18/8/20.

DBMS

* DBMS is a software, not hardware.

DBMS software → client requirement → E.R diagram
↓
R.M
(Relational Model)

① * Data : Any fact which can be recorded or stored for later use is called data.

* Information : It is the relevant data.

② Data Base : It is a collection of inter-related data.

③ DBMS provides → Storing the data
→ Retrive the data
→ Manipulate the data
} has to be efficient and convenient

④ DDL → Data Definition Language
Using DDL we create data.

⑤ DML → Data Manipulate Language
Used to retrive & manipulate data.

⑥ DCL → Data Control Language
Grant & Revoke permission to other users.

DBMS uses the above languages.

* DBMS developed has to be efficient → space efficient
→ time efficient
&
convenient → user-friendly.

ER-Diagram :-

Entity Relationship Diagram.

Applications of Database :-

* In universities to keep a track of students records.
* Online shopping.
* IRCTC bookings, transactions.

* Banking, Net-banking.

DB / MS

This management involves ....
- storage type, defining structures for storage of information
- providing mechanisms for manipulation of information
- the base system should ensure safety of information.
→ If any data or transaction crashes, then the role of database is to recover or undo it.
- Concurrent access (simultaneously) - This occurs due to race condition & has to be avoided.

Applications of DBMS:-

① Enterprise information
- Sales
- Accounting
- Human Resources
- Manufacturing
- Online retailers

② Banking and Finance
- Banking
- Credit card transactions
- Finance

③ Universities
④ Airlines / IRCTC for reservations
⑤ Telecommunication (ex: data consumption per day).
→ checking

⇒ Concurrent Access example:-

Suppose A is constant. A = 100

And B and C want to approach A. B = -5, C = -10

* If they go serially (one after other), then the final value of A is 85.
* But if they go concurrently (simultaneously), then the value of A depends on the last entered one.
~~First B, then C~~ ⇒ Both B and C read 100.
So if B goes it becomes 95, and then if C enters, it overwrites 95 and value changes to 90. But it's never 85
⇒ Ex: in IRCTC bookings, for 1 train many people make reservations and hence availability/visibility of berths might be a problem due to race condition. So concurrent access has to be avoided.

Before DBMS, men used File systems. But due to a few disadvantages in file systems, DBMS was developed. ex: MP3 can't be played in Word.
File systems are - Word, Excel etc.

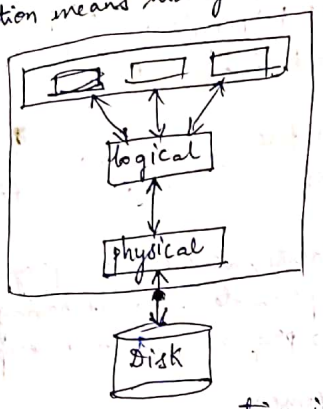
The main problem with file systems is:
* Whenever need arises, we need to generate new application programs.
ex:- To promote all students, we create one program coffee but later if wish to promote students having CGPA > 3, then new program has to developed.

Problems with file systems:-

① Data redundancy & inconsistency.
② Difficulty in accessing the data.
③ Atomicity.
- In DBMS, transaction occurs completely or not at all, but no partial execution.
ex: In ATM if cash is debited from our account but the receiver hasn't got it, then there is some problem in transaction. But with DBMS, the money gets automatically refunded to us in a certain period.
④ Data isolation.
⑤ Integrity.
- Following rules & regulations set up. / following the constraints.
Since file systems is human-controlled, maintaining integrity is very difficult. With DBMS it is possible.
⑥ Concurrent Access.
- In DBMS, different applications run at a very fast rate and appear as if they run parallely.
⑦ Security.
- In file sys., we can't hide data if other person knows how to unhide it. But in DBMS, we have the security.

Level of Abstraction :-

Abstraction means hiding details / complexity.



View level / External level

Conceptual / Logical level

physical level

Fig. Levels of Abstraction in DBMS

* In physical level, we get the details of complex low-level data structure (i.e. at machine level). It tells how the data is actually getting stored. This is called Physical level of Abstraction. It deals with how the data is stored. Hiding
* The logical level, deals with what is the data i.e it defines the data and the relationship between the data. When we hide this, it is called logic level of abstraction.
* In the external level, hiding the unuseful information or for security purposes hiding the data is called external level of abstraction.
A single DBMS can have multiple views.
And we hide the views as per the user roles.

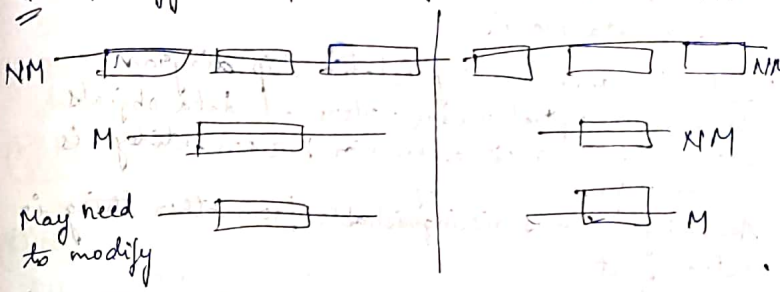
Data Independence :-

In DBMS, any changes / modification done to physical level do not effect application. In this case, the DBMS is said to be physically data independent. Otherwise not.

