴ No NPA.

∴ In 2NF.

Ex. R(A,B,C,D) FD = {AB→CD, B→C}

AB not on RHS.

$(AB)^+ = \{A,B,C,D\}$.

∴ CKs = {AB}

Prime attr = {A,B}

NPA = {C,D}.

All FDs of CK→NPA form:

AB→C, AB→D.

AB→C is partial FD, ∵ B→C.

∴ R is not in 2NF.

- So, basically 2NF is violated when some proper subset of some CK can determine some non-prime attr.

Ex. R(A,B,C,D,E) FD = {AB→D, C→B, E→A}

$(CE)^+ = \{C,E,B,A,D\}$

∴ CKs = {CE} PA = {C,E} NPA = {A,B,D}.

FDs of form CK→NPA.

CE→A, CE→B, CE→D.

∵ CE→A is partial FD, (E→A) ∴ R isn't in 2NF.

Ex. R(A,B,C,D,E) FDs = {AC → BDE, A → B, D → E}

AC not on RHS.

$A^+ = \{A, B\}$

$(AB)^+ = \{A, B\}$ ∴ CKs = {AC}

$(AC)^+ = \{A, C, B, D, E\}$ PAs = {A, C}

~~$(AD)^+$ {A, D, B, E}~~ NPAs = {B, D, E}

All FDs of form CK → NPA:

AC → B, AC → D, AC → E.

∵ AC → B is Partial FD (∵ A → B)

∴ R not in 2NF.

Ex. Can 2NF have partial dependency?

Yes, it is possible.

Ex. {AB → CD, CD → AB, B → D.} R(A,B,C,D)

This is in 2NF.

AB → D is Partial FD & allowed.

Only CK → NPA type of Partial FD disallowed.

Ex. R is in 2NF iff some NPA is fully dependent on all CKs. T/F.

False. All & NPA, not some.

Ex. R is in 2NF iff all NPA are fully dependent on all Superkeys. T/F?

False. Only on all CKs.

Ex. Can a non-prime attr determine some prime attr?

No. Proof: Assume $A \rightarrow B$, A is non-prime & B is prime.

Since B is prime, it is part of some CK, XB. $X \subseteq R$.

Since $A \rightarrow B$,

$$AX \rightarrow XB$$

So, AX is CK. A is prime.

∴ Contradiction.

Ex. Can a non-prime attr. determine another non-prime attr?

Yes. Ex. $AB \rightarrow CD$

$C \rightarrow D$

Ex. Can a prime attr. determine another prime attr?

Yes

- Third Normal Form: A relation R is in 3NF iff every non-prime attr. is non-transitively dependent on every Candidate Key.

So 3NF is violated by R, iff some non-prime attr. is transitively dependent on some CK.

3NF hence is violated when:

(CK $\rightarrow$ Non Superkey $\rightarrow$ Non Prime Attr) $\wedge$ (NPA $\not\subseteq$ CK) $\wedge$ (NPA $\not\subseteq$ Non SK)

(obvious.)

So, such FDs in R cause violation:

Non-Superkey $\rightarrow$ Non-prime attr

∴ (Not a SK ⟷ NPA)

(Some CK —Transitively→ Some NPA)

Proof:

Suppose α is Non SK & is NPA.

α can't determine any CK, since it's Non SK.

If $\alpha \rightarrow A$, then

all CK B, $B \rightarrow \alpha$.

$B \rightarrow \alpha \rightarrow A$

(since α is Non SK)

$\therefore B \xrightarrow{\text{Transitively}} A$.

Ex. Is Some NPA (A) ⟶ Some other NPA (B) a problem to BNF?

Yes. In $A \rightarrow B$, A is non SK.

$\therefore$ Non SK → NPA type of FD.

Also for some CK,

$CK \rightarrow A \rightarrow B$ & $B \not\rightarrow A$ & $B \notin CK$

Ex. Is Non Prime (A) ⟶ Prime (B) a problem to BNF?

Never possible.

Not a SK

Ex. Is Non-Prime + Prime (α) $\longrightarrow$ Non Prime (A) a problem for 3NF?

Yes. Non SK $\longrightarrow$ Non Prime

Also,

$(CK \longrightarrow \alpha \longrightarrow A) \wedge (A \notin CK)$ $(A \notin \alpha)$

So, 3NF violations include FDs of type:

① NPA $\longrightarrow$ NPA.

② Partial dependency of form CK $\rightarrow$ NPA, OR (Proper subset of CK) $\longrightarrow$ NPA. } violation of 2NF

③ Prime + Non Prime $\longrightarrow$ NPA.
Non SK

Ex. R(A, B, C, D) FDs = {A $\rightarrow$ C, B $\rightarrow$ A, A $\rightarrow$ D, AD $\rightarrow$ C}.
Check if in 3NF.

B not on RHS

B^+ = B, A, C, D.

∴ CKs = {B}

∴ PA = {B} NPA = {A, C, D}

A $\rightarrow$ C violates 3NF. (Non Prime $\rightarrow$ Non Prime)

∴ R not in 3NF.

Ex. R(A, B, C, D) FDs = {C $\rightarrow$ D, CD $\rightarrow$ A, AB $\rightarrow$ C, BD $\rightarrow$ A}

B not on RHS.

B^+ = B.

$(AB)^+$ = A, B, C, D ∴ CKs = {AB, CB, DB}

$(CB)^+$ = C, B, D, A Prime = {A, B, C, D}

$(DB)^+$ = D, B, A, C. ∴ No NPA

∴ In 3NF.

Ex. R(A,B,C,D) FD = { BD → C, AB → D, AC → B, BD → A }

$(AB)^+ = \{A, B, D, C\}$

$(AC)^+ = \{A, C, B, D\}$

$(AD)^+ = \{A, D\}$

$(BC)^+ = \{B, C\}$

$(BD)^+ = \{B, D, C, A\}$

$(CD)^+ = \{C, D\}$

∴ CKs = {AB, AC, BD}.

Prime = {A, B, C, D}.

∴ No NPA

∴ 3NF.

Ex. In 3NF can we have transitive dependency?

Yes. Prime Attr → Prime Attr is fine.

Ex. R(A,B,C,D,E)

AB ⟷ CD

A ⟶ C.

Here, AB → C is transitive dependency.

Ex. In 3NF, a PA can be transitively determined by CK?

Yes. There exists a CK → PA FD for all Prime Attr.

Ex. In 3NF, a NPA can be transitively determined by another NPA?

No.

- A partial dependency implies transitive dependency,

$\alpha \longrightarrow \beta \longrightarrow A$ where $\beta \subset \alpha$.

& β is Non SK (∵ Proper subset of CK)

So, any PD of form : $CK \xrightarrow{Par.} NPA$ would also be a Transitive Dependency.

Hence R is not in 2NF $\longleftrightarrow$ R is not in 3NF.

$\Rightarrow$ R is in 3NF $\longrightarrow$ R is in 2NF

∴ If a relation R is in 3NF, it is also in 2NF.

Ex. R(A,B,C,D) FD = {B→C, AC→D, ABD→C, BCD→A}.

Is R in 3NF? 2NF?

B not on RHS.

$B^+ = B, C,$

$AB^+ = A,B,C,D$ ∴ CKs = {AB, DB}

$CB^+ = C,B$ PAttrs = {A, B, D}

$DB^+ = B,D,C,A.$ NPAttrs = {C}

Not in 3NF. $AB \rightarrow B \rightarrow C$ & $C \notin AB$ & $C \notin B$.

∴ Trans. dependency $AB \rightarrow C$.

Not in 2NF. $AB \rightarrow C$, but $B \rightarrow C$. ∴ Partial dependency.

∴ 1NF

Ex. R(A,B,C,D) FD = {AB→C, ABD→C, ABC→D, AC→D}

AB not on RHS,

$(AB)^+ = \{A,B,C,D\}$ ∴ CKs = {AB}

PAttr = {A, B}

NPAttr = {C, D}.

Not 3NF. $AB \rightarrow AC \rightarrow D$

In 2NF.

Ex. R(A, B, C, D) FDs = {ABD→C, A→B, AB→C, B→A}

D not on RHS.

$D^+ = D$.

$AD^+ = ADBC$. ∴ CK's = {AD, BD}

$BD^+ = BDAC$. PAttr = {A, B, D}

$CD^+ = CD$ NPAttr = {C}

Not in 3NF. AD → AB → C.

Not in 2NF. BD → C, but B → C.

∴ In 1NF

Ex. R(A, B, C, D) FDs = {ACD→B, AC→D, D→C, AC→B}

A not on RHS

$A^+ = A$

$AB^+ = AB$ ∴ CKs = {AC, AD}

$AC^+ = ACDB$ PAttrs = {A, C, D}

$AD^+ = ADCB$ NPAttrs = {B}.

All possibe nonSK = {C, D, A, CD}

None determine any NPAttr.

So no NonSK → NPAttr dependency.

∴ In 3NF.

- Boyce-Codd Normal Form: In every non-trivial FD relation R, iff $\alpha \to B$ ($\alpha, \beta \subseteq R$), α is SuperKey then R is in BCNF.

3NF is saying that NonSK → NonPrime should'nt happen. i.e either LHS is SK or RHS is prime.

BCNF is saying that LHS should be SK.

	3NF	BCNF
SK → NPAttr	✓	✓
SK → PAttr	✓	✓
NonSK → PAttr	✓	X
NonSK → NPAttr	X	X.

∴ BCNF is inherently 3NF.

So, if R is in BCNF then R is in 3NF as well.

Ex: When will a relation be in 3NF but not BCNF?

If the relation has no FD of form NonSK → NPAttrs, but there is some FD of the form NonSK → PAttr then relation is in 3NF, not BCNF.

Ex. R(A, B, C, D, E, F) FD = {A → BC, C → F, D → E}

AD not on RHS

$(AD)^+ = \{A, D, B, C, F, E\}$.

∴ CK = {AD,} PAttrs = {A, D} NPAttrs = {B, C, E, F}

Not in BCNF. {A → BC, A isn't SK}

Not in 3NF. AD → A → B ∧ B ∉ AD ∧ B ∉ A.

Not in 2NF. ← AD → B, but A → B.

∴ R is in 1NF.

Ex. R(A,B,C,D,E) FDs = {A→C, B→D, AB→E}
R is in which NF?

A,B not on RHS.

AB^+ = A, B, C, D, E

∴ CKs = {AB} PAttr = {A, B} NPAttr = {C, D, E}

Not in BCNF. {A→C, A not SK}

Not in 3NF. {AB → A → C & C ∉ AB & C ∉ A}

Not in 2NF. {AB → C, but A → C}

∴ R is in 1NF.