* Similarly for logically data independence.

* If the changes done at one level don't effect the application, then the DBMS has data independence.
* The data independence is down-up (levels) but not up-down.

ex: Modify - M, No modification - NM (say).



Instance Vs Schema :-

* At a particular instance or time, if you capture your data, then that is called the instance of your DBMS.
* Instances frequently changes.
* Schema is defining the overall design of the data base. It can be called a signature.
* Schema rarely changes.

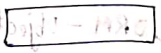
ex:

P#	PName	Product	Price	City
P1	X	Printer	5000	Vizag
P2	Y	Monitor	6000	Rome
P3	Z	Keyboard	1200	Hawai

} Schema

} Instance

Data Models :-



High Level Model

Conceptual Model / Representational Model.

Physical Model / Low-level model.

* The aim of any model is (i) data (ii) relationship (iii) rules

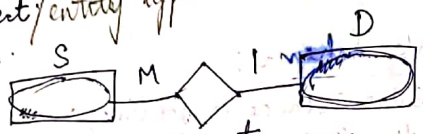
Data Model is a conceptual / logical representation of data objects and their relationship and also deals with the rules to enforce this relationship (or association).

Rules include ---
Enforcement of constraints and other regulatory rules.

Different data models :-

① ER diagram — Entity Relationship diagram.
It is a conceptual model. Here all data objects are entity and their association between entities is relationship.
Anything which is specifically distinguishable from other thing is called object / entity type.

ex: ER diagram



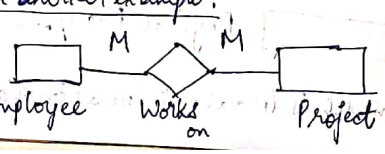
Many to 1 relationship.

i.e many students belong to 1 department (or) 1 department may have many students.

② Relational Model — It is not conceptual model.
Here to represent data and relationship we use tables. These tables are called relations.
* Records belonging to different tables have different record type but records belonging to same table have same record type. (Record types can be numeric, character etc. based on the data in the columns).

→ Diff. data models include ER Model, Relational Model, N/W model, HM, OOM, ORM, Semi-structured data models.
OOM - Object-Oriented Model, ORM - Object Relational Model.
N/W - Network Model
HM - Hierarchial model.

ER diagram another example :



Many to many relationship.

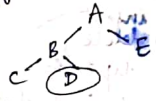
③ OOM — Extended E.R.
Here we can use encapsulation and have methods to access the attributes.

④ Semi-structured data model — Similar to Relational Model.
The only difference is that records belonging to same table may have different record type unlike relational model. So here we use XML.

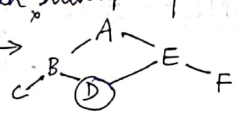
⑤ Hierarchical Model — This model organises the data into a tree-like structure. One part/child has 1 parent only.

⑥ Network Model — Similar to Hierarchical Model.
Only difference is here, a table can have multiple parents.

- HM →



N/W →



⑦ ORM — It combines the feature of Relational model and OOM. Features include encapsulation, inheritance and polymorphism.

Data Base Languages (DBL) :-

- DDL
- DML (query & ML)
- DCL

• DDL — Allows you to define a schema for your table. Suppose EMP (ID, FN, LN, A, E).

ex: Create table EMP (

Field name	Field type	Size
ID	Num(20)	
FN	Varchar	
LN	Varchar	
A	Varchar	
E	Varchar	

)

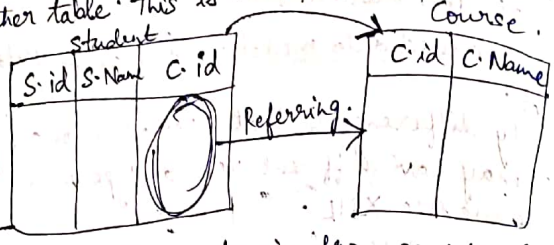
we need to specify constraints also.

Using DDL we can pose constraints

- Domain constraints
- Authorization
- Referential Integrity
- Assertion
- Key constraints

* Referential Integrity Constraints (RIC):
Sometimes we may need to refer the attribute of one table into another table. This is RIC.

ex:



→ Here the C.id is referring from course table. So a student can take up a course only which is available in course table. That is the constraint in student table.

- DML — It is a query and manipulation language.
  - select
  - insert, delete, update

* Select is a command to query & retrieve data.
* Insert, delete, update are commands to manipulate data.

Hence DML allows data retrieval, insertion, deletion and manipulation.

DML — (Non-Declarative) Procedural i.e. PDML
(Non-declarative) Non-Procedural i.e NPDML.

PDML — We specify the Data we need & along with it we specify the procedure.

NPDML — Here we specify only need (what data you need) but not how to get it (no procedure).

PDML — ex: ① Relational Algebra (R.A).
② Tuple Relational Calculus (TRC)
③ Domain Relational Calculus (DRC)

NPDML — ex: SQL (Structured Query Language).
RC (Relational Calculus).

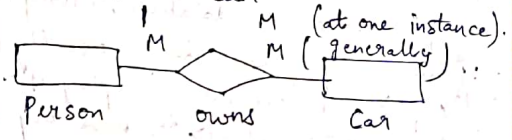
* Query is a statement requesting the retrieval of data.
* Query Lang. & Manipulation Lang. together form the DML.

ER Model :-

* In order to best describe entity we use attribute.
* Relationship is a association between entities.

ex: Entity type: Car, Person
Attribute: Colour, No., Company... Name, id, address...
Relationship: Person owns a car.

ER Diagram:



Attribute :- (Composite Vs Atomic).

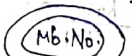
- Atomic / Simple attribute — one which cannot be further divided. eg: age.
- Composite — which can be divided further.
  ex: address.

(Single valued Vs Multi valued).

- Single valued attribute — having only 1 value.
  ex: age.
- Multi valued attribute — having multiple values.
  ex: Mobile number (A person may have 2 numbers).
  Address (permanent & current addresses).

Attribute representation

Multivalued attribute representation



(Stored Vs Derived Attribute).

- Stored — Attributes which we store in data base.
- Derived — Which needn't be stored in database, because we can derive it from stored attributes.

ex: ① If we stored the DOB, then we can calc. age.
Hence DOB - stored, age - derived (DOB) (Age).
② If we store date of joining, then experience of person is known.
Hence Date of joining - stored, experience - derived.

Complex Attribute — Attribute which is Multi-valued and Composite.

ex: Address
- Current Address (Place, City, Pincode, Road...)
- Permanent Address (").

E-R diagram:-

It is a pictorial representational / conceptual design / blue print of actual database, having the requirements.

ex: ★ Every Employees works for exactly 1 dept.
Dept. can have many employees.
A new dept. need not to have any employee.
Create ER diagram for above data.

All nouns - Entity, Verbs - Relationship

Entity type (Schema)

Emp(ID, Name, Address)

Emp 1, Emp 2, Emp 3 → entity (choosing one instance in entity type is entity).

(to be continued) p.t.o.

Key Attribute:-

An attribute that uniquely identifies other attribute is the key attribute. → or a set of attributes,

ex: ①

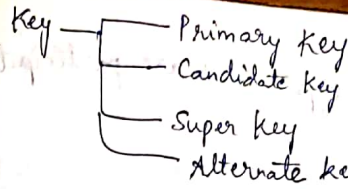
Key: Roll no.	Non-key: Student Name
101	xyz
102	abc
103	xyz

②

Composite Key: Name, City

ID	Name	City	Phone
1	Arya	Noida	1234
2	Bran	Delhi	8743
3	John	Noida	4821
4	Arya	Chennai	7874

Suppose if asked id of Arya from Noida, it is unique.



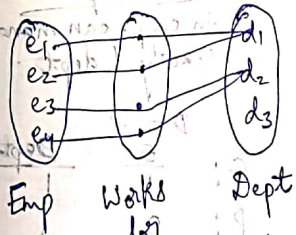
- **Candidate Key** — All attributes or set of attributes can serve as key. The minimal set of attributes which can uniquely identify other attributes is C.K.
  ex: (Aadhar No., PAN no., Lic. No.)
- **Primary Key** — From set of attributes which can serve as keys if we take one for priority it is Pri. key.
- **Alternative Key** — And the rest are Alt. Key.
  ex: (Aadhar No., PAN no., Lic. No.) — P.K; Alt. Key.
- **Super Key** — Not having minimal set of attributes, having some extra attributes in it.
  All candidate keys are super keys but all super keys are not candidate keys.

ex: Roll no. is unique for every student.

Roll No.	S.Name	S.M	S.Add

- (Roll no.) → C.K.
- (Roll no., S.Name) → S.K.
- (Roll no., S.M) → S.K
- (Roll no., S.Add) → S.K.

E-R diagram ex: ★ continued - - -



Rough fig. ↓ state diagram.

Degree(R) = 2.

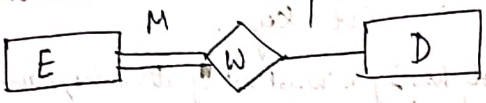
Cardinality	Participation
Emp = 1	Emp = 1
Dept = M	Dept = 0

* Cardinality has to be written on opposite side in E-R diagram.

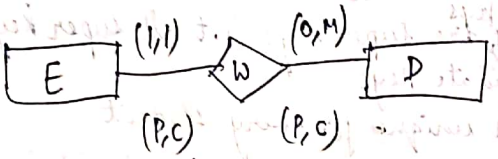
Degree of Relationship :-
Maximum no. of entity type which are participating in the relationship.

Cardanility :-
Maximum no. of ~~entity which~~ relationships in which entity can participate.

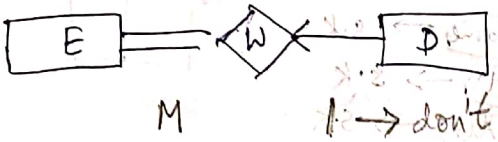
Participation :-
Minimum no. of relationships in which entity can participate.
Total Participation : We use = for T.P (T.P is having neither P nor C as 0). (T.P is having P = 1).