Ex. Which of the following is false?

(A) A relation w/ 2 attr is in BCNF.

(B) A rel. w/ every Key having single attr is in 2NF.

(C) A prime attr. can be transitively dependent on a Key in 3NF.

(D) A prime attr can be transitively dependent on a key in BCNF.

A. BCNF says only SK → α type of dependencies allowed.
For rel. w/ 2 attr (say A, B) possible non-trivial FDs are {A→B, B→A}. For any subset of this:

(1) {A→B}. CK={A}. PA={A}. NPA={B} A→B A is SK.

(2) {B→A} ... B is SK.

(3) {A→B, B→A} ... A, B are SK.

∴ Always in BCNF.

B. If every Key is w/ single attr CK → NPAttr can never be partial. ∴ Always in 2NF.

C. 3NF disallows FD of the form NonSK → NPA. So, NonSK → PA & SK → PA are allowed. So PAttr. transitively dependent on some CK is allowed.

D. Suppose it is allowed. CK $\xrightarrow{\text{Trans.}}$ PAttr. So there must be some α such that,

$$CK \longrightarrow \alpha \longrightarrow A \;\&\; A \notin CK \;\&\; A \notin \alpha.$$

But this is BCNF. So, α must be SKey for $\alpha \to A$ to be valid FD, & if α is SK, then $\alpha \to CK$.
∴ Contradiction.

∴ D is false.

* Natural Join

Ex. Emp

SSN	Name	DoB	Paygrade
101	ABC	213	2
253	DEF	312	3
192	GHI	916	2
211	JKL	619	4

Pay-Grade

Paygrade	Salary
2	70
3	60
4	80

- So say we want to find DoB of (SSN=192). This only requires one table, Emp. DoB is 916.
- What is the salary of SSN = 253. This requires both tables. First we check the Paygrade in Emp for SSN=253. Then from Paygrade table, we find salary for the found Paygrade. This is the natural way of joining two tables.

- Natural join of two tables:
  (1) Find the common attributes (attrs. with same name).
  (2) Equate the values. (Match same values for the common attrs).
  (3) In combined relation write common attributes once.

Ex. Previous example, joined.

SSN	Name	DoB	Payscale	Salary
101	ABC	213	2	70
253	DEF	312	3	80
192	GHI	916	2	70
211	JKL	619	4	80

- Symbol for Natural Join: ⋈
- Only the tuples for which common attributes match are allowed in the joined table.

A	B	C
3	3	2
6	4	3
2	4	1
3	5	1
7	1	2

⋈

B	C	D
5	1	6
1	5	8
4	3	9

⇒

A	B	C	D
6	4	3	9
3	5	1	6

- If b/w two tables if there are no common attributes, then combine every row of first table with every row of second.

* Redundancy & Anomalies.

- We can notice, that imposing functional dependency constraints results in data redundancy.

Ex. ∴ District → State

SSN	Dist.	State	
1	Jaipur	Raj	} Redundancy in state attr.
2	Jaipur	Raj	
3	Kota	Raj	} no redundancy since district is different.

- Although not explicitly stated yet but FD constraints aren't limited to a single relation. Initially there is no tables at requirement stage. There are attrs. that should be there in the DB & the FDs that they should follow. Based on it & other requirements relations are designed. So FDs constraints must hold in the DB not within a single table.

- **Update Anomaly:** If there is redundancy, say due to $X \to Y$, and you want to update Y corresponding to $X = x_1$, then you'll have to update at multiple places $\langle x_1, y_1 \rangle$ to $\langle x_1, y_2 \rangle$. This could lead to consistency, if the data isn't updated in all corresponding places.

- Insertion Anomaly: Suppose the following Schema:

  S (Eid, Name, Salary, Dept)

Now you want to add a new department, This can't be done without adding an employee.

- Deletion Anomaly: Say the last employee of Dept = Music resigns & you delete the data, then with the employee, music department is also gone.

- These anomalies are happening because we are trying to put all attrs. in a single table.

The anomalies can be solved by the process of decomposition.

- Some qualities of good database design:
  - Making sure that semantics of the attributes is clear in schema.
  - Reducing the redundant info. in table.
  - Reducing the null values in tuples.
  - Disallowing the possibility of generating spurious tuples.

* Decomposition

- For a relational schema R, $R_1, R_2 \ldots, R_n$ is called as decomposition of R, iff

$$\bigcup_{i=1}^{n} R_i = R$$

↑ union of schemas

i.e. attributes must be preserved, no less no more.

Ex. Which of the following is/are decomposition of R(A,B,C,D):

(a) {AB, CD} ✓

(b) {AB, BCD} ✓

(c) {A, B, C, CD} ✓

(d) {A, BC}

- Binary Decomposition: Decomposition into exactly two parts.

- Lossless Decomposition: A decomposition which will never give wrong information

Lossy Decomposition: A decomposition which may/mynot give wrong information. (i.e. for some DB instance it will give wrong info)

Ex. R:

a	b	c
1	2	3
2	2	3
3	2	2

⇒

R_1:

a	b
1	2
2	2
3	2

R_2:

b	c
2	3
2	2

Suppose you want a for c = 2.

Original: 3

Decomp: 1, 2, 3

↳ wrong info.

Original table has three tuples.
If we natural join $R_1 \bowtie R_2$, then we'll get $3 \times 2 = 6$ tuples.

The extra tuples are called Spurious tuples.

So, basically in Lossy Decomposition we don't loose tuples, rather we get spurious tuples. Hence this is loss of information, not tuples.

Ex. Same as before:

R_1: a	b
1	2
2	2
3	2

R_2: a	c
1	3
2	3
3	2

So if we do $R_1 \bowtie R_2$ we get exactly R.

∴ For a lossy decomposition $R_1, R_2 \ldots R_n$ of relation R is such that:

$$R \subseteq \bigcup_{i=1}^{n} R_i$$

For lossless decomposition $R_1, R_2, \ldots R_n$ of relation R

$$\bigcup_{i=1}^{n} R_i = R$$

- In a binary decomposition $\{R_1, R_2\}$ we can check Lossy vs Lossless:

A binary decomposition is lossless if the common attr $R_1 \cap R_2$ is superkey of at least one of the schema, i.e.

$$[(R_1 \cap R_2) \rightarrow R_1] \vee [(R_1 \cap R_2) \rightarrow R_2]$$

→ lossless binary decomp.

else the binary decomposition is ~~lossless~~ lossy.

Lossless decomposition is a.k.a Lossless Join Decomposition.
a.k.a Non-Additive Join Decomposition.

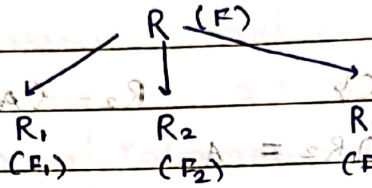
Ex. $R(A,B,C,D,E)$ $FDs = \{A \to BC, CD \to E, B \to D, E \to A\}$

1. Is $R_1 = \{A,B,C\}$ $R_2 = \{A,D,E\}$ lossless?
$R_1 \cap R_2 = A$
Checking $A^+ = ABCDE$
∴ A can determine B, C.
∴ A is SK in R_1 (also in R_2)
This is enough to say R_1, R_2 is lossless decomp

2. $R_1 = \{A,B,C\}$ $R_2 = \{C,D,E\}$
$R_1 \cap R_2 = C$.
$C^+ = C$.
Not SK in either.
∴ Lossy.

Ex. $R(A,B,C,D,E)$ $A \to BC, CD \to E, B \to D, E \to A$.
$R_1(A,B,C)$ $R_2(A,D,E)$.
$R_1 \cap R_2 = D$
$D^+ = D$.
∴ SK in neither R_1 nor R_2.
So Lossy.

- Dependency Preserving Decomposition:
Initially the database has some attributes & FDs on them. Say set of FD on database is F. Then the database/ universal relation is broken down into multiple relations, each of which have set of FDs which only hold on them:



A decomposition of relation R with FD set F into multiple relations, $R_1(F_1)$; $R_2(F_2)$ ... $R_n(F_n)$ is called Dependency Preserving iff

$$(F_1 \cup F_2 \cup \dots F_n) \equiv F$$

i.e

$\bigcup_{i=1}^{n} F_i$ covers F & F covers $\bigcup_{i=1}^{n} F_i$

Ex. R (A, B, C, D, E) FDs = { A → BC, CD → E, B → D, E → A }

R1 (A, B, C) R2 (A, D, E)

$A^+ = ABCDE$

$B^+ = BD$

$C^+ = C$

$D^+ = D$

$E^+ = EABCD$

In R1: $A^+ = ABC$ $B^+ = B$ $C^+ = C$

In R2: $A^+ = ADE$ $D^+ = D$ $E^+ = EAD$

$F_1 = \{ A \to BC, BC \to A \}$

$F_2 = \{ A \to DE, E \to AD \}$

(we are only writing Non-Trival FDs)

Checking Preservence :

$(F_1 \cup F_2)$ doesn't cover F

$(CD)^+ = CD$

Hence $CD \rightarrow E$ isn't covered

∴ Not Dependency preserving.

- Even if a decomposition is lossless but not dependency preserving, it doesn't really loose dependency.
Say R is decomposed into R_1 & R_2 & it is not dependency preserving.
Still when $R_1 \bowtie R_2$, all the dependencies are restored since the decomp. is lossless.
It's called non-dependency preserving since without joining the dependency can't be enforced.
That means if we do a join only the valid rows (which follow all FDs) would be there, but without the join the individual tables can have data that doesn't follow all the dependencies.

Ex. R(Sid, State, District) FDs = {Dist → State, Sid → Dist}

R_1 (Sid, State) R_2 (Sid, Dist)

F_1 = {Sid, State} F_2 = {Sid → Dist}

- This is lossless but not dependency preserving.
In R the tuple <213, Haryana, Jaipur> would not be allowed.
In R_1 <213, Haryana> & in R_2 <213, Jaipur> is allowed
In $R_1 \bowtie R_2$, the tuple won't appear. So although on lossless join dependencies aren't lost, individually the lost FDs can't be enforced on R_1 & R_2.

- In any decomposition lossless property is necessary, dependency preserving is desirable.

Ex. $R(A,B,C,D,E)$ $FDs = \{A \to BC, CD \to E, B \to D, E \to A\}$

① $R_1(A,B,E)$ $R_2(B,C,E)$ $R_3(B,C,D)$

$R_1 \cap R_2 = BE$

$BE^+ = B,E,D,A,C$

∴ R_1, R_2 is lossless decomp.

$F_1 = \{A \to BE, E \to A\}$ $F_2 = \{E \to BC, BC \to E\}$

$F_3 = \{B \to D, CD \to B\}$

$(F_1 \cup F_2 \cup F_3)$ covers F.

F obviously covers $F_1 \cup F_2 \cup F_3$.

∴ This is dependency preserving.

Ex. $R(A,B,C,D)$ $FDs = \{A \to B, B \to C, C \to D, D \to A\}$

$R_1(A,B)$ $R_2(B,C)$ $R_3(C,D)$

$F_1 = \{A \to B, B \to A\}$ $F_2 = \{B \to C, C \to B\}$ $F_3 = \{C \to D, D \to C\}$

$(F_1 \cup F_2 \cup F_3)^+$ covers F

∴ Dependency preserving.

- How much should the database be decomposed depends on the Normal form. As we go from 1NF → 2NF → 3NF → BCNF, redundancy due to FDs keep decreasing with 0% in BCNF (0% due to FDs, there can be other type of redundancy). {such as MVD}

- Normalization: Decomposing a relation R such that each decomposed relation is in a higher normal form than R, and the dependency must be lossless.

- Non-Binary Decomposition: NB Decomposition of a relational schema R is $R_1, R_2, \ldots, R_n$, $n > 3$, such that

$$R_1 \cup R_2 \cup \ldots R_n = R$$

$$\text{i.e. } \bigcup_{i=1}^{n} R_i = R$$

We can check if a Non-Binary decomp. is Dependency Preserving or not by using the defⁿ of Dependency Preserving decomp.

To check if a binary decomp of R: R_1, R_2, is lossless we check

$$R_1 \cap R_2 \rightarrow R_1$$

OR

$$R_1 \cap R_2 \rightarrow R_2$$

This doesn't work for Non Binary decomp.
We use "Chase Algorithm" to check lossless join property of NB decomp.

- Chase Algorithm: Used to check if a decomposition (binary or non-binary) is lossless or not.

Inputs: Relational Schema $R(A_1, A_2, \ldots A_n)$, a set of FDs F, a decomposition $\rho = (R_1, R_2, \ldots R_K)$

- Construct a matrix of size $K \times n$ where each row represents one subschema & each column represents one attribute
- Mark $M[i][j]$ if A_j is in subschema R_i, with a_j, else with b_{ij}
- For each FD $X \rightarrow Y$ in FD set $(X, Y \subseteq R)$ if there are two or more rows such that for all attrs in X, the rows have same value, then in those rows we equate the values for all attrs. in Y.
  If in one of the equal val. for X rows, if A_j $(A_j \in Y)$ has a_j entry then make all the other rows entry as a_j. Else if one of them is b_{ij} make all of them b_{ij}.
- Repeat on the FD set untill no changes to M.
- If now any row looks like $a_1, a_2 \ldots a_n$ then decomp is lossless, else lossy.

Ex. $R(A, B, C, D)$ $FDs = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$.
$R_1(A, B, C)$ $R_2(C, D)$.

	A	B	C	D
R_1	a_1	a_2	a_3	a_4
R_2	b_{21}	b_{22}	a_3	a_4

This is final table.
We can see R_1 is $a_1 a_2 a_3 a_4$.
∴ Lossless decomp.

Ex. R(A, B, C, D) FDs = {A → B, B → C, C → D, D → A}

R_1(A, B) R_2(B, C) R_3(C, D)

	A	B	C	D
R_1	a_1	a_2	~~b_{13}~~ a_3	~~b_{14}~~ a_4
R_2	~~b_{21}~~ a_1	a_2	a_3	~~b_{24}~~ a_4
R_3	~~b_{31}~~ a_1	~~b_{32}~~ a_2	a_3	a_4.

∴ Lossless (rows 1, 2, 3 have $a_1 a_2 a_3 a_4$).

Ex. R(A, B, C, D, E) FDs = {A → C, B → C, C → D, DE → C, CE → A}

R_1(A, D) R_2(A, B) R_3(B, E) R_4(C, D, E) R_5(A, E)

	A	B	C	D	E
R_1	a_1		~~b_{13}~~ a_3	a_4.	
R_2	a_1	a_2	~~b_{23}~~ a_3	a_4	
R_3	a_1	a_2	~~b_{33}~~ a_3	a_4	a_5
R_4	a_1		a_3	a_4	a_5
R_5	a_1		~~b_{53}~~ a_3	a_4	a_5

(assume all initially empty cells have b_{ij})

We can stop at this point, since R_3 has $a_1 a_2 a_3 a_4 a_5$ (not because no further changes).

∴ Lossless decomposition.

Ex. R(A, B, C, D) FDs = {A → B, B → C, C → D, D → B} R_1(A, B) R_2(B, C) R_3(B, D)

	A	B	C	D
R_1	a_1	a_2	a_3	a_4
R_2		a_2	a_3	a_4
R_3		a_2	a_3	a_4

∴ Lossless

Eg. $R(A,B,C,D,E)$ $R_1(A,B,C)$ $R_2(B,C,D)$
$R_3(A,C,E)$.

Check if the above decomp. is (1) lossless & (2) dependency preserving with following FD set:

(A) $B \to E$ & $CE \to A$

Checking DP:

$F_1 = \{BC \to A\}$ $F_2 = \{\}$ $F_3 = \{CE \to A\}$

So, not preserved.

Checking lossless.

	A	B	C	D	E
R_1	a_1	a_2	a_3		b_{15}
R_2	a_1	a_2	a_3	a_4	b_{15}
R_3	a_1		a_3		a_5.

So, lossy.

(B) $AC \to E$ & $BC \to D$

$F_1 = \{\}$ $F_2 = \{BC \to D\}$ $F_3 = \{AC \to E\}$

∴ Dependency preserved.

	A	B	C	D	E
R_1	a_1	a_2	a_3	a_4	a_5
R_2		a_2	a_3	a_4	
R_3	a_1		a_3		a_5

∴ Lossless

(C) $A \to D$, $D \to E$, $B \to D$.

$F_1 = \{\}$ $F_2 = \{B \to D\}$ $F_3 = \{A \to E\}$.

Not DP.

	A	B	C	D	E
R_1	a_1	a_2	a_3	a_4	a_5
R_2		a_2	a_3	a_4	a_5
R_3	a_1		a_3	a_4	a_5

∴ lossless.

(D) $A \to D$, $CD \to E$ & $E \to D$.

$F_1 = \{\}$ $F_2 = \{\}$ $F_3 = \{AC \to E\}$

Not DP.

	A	B	C	D	E
R_1	a_1	a_2	a_3	b_{14}	a_5
R_2		a_2	a_3	a_4	
R_3	a_1		a_3	b_{14}	a_5

∴ Lossy.

Ex. $R(L, M, N, O, P)$ $FDs = \{M \to O, NO \to P, P \to L, L \to MN\}$

$R_1(L, M, N, P)$ $R_2(M, O)$

(1) Is the above decomp. lossless?

(2) Is the above decomp. dependency preserving?

(3) What is the highest normal form satisfied by above decomp?

(1) $R_1 \cap R_2 = M$ $M^+ = MO$ $M \to R_2$

∴ Lossless.

(2) $F_1 = \{L \to MNOP, P \to L\}$ $F_2 = \{M \to O\}$

$NO \to P$ isn't covered by $F_1 \cup F_2$.

∴ Not dependency preserving.

(3) For R_1:

$CKs = \{L, P\}$

∴ R_1 is in BCNF (since all LHS are skey)

For R_2:

$CKs = \{M\}$

∴ R_2 in BCNF

∵ R_2 has 2 attr.

∴ Highest NF satisfied by decomp is BCNF.

* Minimal Cover of FDs

(a.k.a Cannonical Cover, aka Irreducibe FD Set)

most appropriate name.

- Let's say we define stronger FD (non-standard term) as:

  Among 2 FDs, F_1 & F_2 F_1 is stronger than F_2 iff $F_1 \longrightarrow F_2$ (or $F_1 \models F_2$)

  Ex. $A \longrightarrow C$ (f_1) $\qquad$ $A \longrightarrow BC$ (f_2)

  Here, $f_2 \longrightarrow f_1$

  $\therefore$ f_2 is stronger.

  Ex. $AB \longrightarrow C$ (f_1) $\qquad$ $A \longrightarrow C$ (f_2)

  AB^+ (using f_2) $= ABC$ $\qquad$ A^+ (using f_1) $= A$

  $\therefore$ $f_2 \longrightarrow f_1$

  So, f_2 is stronger.

  Ex. $A \longrightarrow B$ (f_1) $\qquad$ $A \longrightarrow C$ (f_2)

  A^+ (using f_2) $= AC$ $\qquad$ A^+ (using f_1) $= AB$

  So neither implies the other

  $\therefore$ Can't compare.

- For an FD set F, its closure F^+ is defined as

  $$F^+ = \{ g \mid g \text{ is a FD} \wedge F \models g \}$$

So, F^+ (closure) includes all the FDs g that can be inferred from F.

- We can go the other way around for a FD set F, and get a set of only essential FDs, i.e, the FD set which is minimal / irreducible / canonical & covers F.

- Extraneous Attributes on LHS of a FD (Redundant)

Ex. $F = \{A \to C, AB \to C\}$

here, we don't have need for B. A can imply C.

So, $\{A \to C\} \to F$

- A FD $\alpha A \to \beta$, ($\alpha \subseteq R$, $A \in R$) has A as extraneous (redundant) variable iff $\alpha \to \beta$, or $B \subseteq \alpha^+$.

Ex. $F = \{AB \to C, A \to B\}$

In $AB \to C$,

- Removing B, $A^+ = AB$
- Removing A, $B^+ = B$

So, both A, B are essential.

~~$AB \to C$~~

~~Removing B, we get $\{A \to C, A \to B\}$~~

Ex. $F = \{AB \to C, A \to B\}$

For $AB \to C$, $A^+ = ABC$ $B^+ = B$

$\therefore$ A can determine C.

So, B is redundant.

$\therefore \{A \to C, A \to B\}$.

Ex. F = { AC → G, ACD → B, BC → D, CG → B, CG → D, CE → A, CE → G, D → E, D → G }

For AC → G: $A^+ = A$ $C^+ = C$

∴ No extraneous.

For ACD → B: $AC^+ = ACGB$

∴ D is extraneous.

$A^+ = A$ $C^+ = C$

∴ Both not extraneous.

∴ AC → B.

For BC → D: $B^+ = B$ $C^+ = C$.

∴ No extra.

For CG → B: $C^+ = C$ $G^+ = G$.