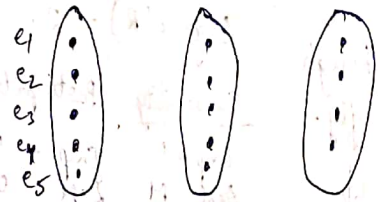
(Cardanility Ratio representation)



(Min Max ER representation)



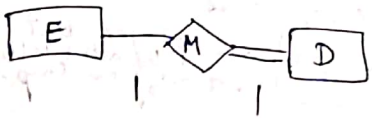
ex 2 : 1-1 Emp. manages Dept :



Assumption.
① An emp. can manage exactly 1 dept.

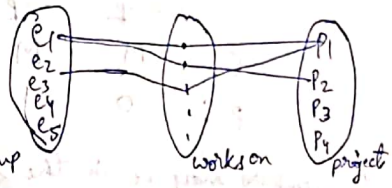
D = 2	E	Dept
C :	1	1
P :	0	1

Assumptions have to be mentioned.



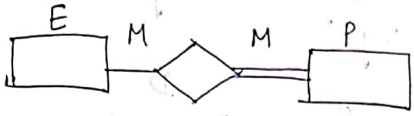
(Cardanility ratio representation)

ex 3 : Emp. works on project.

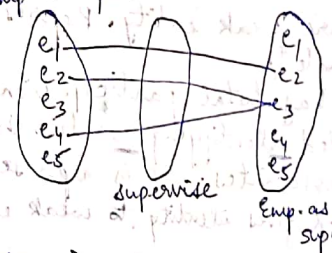


Degree = 2.

	Emp	Project
C :	M	M
P :	1	1



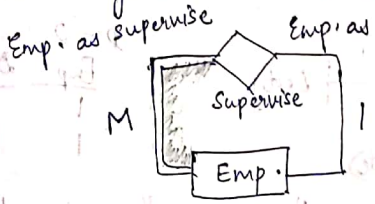
Recursive Relationship :-
ex : Every emp. has to report to his/her (boss) → emp.



Degree = 2.

	Supervise	Supervisor
C :	1	M
P :	0	0

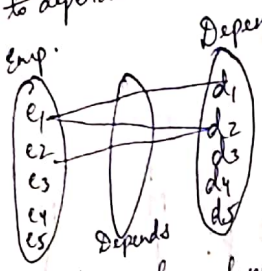
Assumption is there will be atleast 1 who doesn't report to anyone.



Weak Entity :-
Entities without key attributes are weak entities.

Strong Entity :-
Entities with key attributes are strong entities.

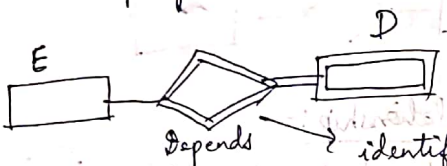
ex: An employee may have dependents.
The company gives id (unique) to employee but to dependents. Hence he needs to borrow the attribute (key) from emp.



Degree = 2

	Emp.	Dep.
C:	M	M
P:	0	1

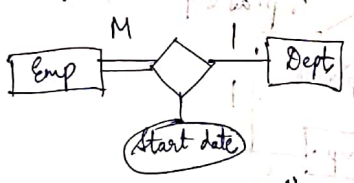
Assumption: Same family members may be in the same company.



* Hence here Dependents are weak entity. Represent it with "double rectangle".
* A weak entity always has "total participation".
* And the relation is called "identifying relationship" with which weak entity associates. It is represented by "double rhombus". I.R gives identity to weak entity.

~~Converting ER~~ Attributes to Relationship:-

ex: 1).



D = 2

	Emp	Dept.
C:	1	M
P:	1	0

Here we have the attribute "start date" exists only bcoz there is relation btw emp & dept. So it is an attribute to relationship.

* We must avoid the usage of Attributes to Relationship, bcoz we may have to create a separate table for that relationship.

* In the above example, - - - (or) M-1
* if the relationship is 1-M, then shift the "A to R" to "many" side.

This is because if we shift to "one" side then the attribute becomes multivalued.

ex: If start date is shifted to dept side then it'll has to define for every emp. in dept separately & becomes multi-valued. Whereas if start date is shifted to ~~was~~ emp side, then simply the start date of that emp. may be specified i.e one more column "start date" has to be created, that's all.

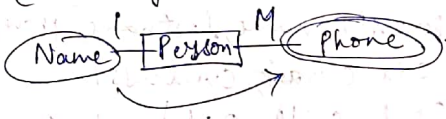
d1 [e1, e2 ...] multi-valued.
d2

* if the relationship is M-M, then shifting "A to R" is not possible and separate table has to be created.
* if it is 1-1, then we can shift to any side ⟷.

Converting E-R diagram to Relational Model (R.M):-

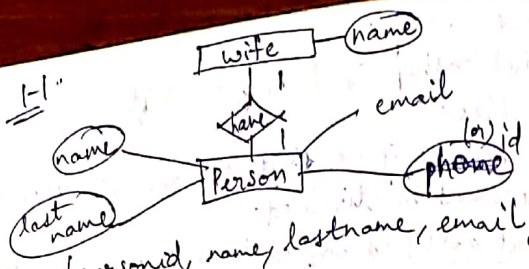
1. For every entity — create a table.
   For every simple attribute — create a column.
2. For every multivalued Attribute — create a separate table.
3. 1-1 (shift attribute to any side to establish relationship)
4. 1-M (" " " many side " " ")
5. M-M (can't shift, should create separate table).

ex: 1-M.