For CG → D: $C^+ = C$ $G^+ = G$.

For CE → A: $C^+ = C$ $E^+ = E$.

For CE → G: $C^+ = C$ $E^+ = E$

∴ FD = { AC → G, AC → B, BC → D, CG → B, CG → D, CE → A, CE → G, D → E, D → G }

Removing Redundant FD

- First make sure that on RHS we only have single attributes.
- In a FD set F, a FD X → A ($X \subseteq R, A \in R$) is redundant iff F − {X → A} still infers X → A (or still infers $A \in X^+$).
- Basically, if using all FDs except X → A if we can show that X^+ contains A, then X → A is redundant.

Ex. $F = \{AC \to G, CD \to B, BC \to D, CG \to B, CG \to D, CE \to A, CE \to G, D \to E, D \to G\}$.

For $AC \to G$: $AC^+ = AC$

For $CD \to B$: $CD^+ = CDEGBA$ ∴ $CD \to B$ is redundant.

Now FD is $F - \{CD - B\}$. Further checking will be done with it.

For $BC \to D$: $BC^+ = BC$.

For $CG \to B$: $CG^+ = CGDEA$

For $CG \to D$: $CG^+ = CGBDEA$ ∴ $CG \to D$ is red.

Now FD is $F - \{CD \to B, CG \to D\}$.

For $CE \to A$: $CE^+ = CEGBD$

For $CE \to G$: $CE^+ = CEAGBD$ ∴ $CE \to G$ is red.

FD is $F - \{CD \to B, CG \to D, CE \to G\}$.

For $D \to E$: $D^+ = DG$

For $D \to G$: $D^+ = DE$

∴ $FD = \{AC \to G, BC \to D, CG \to B, CE \to A, D \to E, D \to G\}$.

Note: Minimal Covers for a FD is not unique. It depends on order of removing extraneous var. on LHS & order of removing redundant FDs.

- Steps to find minimal cover of FD:
  - Break FDs if needed, such that every FD on RHS has only one attr.
  - For every FD, check if one or more variable is redundant & remove the redundant variables on LHS.
  - For all remaining FDs check if they are redundant, and remove them.

- Minimal Cover doesn't mean minimum no. of FDs, rather it means that no FD can further be removed from the cover.

- Order of removing redundant LHS vars & removing redundant FDs doesn't matter.

Ex. R(A, B, C, D, E, F, G)

FDs = { AC → G, DE → E, D → G, BC → D, CG → B, CG → D, ACD → B, CE → A, CE → G }

S1: Removing extraneous on LHS

- AC → G : $A^+ = A$ $C^+ = C$
- DE → E
- D → G
- BC → D : $B^+ = B$ $C^+ = C$
- CG → B : $C^+ = C$ $G^+ = G$
- CG → D : $C^+ = C$ $G^+ = G$
- ACD → B : $AC^+ = ACGDBE$ ∴ D is red.
- CE → A : $C^+ = C$ $E^+ = E$
- CE → G : —11—

∴ F = {AC → G, D → E, D → G, BC → D, CG → B, CG → D, AC → B, CE → A, CE → G}

S2: Removing red. FDs.

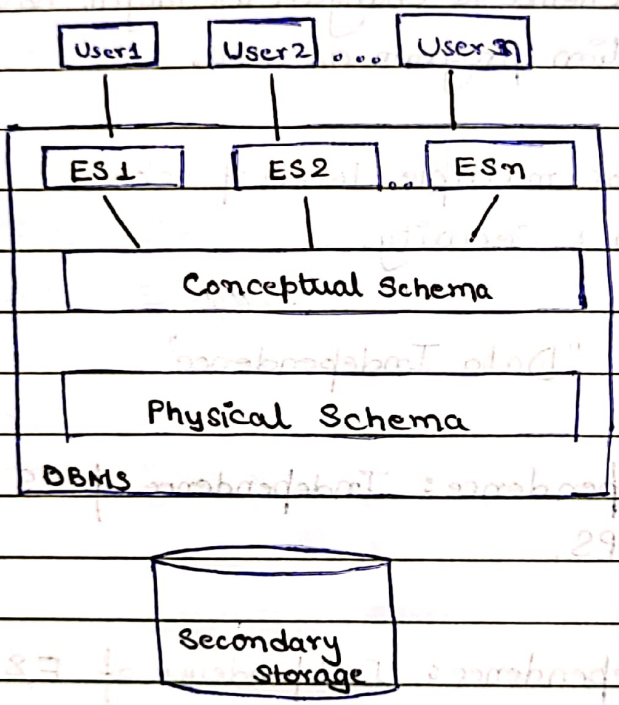
Red. FDs ⇒ { CG → B, CE → G }.

∴ Minimal Cover: {AC → G, D → E, D → G, BC → D, CG → D, AC → B, CE → A}

* Three levels of Schema.

- In a database, not all the data be visible to every user. Different users require different level of access & visibility to data.

Ex. In a bank's database, a user must not be able to access other customer's transactions.



ES: External Schema.

Physical Schema: Tells how the data is actually stored on the disk, sorted or unsorted, indices, datastruct used, etc. (Internal schema)

Conceptual Schema: Tells what is the meaning (structure, interpretation, constraints, etc) of the data. (Logical Schema)

External Schema: Tells about the use of data by various DB users.

Data is actually stored on secondary storage.

- External schema isn't stored on disk. It is derived as per requirement from conceptual schema.

Ex. Application program can require data consisting of attr. from multiple relations, or based on some filters, or some specific ordering.

- If conceptual schema is changed, we might have to change application program as well.

- One of the reason for multiple levels of schema is Data Abstraction & Security.

- Another reason is "Data Independence"

- Physical Data Independence: Independence of CS from changes in PS.

- Conceptual Data Independence: Independence of ES from changes in CS.

- Although possible, it is rare that change to PS result in change at CS level. This is ensured by DBMS. So DBMS provides us the facility to change physical changes without requiring changes at CS level & Application level.

- DBMS also takes up the responsibility to ensure that changes made at CS level should merely result in changes at ES level. This is more difficult to achieve. [Conceptual / Logical Data Indep.]

Normalization

(3NF & BCNF Decomposition)

* Redundancies

- Trivial FDs never create redundancies.
- Non-Trivial FDs may/maynot create redundancies.

Ex. Student (sid, name, district, state) Dist → State.
Say there are 500 students from Jaipur, then all of them will have state Rajasthan. So the repetition of state value is redundancy.
Suppose we decompose the relation as:
S (sid, name, dist) R (dist, state).
Now <dist, state> tuple will be stored once & state won't be redundantly specified.
So, decomposition is a sol^n to redundancy.
Why does (dist → state) causes redundancy?
① Repetition of dist causes repetition of state.
② Since dist is not SK in rel^n. so dist can repeat.

Ex. R(A,B,C) FDs = {AB → C, C → B}
Which FD may create redundancy?
CKs = {AB, AC}
So, AB is SK, it won't repeat.
C is not SK, it can repeat, & repetition of C causes repetition of B.
∴ C → B may create redundancy

- So, non-trivial FD with LHS as superkey can't cause redundancy.

- Non trivial FD $\alpha \rightarrow \beta$ can cause redundancy iff α is not SK

- In BCNF every non-trivial FDs have SK on LHS. So, BCNF is free from FD caused redundancy.
3NF may have FD caused redundancies.

Ex. Courses (sid, Cid) Movies (sid, Movies)
One student can take many courses & one course can be taken by many students. Many-to-many relationship.
There is no redundancy in this table.
Similarly, no redundancy in Movies table. It is also many-to-many relationship.
If we merge them in one student tabele.
Student (sid, Cid, Movies).

C:

Sid	Cid
S_1	C_1
S_1	C_2
S_2	C_1
S_3	C_3

Movies:

Sid	Movie
S_1	A
S_1	B
S_1	C.

when merged:

S:

Sid	Cid	Movies
S_1	C_1	A
S_1	C_1	B
S_1	C_1	C
S_1	C_2	A
S_1	C_2	B
S_1	C_2	C

So, we can clearly see there is redundancy. Same info

repeats multiple times. But there's no non-trivial FDs in Student relationship. So, where is this redundancy coming from? This happened because more than one many-to-many relationship in a single table.
(Actually this creates a new type of dependency, called Multivalued Dependency (MVD))
This dependency says, every movie causes repetition of courses, & vice-versa.

- Redundancy
  - FD caused (removed by BCNF)
  - MVD caused (removed by 4NF)

* BCNF Decomposition

- A relation R is in BCNF iff for every non-trivial FD of R $\alpha \rightarrow \beta$ α is SK of R.
- A relation R with two attributes must be in BCNF.
- A set of relations $S = \{R_1, R_2, R_3 .. R_n\}$ is in BCNF iff $\forall i$ R_i is in BCNF.
- If a relation R is not in BCNF, we have to decompose it into subrelations such that each subrelation is in BCNF.

Ex. $R(A,B,C,D)$ $R_1(A,B)$, $R_2(C,D)$.
This is BCNF decomposition. But it is lossy. $R_1 \cap R_2 = \phi$.

- To check if a decomposed set of relation is in BCNF, we have to check if all relations are BCNF.

(So, if asked to decompose to BCNF, we must give a lossless decomposition. If decomposition is already given to check, then it may be BCNF but not lossless.)

Note: Question asks to find BCNF decomposition:

(1) Must be lossless.

(2) Try for dependency preservation, if possible.

Question asks if given decomposition is BCNF, then it doesn't matter if lossless or lossy.

(The same is true for 2NF & 3NF decomposition also)

- BCNF Decomposition Algorithm:
  - Guarentees lossless BCNF Decomposition

- For a relation R & FD set F, if R is in BCNF, stop. If R is not in BCNF, then for the violating FD: $X \to \alpha$ ($X \subset R$, $\alpha \in R$ & the FD is non trivial), split R into

$R_1 = X^+$ $\qquad$ $R_2 = (X - X^+) \cup X$

This is guarenteed lossless, since $R_1 \cap R_2 = X$ & $X \to R_1$.

Ex. $R(A, B, C)$ $\qquad$ $FD = \{A \to B\}$

$CK_S = \{AC\}$

So, $A \to B$ violates BCNF. $\qquad$ $\therefore R_1(A, B)$

$R_1 = A^+ = \{A, B\}$ $\qquad$ $R_2(A, C)$

$R_2 = (\{A, B, C\} - \{AB\}) \cup \{A\} = \{A, C\}$

Ex. $R(A,B,C,D)$ $F=\{A \to B ; A \to C\}$

$CK_s = \{AD\}$

So, $A \to B$ is violation: $A^+ = \{A,B,C\}$

$R_1 = \{A,B,C\} = A^+$

$R_2 = (R - A^+) \cup \{A\} = \{A,D\}$

$\therefore$ $R_1(A,B,C)$ $R_2(A,D)$

$F_1 = \{A \to B, A \to C\}$ $F_2 = \{\}$

Both in BCNF.

Ex. $R(A,B,C,D)$ $F=\{A \to B, B \to C\}$

$CK_s = \{AD\}$

So, $B \to C$ is violating.

$R_1 = B^+ = \{B,C\}$

$R_2 = (R - B^+) \cup \{B\} = \{A,B,D\}$

$R_1(B,C)$ $R_2(A,B,D)$

$F_1 = \{B \to C\}$ $F_2 = \{A \to B\}$

This is in BCNF. $CK_s = \{AD\}$

Not in BCNF. $(A \to B)$

$R_{21} = A^+ = \{AB\}$

$R_{22} = \{AD\}$

$\therefore$ $R_1(B,C)$ $R_{21}(A,B)$ $R_{22}(A,D)$

are all in BCNF.

Note: You may get different BCNF decomposition based on the order in which we resolve violation.

Ex. R(A, B, C, D, E)

$F = \{AB \to C, C \to D, D \to E\}$

$CK_s = \{AB\}$

So, $C \to D$ violates BCNF.

$R_1 = C^+ = CDE$

$R_2 = ABC$.

$R_1(C, D, E)$ $\qquad$ $R_2(A, B, C)$

$F_1 = \{C \to D, D \to E\}$ $\qquad$ $F_2 = \{AB \to C\}$

R_2 in BCNF. R_1 is not.

Violation: $D \to E$

$R_{11} = D^+ = DE$

$R_{12} = CD$

$R_{11}(C, D)$ $\qquad$ $R_{12}(D, E)$ $\qquad$ $R_2(A, B, C)$

All are in BCNF.

Ex. R(A, B, C, D, E)

$F = \{A \to BC, CD \to E, B \to D, E \to A\}$

$CK_s = \{A, E, BC, CD\}$

Violation: $B \to D$.

$R_1 = B^+ = BD$ $\qquad$ $F_1 = \{B \to D\}$

$R_2 = ABCE$ $\qquad$ $F_2 = \{A \to BC, E \to A, BC \to E\}$

$CK_2 = \{A, E, BC\}$

So, both are in BCNF.

Ex. $R(A, B, C)$ $F = \{AB \to C, C \to B\}$

$CKs = \{AB, AC\}$

R Violation: $C \to B$.

$R_1 = C^+ = \{C, B\}$ Both in BCNF.

$R_2 = \{A, C\}$

$F_1 = \{C \to B\}$ $F_2 = \{\}$

This is lossless, but not dependency preserving.

- A relation R with FD set F can always be decomposed into some lossless set of tables $\{R_1, R_2 .. R_n\}$, but it is not guarenteed that some dependency preserving BCNF decomposition exists for R.

So, for all relations some lossless BCNF decomposition always exist. For some relations, no lossless dep. preserving BCNF decomposition doesn't exist.

* 3NF Decomposition

- For 3NF Decomposition, we'll use "Synthesis Algorithm". It guarentees lossless & dependency preserving decomposition.

Although the BCNF decomp. algo. also is 3NF decomp., but it doesn't guarentee

- Synthesis Algo : Input: Relation R, FD set F
  - Find minimal cover of F, say F_c.
  - In F_c, if for two or more FDs have LHS same, merge them.

    $\alpha \to A$, $\alpha \to B \Rightarrow \alpha \to AB$.

    So, you'll have FDs in F_c of form $\alpha \to B$. $(\alpha, B \subseteq R)$.

- For every FD f_i in F_c : $\alpha \rightarrow \beta$ create a new relation R_i (.α.., ..β..)
- If none of the relations R_i, contains a CK, then take any one CK & make a new relation for it.
- Out of all new relations, delete any such R_i, for which there is some R_j, $R_i \subset R_j$, until no other relation can be deleted.

Ex. Give 3NF decomposition:

R(A, B, C, D, E)

F = { AB → C, C → B, A → D }

.Checking 3NF:

CKs = { AEC, AEB } N.P. Attrs = {D}

Not even in 2NF. AEB → D is partial dependency.

.Applying Synthesis algo:

S1: Finding Minimal Cover:

AB → C, C → B, A → D —(LHS minim.)→ AB → C, C → B, A → D —(Useless FD Removal)→ AB → C, C → B, A → D

S2: R_1 (A, B, C) R_2 (C, B) R_3 (A, D)

S4: R_1 (A, B, C) R_2 (B, C) R_3 (A, D) R_4 (A, C, E) ← CK.

S5: Remove R_2 ($R_2 \subset R_1$)

R_1 (A, B, C) R_2 (A, D) R_3 (A, C, E)

- Note that Synthesis Algo. doesn't care if the original relation R is in 3NF or not.
This algo is good, since algorithmically, checking if R is in 3NF or not is more expensive than just applying Synthesis Algo.
Hence it's not called 3NF Decomp. Algo, but 3NF Synthesis Algo.

- So, for ALL relation R, it is always possible to get a:
(1) Lossless BCNF decomposition.
(2) Lossless Dependency Preserving 3NF decomposition.

Ex. R(A,B,C,D,E,F) $F_s = \{A \to BCD, BC \to DE, B \to D, D \to A\}$
(We won't check for 3NF)

S1: Minimal Cover.

$A \to B$ $BC \to D$ $\Rightarrow$ $A \to B$ $B \to D$ $\Rightarrow$ $A \to B$ $B \to D$
$A \to C$ $BC \to E$ $A \to C$ $B \to E$ $A \to C$ $B \to E$
$A \to D$ $B \to D$ $A \to D$ $D \to A$.
$D \to A$ $D \to A$

S2: $A \to BC$, $B \to DE$, $D \to A$ S3: $R_1(A,B,C)$ $R_2(B,D,E)$ $R_3(A,D)$
S4: $CK_s = \{FA, FB, FD\}$
$\Rightarrow$ $R_1(A,B,C)$ $R_2(B,D,E)$ $R_3(A,D)$ $R_4(A,F)$.
S5:
$R_1(A,B,C,D)$ $R_2(B,D,E)$ $R_3(A,F)$. $R_4(A,D)$.

Ex. R(A,B,C,D) F = $\{AB \to C, BC \to D, CD \to A, AD \to B\}$
S1: $AB \to C$ $AB \to C$ $AB \to C$
$BC \to D$ $\Rightarrow$ $BC \to D$ $\Rightarrow$ $BC \to D$
$CD \to A$ $CD \to A$ $CD \to A$
$AD \to B$ $AD \to B$ $AD \to B$
$CK_s = \{AB, BC, AD, CD\}$

S_3: $R_1(A,B,C)$ $R_2(B,C,D)$ $R_3(A,C,D)$ $R_4(A,B,D)$

$S_{4,5}$: $R_1(A,BC)$ $R_3(A,C,D)$

$R_2(B,CD)$ $R_4(A,B,D)$

* 2NF Decomposition

- In a relation R, 2NF violation is caused by presence of FDs of form:

  Proper subset of CK → NPA

  as this type of FD makes some CK → NPA, a partial dependency.

- For a relation R with FDSet F, if any FD $\alpha \to A$ in F^+ violates 2NF, then decompose: $\{\alpha \subseteq R, A \in R\}$

  $R_1 = \alpha^+$

  $R_2 = (R - \alpha^+) \cup \alpha$

- This guarentees lossless ↑ DP 2NF decomposition.

Ex. R(A, B, C, D, E) F = {AB → C, C → B, A → D}

CKs = {ABE, ACE} N.P.Attr = {D}

Not in 2NF, due to A → D.

2NF Violation: A → D

$R_1 = A^+ = AD$ $F_1 = \{A \to D\}$ ✓ (In 2NF)

$R_2 = ABCE$ $F_2 = \{AB \to C, C \to B\}$

$CKs_2 = \{ABE, ACE\}$ ∴ No NPAttr, so in 2NF.

- So, for every rel. R we can guarentee a lossless dependency preserving 2NF decomposition.

* In a database, only insertion or modification can violate a FD.

* A ~~dep~~ decomposition is Dependency Preserving iff all the FDs in the database can be enforced ~~only~~ without any join, i.e. with help of any individual table (not two or more).

* In general we prefer 3NF, (not BCNF), since we can guarentee a lossless D.P. Decomposition.

Queries

- Schema tells us the name of relation, the name of the attributes and their types.
  Instance on the other hand ✱ has the actual tuples of the relation.
  Schema is also called Intension.
  Instance is also called Extension.

- EF Codd in his paper gave "Relational Model", along with two Query Languages (both theoretical):
  (1) Relational Algebra
  (2) Tuple Relational Calculus

- Query Languages: Specialised languages designed specifically to ask questions related to data in the database.

- The Query Languages of interest for Relational Model are:
  (1) Relational Algebra } Theoretical
  (2) Tuple Relational Calculus
  (3) Structured Query Language. } Practical.

- Procedural QL vs Declarative QL

A function ✱ tells us the mapping of input to output. An algorithm gives step-by-step instruction for getting the output for a input.
Ex. f: Sort Inp: 2,1,3 → Out: 1,2,3
Algo: MergeSort: Do this, then that, then....

Similarly, Procedural QL require the user to tell what they want & how to get it. Relational Algebra is Procedural QL.

Declarative QL only requires the user to tell what they want. Relational Calculus is such language. SQL too.

So, for user, Declarative QL is better. For DBMS software, Procedural QL is better.

In SQL actually, the user query (which is declarative) is converted to equivalent relational algebra query (which is procedural) by Query Processor & further optimized by Query Optimizer.

* Relational Algebra

Relational Algebra		Algebra
Schema	↔	Variables
Instances	↔	Values
Operators: Basic: ↓ Derived:	↔	Operators: Basic: +, - ↓ Derived: ×, ÷, etc.
Ex. $R_1 \cup R_2$	↔	Ex. $a + b$
Expr. is eval. based on instances		Expr. will be eval. based on value of a & b.

- For some query Q in RA, inputs are instance(s), and output would be ~~some~~ instance. Query Q itself is written in terms of schema & operators.
Sometime for complex queries we require intermediate results, that also is instance.

- In a relation, the position of attributes are fixed and they can be referred to by their positions.