Persons (personid, lastname, email)
phones (phoneid, personid, phone)

ex: 1-1

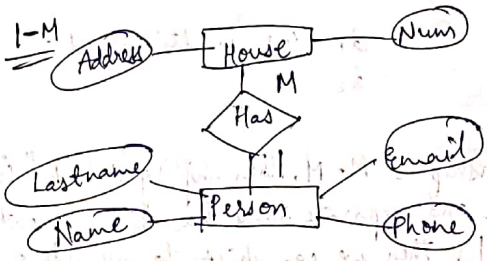


Person (personid, name, lastname, email, wifeid)
Wife (wifeid, name)

(or)

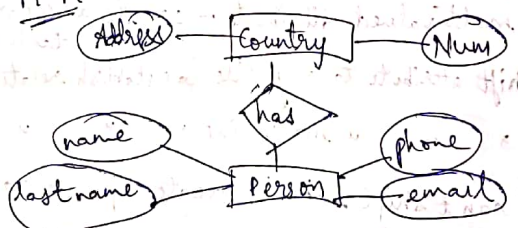
person (personid, name, lastname, email)
wife (wifeid, name, personid)

ex: 1-M



Person (p.id, Name, lastname, phone, email)
House (H.id, House add., p.id)

ex: M-M.

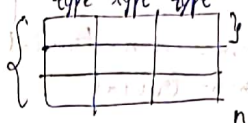


Person (personid, name, lastname, phone, email)
Country (cid, C.Name, C.Address)
PC (Pid, Cid, Rid) → for separate table

Relational Model:-

Here both data & relationship are represented by relation / table.

columns / attributes



n-tuple relation if there are n-tuples.

* A key is an attribute or set of attributes which uniquely identifies other attributes or all tuples of the record / data.

Mathematical definition of Functional Dependency (FD):

If $T_1[X] = T_2[X]$,
then $T_1[Y] = T_2[Y]$.

ex:

T_1	1	X
	2	Y
	3	Y
T_2	1	X

- Fd: $X \rightarrow Y$ where X and Y $\in$ relation R
  - * Y is functionally determinate by X.
  - * Or X uniquely determines Y.
  - * If the value of X replicates, value of Y must also replicates.
  - * X is determinant.
  - * Y is dependent.

Types of Dependencies:-

- Trivial (Obvious):
  - → Dependencies which will always hold in a relation.
  - → $(X \rightarrow Y)$ is trivial if $Y \subseteq X$.
  - → Example $(AB \rightarrow A)$.
- Non trivial (non-obvious):
  - → $(X \rightarrow Y)$ is non trivial if Y is not $\subseteq$ X.
  - → Example $(AB \rightarrow ABC)$.

ex: ① $ABC \rightarrow AB$ is trivial [$\because AB \subseteq ABC$]

② $AB \rightarrow ABC$

then $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow C$ [$C \not\subseteq AB$]

So entire set becomes non-trivial.

$\therefore$ $AB \rightarrow ABC$ is non-trivial.

Properties of FD:-

* Reflexive — if $Y \subseteq X$ then $X \rightarrow Y$.
* Transitive — if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
* Decomposition — if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.
* Union — if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
* Composition — if $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$.
* Pseudo transitivity — if $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$.
* Augmentation — if $X \rightarrow Y$ then $XZ \rightarrow YZ$.

Foreign Key: when we borrow key from other table, then it becomes foreign key.

ex: Person has wife.

Person (Pid, PN, Wife id) → foreign key here.

Wife (Wife id, W.N) → primary key here.

How to find key attribute?

→ Find the closure of attribute.

The closure of a attribute X contains all the attribute which are functionally determine by X.

• FD set for student relation S.

s_id → s_name c_id

c_id → c_name

Closure of s_id is $s\_id^+ = (s\_id, s\_name, c\_id, c\_name)$ [$\because$ from transitivity]

Hence s_id is a key attribute.

$s\_name^+ = \{s\_name\}$.
$c\_id^+ = \{c\_id, c\_name\}$.
$c\_name^+ = \{c\_name\}$.
} Not including all attributes so, they are not c.keys/key.

So s_id is only candidate key and therefore primary key, no alternate key for this relation.
And every super set of s_id is super key for this relation.

ex: 1: R (ABCDEH)

FD (A → BC, CD → E, E → C, D → AEH, ABH → BD, DH → BC).

The no. of candidate keys for relation R is ?

soln: $A^+ = \{A, B, C\}$

$B^+ = \{B\}$

$C^+ = \{C\}$

$D^+ = \{D, A, E, H, C, B\}$ ✓

$E^+ = \{E, C\}$

$H^+ = \{H\}$

~~D is the only~~

$AH^+ = \{A, H, B, C, D, E\}$ ✓

∴ 2 candidate keys: AH & D.

2: Given the foll. relation instance. Which of the foll. functional dependencies are satisfied by the instance?

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

a) $XY \rightarrow Z$ and $Z \rightarrow Y$ ✗

b) $YZ \rightarrow X$ and $Y \rightarrow Z$ ✓

c) $YZ \rightarrow X$ and $X \rightarrow Z$ ✗

d) $XZ \rightarrow Y$ and $Y \rightarrow X$ ✗

3. $R = (A, B, C, D, E, F)$

FD : $(C \to F, E \to A, EC \to D, A \to B)$.

$CD^+ = \{C, D, F\}$

$EC^+ = \{E, C, F, A, D, B\}$ ✓

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

$AC^+ = \{A, C, F, B\}$

∴ EC is candidate key.

4. From the foll. instance of relational schema $R(A, B, C)$ we can conclude that:

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

∴ A functionally determines B and B does not functionally determine C.

5. $R(ABCD\ EF\ GH)$.

FD : $(CH \to G, A \to BC, B \to CFH, E \to A, F \to EG)$.

How many C·K ?

sol^n: $D^+ = \{D\}$, $CD^+ = \{C, D\}$, $GD^+ = \{G, D\}$, $HD^+ = \{H, D\}$.

~~$ABFD^+ = \{A, B, F, C, H, E, G, D\}$~~

$AD^+ = \{A, D, B, C, F, H, E, G\}$ ✓

$BD^+ = \{B, D, C, F, H, E, G, A\}$ ✓

$ED^+ = \{E, D, A, B, C, F, H, G\}$ ✓

$FD^+ = \{F, D, E, G, A, B, C, H\}$ ✓

∴ 4 candidate keys.

*Trick: If any variable/attribute is not on RHS then it has to be a C·K (or) a part of C·K.

6. $R(ABCDEF)$

FD : $(DF \to C, BC \to F, E \to A, ABC \to E)$.

sol^n: $BD^+ = \{B, D\}$.

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

$CBD^+ = \{C, B, D, A, F\}$ missing AE

$EBD^+ = \{E, B, D, A\}$. CF

$FBD^+ = \{F, B, D, C\}$ AE

→ $ABCD^+ = \{A, B, C, D, E, F\}$ ✓

$AFBD^+ = \{A, F, B, D, C, E\}$ ✓

$EFBD^+ = \{E, F, B, D, C, A\}$ ✓

Same

$EBCD^+ = \{E, B, C, D, F, A\}$ ✓

$CEBD^+ = \{C, E, B, D, A, F\}$ ✓

~~$FBD^+ = \{F, B, D, C\}$~~ ✓

∴ Total 4 C·K

7. $R(ABCDE)$

FD : $(A \to BC, CD \to E, B \to D, E \to A)$.

sol^n: $A^+ = \{A, B, C, D, E\}$ ✓ C·K ✓

$E^+ = \{E, A, B, C, D\}$ ✓ C·K ✓

$AB^+ = \{A, B, C, D, E\}$ ✓ S·K

$AC^+ = \{A, C, B, D, E\}$ ✓ S·K

$AD^+ = \{A, D, B, C, E\}$ ✓ S·K

$AE^+ = \{A, E, B, C, D\}$ ✓ S·K

$BC^+ = \{B, C, D, E, A\}$ ✓ C·K ✓

$BE^+ = \{B, E, D, A, C\}$ ✓ S·K

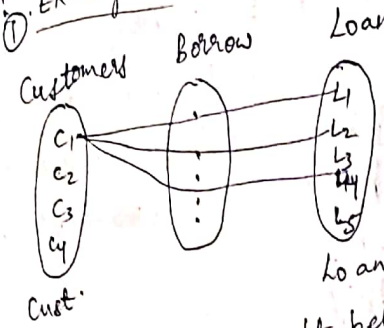
$CD^+ = \{C, D, E, A, B\}$ ✓ C·K ✓

$CE^+ = \{C, E, A, B, D\}$ ✓ S·K

$DE^+ = \{D, E, A, B, C\}$ ✓ S·K

∴ 4 C·K.

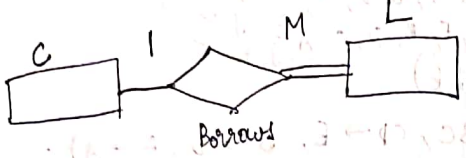
ex: ① ER-diagram.



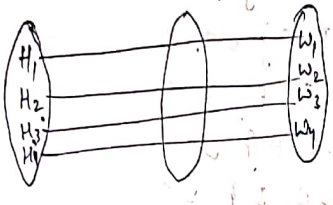
D = 2

	Cust.	Loan
C:	M	1
P:	O	1

Assumption: One loan can't belong to multiple customers.

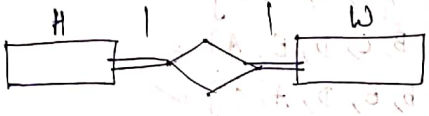


② Husband having wife



D = 2

	H	W
C:	1	1
P:	1	1



8/9/20. SQL. Unit-2.

SQL → Sequence Query Language.

* The SQL language has several parts:
  - → DDL
  - → DML
  - → Integrity constraints
  - → View definition
  - → Transaction control — A transaction is either complete or null process. No interruption. If we interrupt then it rolls back.
  - → Authorization

DDL:-

Allows the specification of:
- The schema for each relation.
- The types of values associated with each attribute/domain
- Integrity constraints.
- Security and Authorization information for each relation.
- The set of indices to be maintained for each relation.

Create table construct:-

ex:

```
create table branch
    (branch_name  char(15),
     branch_city  char(30),
     assets       integer);
```

desc branch
to display table or show the schema

* Drop: Drop will delete the complete table including schema.
* Truncate: Truncate will just delete tuples (rows) but not schema.