  Ex. R(A, B, C)

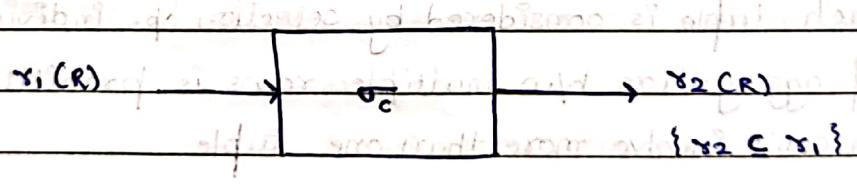
  R.A ≡ R.1 , R.B ≡ R.2 , R.C ≡ R.3.

- Basic Operators in RA: Select (σ), Project (π), Cartesian Product (x), Set Difference (−), Union (∪), Rename (ρ)

Derived Operators in RA: Set Intersection (∩), Natural Join (⋈), Conditional Join (⋈c), Division (÷), etc.

Selection Operator (σ_c)

- Unary operator, which, based on some condition c, selects tuples from input instance & discards the others.



σ_c here selects tuples from $r_1(R)$ which satisfy the condition c.

Ex. R(roll, name, gpa)

- $\sigma_{gpa>7.5}$: Select all tuples with gpa > 7.5.
  But this isn't complete query.
- $\sigma_{gpa>7.5}(R)$: Select all tuples from R with attr gpa > 7.5.
  The output instance would have schema : R(roll, name, gpa).
- $S = \sigma_{gpa>7.5}(R)$: Select all tuples from R (or instance of schema R) with gpa > 7.5. The outp. instance would have schema : S(roll, name, gpa).

- So selection operator without assignment doesn't change schema. Also it doesn't change the degree, but cardinality of outp. may differ from input.

- Selection operator applies on each tuple individually in the input instance.

- The condition c in σ_c is propositional logic formula, and can have connectives such as $\wedge, \vee, \neg$. It can have conditions on attributes of a tuple, such as $>, <, \geq, \leq, =, \neq (x = y)$

Ex. User (uid, name, age, pop).

Q: Select users with popularity at least 0.9, and age under 10 or above 12.

$$\sigma_{(pop \geq 0.9) \wedge (age < 10 \vee age > 12)} (User)$$

- Since each tuple is considered by selection op. individually any form of aggregation b/w multiple rows is prohibited. So selection condition can't involve more than one tuple.

Ex. $\sigma_{pop \geq max(pop)} (User)$

This is wrong. max(prop) requires more than one tuple at a time.

- So, it does horizontal partitioning.

• Projection Operator (Π_a)

- Unary operator which projects specified attributes from a given relation. If after projection, some tuples are duplicate it will only keep single ~~one~~ copy of that tuple.

Ex. R:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Q: $\Pi_{(A,C)}(R)$ →

A	C
α	1
β	1
β	2

- So, it is doing vertical partitioning.
- Again, assignment such as, $S = \Pi_{(A,C)}(R)$, would change the name of output's schema, from R to S.

- Importantly, any operation in RA on schema & instances stored in DB, doesn't affect the original tables, rather all outputs are temporary copy.
- If not assigned to any var, the result of a query won't be further usable. If assigned, the var name can be used in further queries.
- The output instance would have attributes in the order as specified in projection operator's attribute list.

Ex. $R(A, B, C)$ $\quad S = \Pi_{(C,A)}(R) \quad$ $S(C, A)$

- So, the degree of output of $\Pi_{\langle a \rangle}$ would be same as $|\langle a \rangle|$ & may differ from inp.'s ~~one~~ degree. Cardinality of out. may also differ from inp's (in case of duplicate tuples).

Ex. To make sure the cardinality of projected instance is necessarily same as inp. instance, what can you say about the attribute list of proj. operator?

The attribute list must be some superkey of the input's schema.

Ex. R(a, b, c). FDs = { a→b, bc→a }. Which of the following queries are guaranteed to return same no. of tuples for any inp. instance r of R?

- (i) $\Pi_a(r)$ (ii) $\Pi_{a,c}(r)$ (iii) $\Pi_c(r)$ (iv) $\Pi_b(r)$
(v) $\Pi_{a,b}(r)$ (vi) $\Pi_{a,b,c}(r)$

CKs = { ac, bc } So, (ii), (vi) are guaranteed to have same no. of tuples as input.

- Properties of Selection & Projection:

1. Selection is commutative: $\sigma_A(\sigma_B(R)) \equiv \sigma_B(\sigma_A(R))$
since ultimately we have tuples which pass both conditions.
$\equiv \sigma_{A \wedge B}(R)$

2. Projection is not commutative: $\Pi_{(A)}(\Pi_{(A,C)}(R)) \not\equiv \Pi_{(A,C)}(\Pi_{(A)}(R))$
↑ invalid query

3. If $X, Y \subseteq R$; $X \subseteq Y$, then, $\Pi_{(X)}(\Pi_{(Y)}(R)) \equiv \Pi_{(X)}(R)$

Ex. R (sid, sname, rating, age)

Select the name & rating of all students with rating > 8.

$\Pi_{(sname, rating)} (\sigma_{rating > 8} (R))$

4. We can perform the two operations on relation R:

(1) Select tuples based on C

(2) Project attributes of list A

in either order iff C uses attr. only in the list A.

i.e. $\sigma_C (\Pi_A (R)) \equiv \Pi_A (\sigma_C (R))$

iff all attr. condition C uses are present in list A.

Ex. In general which transformation (substitution) is valid:

(A) $\Pi_A (\sigma_C (R)) \longrightarrow \sigma_C (\Pi_A (R))$ (Subs. LHS with RHS)

(B) $\sigma_C (\Pi_A (R)) \longrightarrow \Pi_A (\sigma_C (R))$

B is valid. If $\sigma_C (\Pi_A (R))$ is a valid query it means C uses only attr. available in A, so, $\Pi_A (\sigma_C (R))$ would also be valid since R is superset of A.

Union Operation (∪), Set Difference (-), Intersection (∩)

- Binary operations which just behave like set operations. Since instance of any relation simply is a set of tuples, these operations behave simply as set operations.

- For these operations b/w two relations, R & S, they must have compatible schema (a.k.a. Union Compatible Schema).

Relation R is Union Compatible with relation S, iff

(1) Degree of both relations is same.

(2) The domain of i^{th} attribute of both relation must be same.

(Name of attr. may differ)

Ex. R:

A	B
α	1
α	2
β	1

S:

B	D
α	2
β	3

R ∪ S:

A	B
α	1
α	2
β	1
β	3

R − S:

A	B
α	1
β	1

S − R:

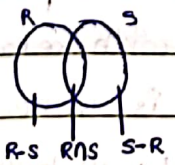
B	D
β	3

S ∩ R:

B	D
αB	2

- The schema of output instance $ will be same as the schema of first operand.
- Union is commutative (we don't worry about output schema, but only output data).
- Set Difference is not Commutative : $R - S \neq S - R$ (since data may differ)
- Intersection is also commutative (although schema of o/p may differ, but data is same) : $R \cap S \equiv S \cap R$.
- Intersection is actually a derived operator :

$$R \cap S \equiv [(R \cup S) - (R - S)] - (S - R) \equiv [R - (R - S)] \overset{\text{B}}{\equiv} [S - (S - R)]$$



- Cross Product (×)
- Binary operation

Ex. Student_Course (Sid, Sname, Cid, Cname).

Get Sid, Sname of all students who have registered in "DB" course but not in "Algo" course.

$$R = \Pi_{(Sid, Sname)}(\sigma_{Cname="DB"}(S\_C)) - \Pi_{(Sid, Sname)}(\sigma_{Cname="Algo"}(S\_C))$$

- Cross Product (×)
  - Binary operation.
  - R×S, pairs every tuple of R with every tuple of S.
  - The schem of output instance for R×S would consist of all attr. of R with all attr. of S in order.
  - suppose R(A, B), S(C, B, D), then there will be confusion in output of R×S, say X(A, B, C, B, D).

    Convention 1: X(R.A, R.B, S.C, S.B, S.D) or
    X(A, R.B, C, S.B, D)

    Convention 2: X(A 2 C 4 D) or
    X(1 2 3 4 5).

    (we generally follow convention 1)

Ex. R:

A	B
α	1
β	1

S:

B	C
a	+
b	-
a	-

R×S:

A	R.B	S.B	C
α	1	a	+
α	1	b	-
α	1	a	-
β	1	a	+
β	1	b	-
β	1	a	-

- For $R \times S$,

  $|R \times S| = |R| \times |S|$

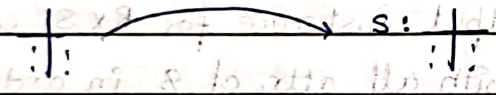
  & $\deg(R \times S) = \deg(R) + \deg(S)$

- Cross product is commutative: $R \times S = S \times R$, since same data, even though diff. schema.

- Rename Operator (ρ)-

  - Unary operator
  - Renames the given input & after creating its copy.

  $\rho_s(R)$:



  - Renaming can be done for schema, and optionally for attributes also.

  Both schema renaming & attr. renaming are optional.

Ex. R(A, B, C)

Change Schema Name (copy in main m/m): $\rho_s(R)$

Change Schema & all attr. (——11——): $\rho_{s(1,2,3)}(R)$

Change Sch. name & some attr (——11——): $\rho_{s(A \to 1, C \to 2)}(R)$

Change only attr name (——11——): $\rho_{(A \to 1, C \to 5)}(R)$

Since renaming just copies the rel. R to main m/m & doesn't affect on disk, we can skip renaming of S (this won't create two copies of S in main m/m).

Ex. Follows (Uid1, Uid2). $\langle x,y \rangle \in$ Follows if x follows y.

(a) Find all such pairs $\langle x,y \rangle$ where x follows y & y follows x.

$$\Pi_{(F_1.Uid1,\ F_1.Uid2)} \left(\sigma_{(F_1.Uid1 = F_2.Uid2) \wedge (F_1.Uid2 = F_2.Uid1)} \left(\rho_{F_1}(\text{Follows}) \times \rho_{F_2}(\text{Follows}) \right) \right)$$

(b) Find uid of those who follow at least two people.

$$\Pi_{(F_1.Uid1)} \left(\sigma_{(F_1.Uid1 = F_2.Uid1) \wedge (F_1.Uid2 \neq F_2.Uid2)} \left(\rho_{F_1}(\text{Follows}) \times \rho_{F_2}(\text{Follows}) \right) \right)$$

(c) Find uid of those who follow exactly one person.

$$\rho_{(Uid)} \left(\Pi_{(Uid1)} (\text{Follows}) \right) - \rho_{(Uid)} \left(\begin{matrix} \text{Follows} \\ \text{Atleast two} \\ \text{people} \end{matrix} \right)$$

• Join Operations

- Usually performing cross product (×) such as A×B results in a lot of useless tuples & rarely tells us the required information.
- So, cross product is generally followed by selection (σ) or Projection (Π) operations.
- Since cross product followed by selection is so common, we've created new operator : Join. (⋈).
- Cross Prod followedy by Selection = Conditional Join
  $\{ \sigma_c (R \times S) \}$ $\{ R \bowtie_c S \}$
- There are two types of Join Operators:
  (1) Inner Join (or Join): Conditional Join / Theta Join ($\bowtie_c$), EquiJoin, Natural Join ($\bowtie$)

(2) Outer Join: Left OJ (⟕), Right OJ (⟖), Full OJ (⟗)

- Conditional Join ($\bowtie_c$)
  - Pair rows of R with rows of S which satisfies the condition(c): $R \bowtie_c S$
  - Practically the DBMS, doesn't convert $\bowtie_c$ to $\sigma_c(x)$. There are join algorithms which directly perform the operation.

Ex. R:

A	B	C
1	2	4
2	5	5
3	8	6

S:

D	E
3	1
6	2

$R \underset{(R.B < S.D)}{\bowtie} S$:

A	B	C	D	E
1	2	4	3	1
1	2	4	6	2
2	5	5	3	1

So, R has a "dangling row", since it doesn't pair with any row of S, (3, 8, 6).

- Equi Join
  - Conditional Join with only equality conditions.

Ex. $R \bowtie_{(R.1 = S.2)} S$

$R \underset{(R.B = S.E)}{\bowtie_{(R.A = S.C) \wedge}} S$

- No other conditional operator is allowed.

- Natural Join ($\bowtie$)
  - Performs join based on only the name of attributes (not the domain).
  - Say $R(A,B,C)$ & $S(B,C,D)$ & we write: $R \bowtie S$, then it means join R & S based on equality of all common attributes, here (B,C).
    The schema of $R \bowtie S$ would be (A,B,C,D), (i.e. no redundant attribute).
    So, $R \bowtie S$ would behave as $R \underset{(R.B=S.B) \wedge (R.C=S.C)}{\bowtie} S$
  - If no common attr. b/w R & S by name: $R \bowtie S = R \times S$
  - $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
- Division ($\div$)
  - In Number Theory $\frac{N}{D} = Q$, Q is the largest integer such that $Q \times D \leq N$.
  - The Division operation in Relation Algebra behaves quite similarly, for relation R & S, $R/S = Q$, where Q is the largest relation such that $Q \times S \subseteq R$.

Ex. $R(a,b) = \{(1,1), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,3), (3,4)\}$
$S(b) = \{(1), (3)\}$
$Q(R/S)$ has to satisfy: $Q \times S = R$. ∴ Schema for Q must be (a).
$R/S = \{(2), (3)\} = Q$
Let's check. $Q(a) \times S(b) = \{(2,1), (2,3), (3,1), (3,3)\}$
So, this is subset of R.
Also, this Q is largest possible.

Ex. $R(A,B,C,D,E) = \{(\alpha, a, \alpha, a, 1), (\alpha, a, \gamma, a, 1), (\alpha, a, \gamma, b, 1), (\beta, a, \gamma, a, 1), (\gamma, a, \gamma, a, 1), (\gamma, a, \gamma, b, 1), (\gamma, a, \beta, b, 1)\}$
$S(D,E) = \{(a,1), (b,1)\}$
$Q_{(A,B,C)} = \{(\alpha, a, \gamma), (\gamma, a, \gamma)\}$

- For R/S to be defined, the attribute set S must be subset of attribute set R.

Ex. Division operation is valid for:

(A) R(A,B,C) S(B,C)

(B) R(A,B) S(B,C)

Ex. For any relation R & S what is $(R \times S) \div S$

R

Ex. If $Y \times S \subseteq R$, then $R/S = Y$?

No. Y should be the largest set, such that $Y \times S \subseteq R$

- We can rewrite Division operator in terms of basic operators:

$$R/S = \underbrace{\Pi_{(R-S)}}_{\text{attribute set}}(R) - \left\{ \Pi_{(R-S)} \left[\left(\Pi_{(R-S)}(R) \times S \right) - R \right] \right\}$$

(Healy Implementation)

- Let S[i] represent ith tuple in S.

$$R/S = \bigcap_{i=1}^{n} \Pi_{(R-S)}(S[i] \bowtie R)$$

OR

$$R/S = \bigcap_{\forall\, b_i \in S} \Pi_{(R-S)} \left(\sigma_{(R-S = b_i)}(R) \right)$$

(Maier's Implementation)

Ex. Given R, S, $|R| = m$, $|S| = n$, $(m, n > 0)$, then what is the max. & min cardinality of $R \div S$.

Min: 0, when no (R-S), such that all R tuples are satisfied.

Max: $\left\lfloor \frac{|R|}{|S|} \right\rfloor$. when all values in (R-S) schema satisfy all R tuples.

Ex. $R_{(A,B)} = \{(a,2), (a,1), (a,3), (b,1), (b,2)\}$

$S(B) = \{(1), (2)\}$

$R/S = \{(a), (b)\}$

$\therefore |R/S| = \left\lfloor \frac{5}{2} \right\rfloor = 2.$

Ex.. In R/S, if $|R| = m, (m > 0)$, $|S| = 0$, then $|R \div S|$?

We can use Healy's def^n. $\prod_{(R-S)}(R) - \left[\prod_{(R-S)} \left(\prod_{R-S}(R) \times S - R \right) \right]$

The cardinality $\left| \prod_{(R-S)}(R) \times S \right| = |T| \times |S| = |T| \times 0 = 0.$

The card. $|M - R| = 0.$

$\therefore$ Ultimately, $\prod_{(R-S)}(R) - \underbrace{[\ldots]}_{\{0 \text{ card.}\}} = \prod_{(R-S)}(R)$.

So. at max, this can be $= |R| = m$.

at min, this can be $= \left| \prod_{(R-S)}(R) \right|_{\min} = 1$

. If $|R| = 0$

Then $|R/S| = 0$

- Outer Join
  - It includes the dangling tuples with help of NULL.
  - Left Outer Join ($\bowtie$)
    R $\bowtie$ S includes the dangling tuples of R.
  - Right Outer Join ($\bowtie$)
    R $\bowtie$ S includes the dangling tuples of S.
  - Full Outer Join ($\bowtie$)
    R $\bowtie$ S includes the dangling tuples of both R & S.

Ex. R:

A	B
1	2
3	4

S:

B	C
2	5
6	9

R $\bowtie$ S:

A	B	C
1	2	5

R $\bowtie$ S:

A	B	C
1	2	5
3	4	NULL

R $\bowtie$ S:

A	B	C
1	2	5
NULL	6	9

R $\bowtie$ S:

A	B	C
1	2	5
3	4	NULL
NULL	6	9

- By default Outer Join ≡ Full Outer Join ($\bowtie$ or $\bowtie$)
- By default, we first do natural join then add dangling tuples for Outer Join.
- We can also have conditional outer join. Include all rows which satisfy condition, then include dangling tuples (based on left, right, full).

* Tuple Relational Calculus

- In RA, we write queries which tell, what we want & the order in which operations must be performed to get the results. So, RA is a procedural language.

- TRC queries just tell what we want, and not the order of the operations to be performed to get that. So TRC is Declarative Language.

- In TRC, we represent tuples as variable, so every variable in TRC can be replaced with tuple.
To which relation the tuple belongs can be indicated as:
$t \in R$ or $R(t)$
The attribute of tuple can be represented as: $t.A$ or $t[A]$.
The queries in TRC are of the form: $\{t \mid P(t)\}$ where $P(t)$ is predicate on t.

- There are two ways of writing TRC queries:
(1) We first come up with logic, then get results.
(2) We first picturise the results, then get derive logic for it.

Ex. Write SELECT operation of RA (σ) in TRC.
say, R(A, B) RA Query: $\sigma_{B=17}(R)$
TRC Query: $\{t \mid t \in R \wedge t[B] = 17\}$

Ex. Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day)

(1) Find all sailors w/ rating > 7.

RA: $\sigma_{rating>7}(S)$

TRC: $\{t \mid t \in S \wedge t[rating] > 7\}$

Ex. Write Projection o/p^n in TRC.

Say, R(A, B, C)

RA: $\Pi_A(R)$ RA: $\Pi_{A,C}(R)$

TRC: $\{t.A \mid t \in R\}$ TRC: $\{t.A, t.C \mid t \in R\}$

This is way 1.

Now, suppor r(R) is an instance:

r:	A	B	C
	α	10	1
	α	20	1
	β	30	1

We perform $\Pi_A(r)$

r': A
α
β

why (α,) is there in r'? Because there exists a tuple in r such that $t.A = \alpha$.

Similarly for (β,) tuple in result r'.

So,

$\{t \mid \exists_{x \in r}\ (x.A = t.A)\}$

there is no info, where tuple t belongs to. but since with 't' we only use attr. A, we can say schema of t is (A,) only.

Suppose we wanted query $\Pi_{A,C}(r)$:

$\{t \mid \exists_{x \in r}\ [(x.A = t.A) \wedge (x.C = t.C)]\}$

∴ schema of t is (A, C).

Ex. R(A,B) $\sigma_{B=17}(R)$

Way1: $\{t \mid t \in R \wedge t.B = 17\}$

Way2: $\{t \mid \exists x \in R\ (t.A = x.A \wedge t.B = x.B \wedge x.B = 17)\}$

↑ this t represents the tuple in result table.

Schema of t : (A, B)

Ex. R(A,B) $\Pi_A(\sigma_{B=17}(R))$

Way1: $\{t.A \mid t \in R \wedge t.B = 17\}$

Way2: $\{t.A \mid \exists x \in R\ (t.A = x.A \wedge x.B = 17)\}$

Ex. R(A,B), S(C,D) (R & S are union comptabile)

① R ∪ S

W1: $\{t \mid t \in R \vee t \in S\}$

W2: $\{t \mid \exists x \in R\ (t.A = x.A \wedge t.B = x.B) \vee \exists x \in S\ (t.A = x.C \wedge t.B = x.D)\}$

② R ∩ S

W1: $\{t \mid t \in R \wedge t \in S\}$

W2: $\{t \mid \exists x \in R\ (t.A = x.A \wedge t.B = x.B) \wedge \exists x \in S\ (t.A = x.C \wedge t.B = x.D)\}$

③ R − S

W1: $\{t \mid t \in R \wedge t \notin S\}$

W2: $\{t \mid \exists x \in R\ (t.A = x.A \wedge t.B = x.B) \wedge \forall x \in S\ (t.A \neq x.C \vee t.B \neq x.D)\}$

Ex. R(A,B) S(C,D,E) R × S

W1: $\{t_1, t_2 \mid t_1 \in R \wedge t_2 \in S\}$

W2: $\{t \mid \exists x \in R\ \exists y \in S\ (t.A = x.A \wedge t.B = x.B \wedge t.C = y.C \wedge t.D = y.D \wedge t.E = y.E)\}$

Note: In Way2, the P(t) predicate answers the question, "why the tuple t is in the resultant relation?")