Syntax:

```
DROP TABLE table_name;
TRUNCATE TABLE table_name;
```

* Both are DDL commands.

* When we alter the schema of a table, the table must be empty.

ex: alter table table_name add column_name varchar(30)
= alter table table_name drop column column_name
alter table table_name modify column_name char(50)

Syntax:
create table table_name
(Attribute domain_type constraints)

Insert into syntax:
insert into student values(101, 'Rohit', 'Kommadi')
↳ for insertion into all columns.
insert into student (SID, SNAME) values (102, 'Ram')
↳ for insertion into specific columns.

Selecting columns:
select SID, SNAME from student;
↳ for specific column.
select * from table_name
↳ for all columns.

Delete tuples:
delete from student where SID = 101

* SQL is case-insensitive but the values of attributes are case-sensitive. i.e. the name of attributes are case-insensitive and their values are case-sensitive.
* Delete is DML & Truncate is DDL.

Selection:
select distinct (sname) from student

Select distinct will show repeated ones only once.
↳ (i.e removes duplicates)

* The 'select' clause can contain arithmetic expressions

ex: select loan_number, amount * 100
= from loan

where clause:
select loan_number
from loan
where branch_name = 'Perryridge' and amount > 10

We may use 'and', or 'or' or 'not'.

from clause:
• ex: A = {1, 2, 3}, B = {a, b}.
= Cartesian product A * B = {<1,a>, <1,b>, <2,a>, <2,b>, <3,a>, <3,b>}.

• ex:

Std.

S.No	S.Name
1	S1
2	S2

Course.

C.No.	C.Name
101	C1
102	C2

from Std, Course

⇒

S.No	S.Name	C.No.	C.Name
1	S1	101	C1
1	S1	102	C2
2	S2	101	C1
2	S2	102	C2

• Find the Cartesian product borrower X loan.
select *
from borrower, loan.

* select is a vertical filter clause.
* where is a horizontal filter clause

SQL clauses — Select, Where, From, Insert.

Rename Operation:

old_name as new_name

ex: Find the name, loan number and loan amount of all customers; rename the column name loan_number as loan_id.

Sol^n:
Given
C customer (customer_name, customer_city)
L loan (loan_number, branch_name, amount)
B borrower (customer_name, loan_number).

select customer_name, borrower.loan_number as loan_id, amount
From C, L, B
where C.customer_name = B.customer_name
and L.loan_number = B.loan_number

② Find customer names and their loan numbers and amount for all customers having a loan at some branch.

select customer_name, B.loan_number, L.amount
from borrower as B, loan as L
where B.loan_number = L.loan_number

String Operations:-

The operator 'like' uses patterns that are described using two special characters.

- percent (%) : The % character matches any substring.
- underscore (_) : The _ character matches any character.

ex: ① Find the names of all customers whose street includes the substring "Main".

select customer_name
from customer
where customer_street like '% Main %'

Order by clause:

We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.

ex: order by customer_name desc

Aggregate functions:

avg, min, max, sum, count.

These fn.s operate on the multiset of values of column.

ex: ① Find the average account balance at the Perryridge branch.

select avg(balance)
from account
where branch_name = 'Perryridge'

② Find no. of tuples in the 'customer' relation.

select count(*)
from customer

③ Find the no. of depositors for each branch.

select branch_name, count (distinct customer_name)
from depositor, account
where depositor.account_number = account.account_number
group by branch_name

Having clause:

Having is same as where but is applicable mainly on groups. Where is for the total database.

ex: Find the names of all branches where the average account balance is more than $1200.

```
select branch_name, avg (balance)
    from account
    group by branch_name
    having avg (balance) > 1200
```

Set Operations:-

Union, intersect and except are the set operations.

(A∪B) (A∩B) (A−B)

* (A∪B) − (A−B) − (B−A) = A∩B
* Each of above operations automatically eliminates duplicates, to retain all duplicates use "union all, intersect all, and except all".

To perform set operations (like union, ...) on relations R_1, R_2 (say), then they must have same "arity".

Arity:

① No. of attributes in R_1 & R_2 must be same.
② The domain of corresponding attributes in R_1 & R_2 must be same.

ex: ① Find all customers who have a loan, an account, or both:

```
(select customer_name from depositor)
union
(select customer_name from borrower)
```

② Find all customers who have both a loan and an account.

```
(select customer_name from depositor)
intersect
(select customer_name from borrower)
```

③ Find all customers who have an account but no loan.

(or)

Find all customers who have only account number.

```
(select customer_name from depositor)
except
(select customer_name from borrower)
```

Nested Subqueries:-

A sub-query is a select-from-where expression that is nested within another query.

* A common use of subqueries is to perform tests for set membership, set comparisons and set cardinality.

"In" Construct:

ex: Find all customers who have both an account and a loan at the bank.

```
select distinct customer_name
    from borrower
    where customer_name in (select customer_name
                            from depositor)
```

ex: Find all customers who have a loan at the bank but do not have an account at the bank.

```
select distinct customer_name
    from borrower
    where customer_name not in (select customer_name
                                from depositor)
```

"Some" Construct:-

- Find all branches that have greater assests than some branch located in Brooklyn.