Ex. R(A,B) S(B) $R \div S$

W2: $\{ t \mid \forall x \in S \; \exists y \in R (x.B = y.B \wedge t.A = y.A) \}$

W1: $\{ t.A \mid t \in R \wedge \forall x \in S \; \exists y \in R (x.B = y.B \wedge t.A = y.A) \}$

Ex. R(A,B) S(A,D) $R \bowtie S$

W1: $\{ t_1.A, t_1.B, t_2.D \mid t_1 \in R \wedge t_2 \in S \wedge t_1.A = t_2.A \}$

W2: $\{ t \mid \exists x \in R (t.A = x.A \wedge t.B = x.B) \wedge \exists x \in S (t.A = x.A \wedge t.D = x.D) \}$

- The benefit of way 2 of TRC queries is that there's always only one free variable.

- Safe vs Unsafe Query
  - Tuples in the database instance are real data.
  - If a Query gives some data which isn't present in the database (or can't be derived from it), then the Query is unsafe.
    Else the query is safe.
  - Also, if Query gives infinite result, then its also unsafe (since database instance is finite).
    Ex. $\{ t \mid t \notin R \}$
    Query gives inf. result ⟶ Query is unsafe
    { One way }

- Expressive Power of TRC
  We have shown that for every Rel. Algebra operator we have equivalent TRC Query (safe query), so,
  RA ≡ Safe TRC
  in terms of expressive power.
  { TRC (safe + unsafe) has higher expressive power than RA }

* Structured Query Language (SQL)

- Declarative Query Language.
- It has multiple subsets based on query type:
  - Data Definition Language: For creation, deletion & modification of def^n of tables & views. Constraints can also be included here.
  - Data Manipulation Language: For insertion, deletion, modification of data. Also to query/retreive data.

(Note: In GATE, retreival of data is of concern.)

- Although motivated from theoretical query lang., such as RA & TRC, SQL has some major differences. In RA:
  - Two rows couldn't be entirely same.
  - There is exactly one PK (at a time).
  - There is ≥ 1 CK & ≥ 1 SK for relation.
  - Relation is a set of tuples.
  - Don't have NULL in table (except for outer join).

Set	vs	Multiset
- Unordered.		
- Duplicates not allowed		- Duplicates allowed.
$\{1,2,2\} = \{1,2\}$		$\{1,2,2\} \neq \{1,2\}$.
$\lvert\{1,2,2\}\rvert = \lvert\{1,2\}\rvert = 2$		$\lvert\{1,2,2\}\rvert = 3$.

In SQL, Table is equivalent to multiset.

- In SQL,
  - Two rows may have exactly same value (duplicates allowed).
  - May/Maynot have PK, CK, SK.
  - NULL values are allowed.

Note: Although SQL keywords are case-insensitive, we'll write them in BLOCK letters.

- Basic SQL Query:

```
SELECT  A1, A2, A...    ← Attribute selection
FROM    R1, R2, R3...   ← Conceptually Crossprod of R1 × R2 × R3...
WHERE   -condition-     ← [optional] tuple selection condition
```

(SELECT, FROM are mandatory)

Ex. R(A, B), S(C, D)

① SELECT A, D
FROM R, S
WHERE (A>10) AND (C>5)

② SELECT DISTINCT A, D
FROM R, S
WHERE (A>10) AND (C>5)

① similar to: $\Pi_{A,D}(\sigma_{(A>10)\wedge(C>5)}(R \times S))$, but ① is SQL query & will allow duplicates.

② is exactly same as: $\Pi_{A,D}(\sigma_{(A>10)\wedge(C>5)}(R \times S))$, since it doesn't allow duplicates by use of "DISTINCT".

- In SQL also, if you say (X, Y, ...) is primary key, then none of them can have NULL value. (Entity IC)
And obviously, it can't have exactly same two tuples.

Ex. Suppose in above query, R has PK A & S has PK C, D, do we still need DISTINCT in query to avoid duplicate tuples?
Yes. Still dup. tuples can be there in result (without DISTINCT). Ex.

R: A	B
110	1
120	2

S: C	D
21	1
22	1

Ex. Same Query, But "SELECT A, C, D ..." Do we need DISTINCT Keyword to avoid duplicate.

No. Since A is PK in R & C, D is PK in S, no duplicate tuple will result from this query.

Note: SELECT $A_1, A_2, A_3 \ldots$ FROM $R_1, R_2, \ldots$ WHERE cond } This can be thought of as: projecting $A_1, A_2 \ldots$ from the crossprod. $R_1 \times R_2 \ldots$ based on condition 'cond'.

- Putting * after SELECT, means project all attributes in $R_1 \times R_2 \times \ldots$

Ex. SELECT * FROM Student S;
would give all tuples, all attributes of Student table.

- When we run SQL data retrieval query, original data doesn't change. It exists on disk. It is copied to main m/m & op^n are performed on the copy.

- Once we write "... FROM Student S ...", i.e. alias Student to S, we can't use "Student" in the query, only the alias "S" can be used.

Ex. "SELECT Student.Name FROM Student S;" is wrong.

- The way we understand SQL query evaluation is not how it actually happens.

Ex. SELECT A FROM R, S WHERE c

Our Understanding: ① $X = R \times S \rightarrow$ ② $Y = \sigma_c(X) \rightarrow$ ③ $Z = \Pi_A(Y)$

This isn't what SQL does internally, although this is how we understand it.

But the actual op" will produce the same result (but it uses more optimized algorithms).

Ex. Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day).

(1) Find name & age of all sailors.
SELECT sname, age FROM Sailors;

(2) Find all sailors above rating 7.
SELECT * FROM Sailors WHERE rating > 7;

- ~~DISTINCT~~
- DISTINCT : Whenever used after SELECT keyword, it works on th entire tuple for each tuple in result.

Ex. SELECT DISTINCT A,C FROM ....
would compare (A,C) for each tuple to be distinct, not A, not C separately.
i.e. two tuples are non-distinct t_1, t_2, iff:
$t_1[A] = t_2[A]$ AND $t_1[C] = t_2[C]$

- AS : Optional keyword for aliasing.

"... FROM Sailors S" ≡ "... FROM Sailors AS S"
Helpful when same table used two or more times in single query.

Ex. contd.
(3) Find name of sailors who have reserved boat with id 103.
SELECT Sname FROM Sailors S, Reserves R WHERE S.Sid = R.Sid AND R.bid = 103;

(4) Find Sailor names who have reserved at least one boat.

```
SELECT sname FROM Sailors S, Reserves R WHERE S.sid = R.sid;
```

(5) "SELECT S.sid FROM Sailors S, Reserves R WHERE S.sid = R.sid;"

In this query, does adding DISTINCT make any difference?

Yes. Although S.sid is PK, but after cross-prod with R, we have multiple ~~entries~~ repetition of S.sid.

Ex. S:

sid	sname ...
212	A
231	B

R:

sid	bid ...
212	103
212	104
231	103

(6) Find Sname of Sailors who've reserved red boat.

```
SELECT sname FROM Sailors S, Reserves R, Boats B WHERE
S.sid = R.sid AND R.bid = B.bid AND B.color = "red";
```

(7) Find color of boats reserved by Sailor named "Lubber".

```
SELECT color FROM Sailors S, Reserves R, Boats B WHERE
S.sname="Lubber" AND S.sid = R.sid AND R.bid = B.bid
```

- In SQL Union is performed using Keyword UNION, difference with EXCEPT (or MINUS in ORACLE), and intersection using INTERSECT. These keywords also require union compatible table. Although the input tables may have duplicates, the output will only have unique tuples. Imp

Ex. R: A S: B

a_1	a_2
a_2	a_3
a_2	a_3
a_3	a_2

. (SELECT * FROM R) UNION (SELECT * FROM S);

R∪S:

A
a_1
a_2
a_3

. (SELECT * FROM R) INTERSECT (SELECT * FROM S);

R∩S:

A
a_2

. (SELECT * FROM R) EXCEPT (SELECT * FROM S);

R-S:

A
a_1

There are also versions of these operations which don't remove duplicates. UNION ALL, INTERSECT ALL, EXCEPT ALL. They perform frequency based operations.

. UnionAll ⇒

A
a_1
a_2
a_2
a_2
a_3

. Intersect All ⇒

A
a_2
a_2

{ at most two a_2's are common. S doesn't have a third. }

. Except All ⇒

A
a_1
a_2

{ Two a_2 get removed, one remains. }

. Can't write R UNION S directly. Invalid.
(SEL * FROM R) UNION (SEL * FROM S);

- Note: Take care of union, intersection & except when performing on Key vs non-Key attributes.

Ex. [contd.]

(8) Find Sid of Sailors who've reserved a red or a green boat.

① SELECT sid FROM Reserves R, Boats B WHERE (R.bid = B.bid) AND (B.color = "red" OR B.color = "green");
{This will give non-distinct}

② (SELECT sid FROM .... WHERE R.bid = B.bid AND B.color = "red")
UNION
(SELECT sid FROM ... WHERE .... B.color = "green")
{This will give distinct results}

① with DISTINCT keyword would be equivalent to ②.

(9) Find Sid of Sailors who've reserved both a red & a green boat.

(SELECT sid FROM Reserves R, Boats B WHERE (R.bid = B.bid) AND B.colors = "red")
INTERSECTS
(SELECT sid .... FROM .... WHERE ... AND B.color = "green")

(10) Find Sname of Sailors who've reserved both a red & a green boat.

Now the previous approach won't work, since Sname isn't Key attr. Suppose <sid: 112, sname: A> has hired red boat & <Sid: 223, Sname: A> has hired green. If we intersect the (Sname hired red) ∩ (Sname hired green), we would get A in result, which is wrong. So, one approach is using "nested query", which will come later. Another is to intersect <sid, sname> pairs.

(SELECT S.sid, S.sname FROM Sailors S, Res R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = "red") INTERSECT (... B.color = "green");