Given,

branch (branch_name, branch_city, assets).

```
select branch_name
from branch
where assets > some
    (select assets
     from branch
     where branch_city = 'Brooklyn')
```

"All" Construct:-

- Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
from branch.
where assets > all
    (select assets
     from branch
     where branch_city = 'Brooklyn').
```

Modification of the Database – Deletion:-

ex: ① Delete all account tuples at the Perryridge branch.

```
delete from account
where branch_name = 'Perryridge';
```

② Delete all accounts at every branch located in the city 'Needham'.

```
delete from account
where branch_name in (select branch_name
                      from branch
                      where branch_city = 'Needham');
```

③ Delete the record of all accounts with balance below the average at the bank.

```
delete from account
having where balance < ( select avg (balance)
                         from account )
```

Modification – Insertion:-

ex: ① Add a new tuple to account

```
insert into account
values ('A-9732', 'Perryridge', 1200);
```

Modification – Updates:-

ex: Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

- Write two update statements:

```
update account
set balance = balance * 1.06
where balance > 10000;

update account
set balance = balance * 1.05
where balance ≤ 10000;
```

- Using case statement:

```
update account
set balance = case
              when balance <= 10000 then balance * 1.05
              else balance * 1.06
              end;
```

Joined Relations:-

These are used to merge two tables.

ex: Find the customer names who are borrowers.

```
select customer_name from
from customer, borrower
where customer.customer_name = borrower.
                               customer_name
```

↳ joined relation.

Inner join can work on >, <, = ... It equates the same tuples in two tables.

Outer join: Left, Right, Full.

ex: Relation student

Sid	Sname	did
1	X	3
2	Y	5

Relation dept.

did	dname
3	A
4	B

Left: (L)

1	X	3	3	A
2	Y	5	Null	Null

Right: (R)

~~Null~~	~~Null~~	~~Nu~~		
1	X	3	3	A
Null	Null	Null	4	B

Full: (F)

1	X	3	3	A
2	Y	5	Null	Null
Null	Null	Null	4	B

In L, the total no. of tuples <.= no. of tuples in left table

Inner join: 1 X 3 3 A

* Natural join is inner join. Same like inner join but natural join always works on '='.

ex: Find all customers who have either an account or a loan (but not both) at the bank.

```
select customer_name
  from (depositor
natural full outer join
    borrower)
  where account_number is null
    or loan_number is null.
```

Derived relations:-

A subquery expression can be used in from clause.

ex: Find the average account balance of those branches where the average account balance is greater than $1200.

```
select branch-name, avg-balance
from (select branch_name, avg (balance)
      from account
      group by branch_name)
    as branch_avg (branch_name, avg_balance)
  where avg_balance > 1200
```

Here we are deriving the table name from 'from' clause and naming it as branch_avg, giving the attributes branch_name and avg_balance.

Exists:-

It checks for empty relations.

It returns true if the sub query returns atleast 1 record. Otherwise it returns false.

* Ly non-exists returns true if the sub-query returns 0 records or empty results.
* If we have any reference of outer query inside the inner query, then it is Corelated query.

ex: Find all customers who have an account at all branches located in Brooklyn.

```
select distinct S.customer_name
  from depositor as S
  where not exists (
    (select branch_name
     from branch
     where branch_city = 'Brooklyn')
    except
    (select R.branch_name
     from depositor as T, account as R
     where T.account_number = R.account_number
       and S.customer_name = T.customer_name))
```

Unique:-

This is also for Corelated query.

* If the result of inner query returns or has duplicates then 'unique' returns false otherwise it returns true.

ex: Find all customers who have atmost one account at the Perryridge branch.

```
select T.customer_name
from depositor as T
where unique (
    select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name
    and R.account_number = account.account_number
    and account.branch_name = 'Perryridge').
```

View Definition:-

A relation that is not of the conceptual model but is made visible to user as a "virtual relation" is called a view. Views are temporary tables. We don't store them.

ex: Emp:

Emp ID	EN	ES

* If we want to hide ES and create a virtual table.

```
create view Empp as (select EmpID, EN from Emp).
```

↳ new view table name

Empp:

Emp ID	EN

Null:-

* Key attribute can't be null and has to be unique.
* Any arithmetic operation with null results in null.
* All Aggregate functions (like sum, ...) ignore the null, except count (*).

ex: sum(A)

A
2
3
Null

⇒ results 5.

ex: ① $C_1 \wedge C_2$

C_1 AND Unknown

Only if C_1 is F ⇓ False | otherwise unknown.

② C_1 OR C_2

Unknown OR C_2

Only if C_2 is T ⇓ True | otherwise unknown.

* Any comparison with null returns unknown.
  ex: 5 < null or null <> null or null = null.
* (not unknown) = unknown.
* customer_name = NULL ⇓ unknown | customer_name is null ⇓ true.
* Result of where clause is false if it evaluates to unknown.

View Expansion:-

The virtual relation is not pre-computed and stored, & hence are always updated.

View from another view:

```
Create view V1 as (select XYZ
                   from V2).
```

Views may --- depend directly on another view.

depend on indirectly

recursive (i.e. self-dependent).

* Materialized view have physical existence. These may become outdated since these views are pre-computed and stored. → maintenence.

Hence view updated may be --

- Immediate - whenever we update base table immediately (only for M.V) update the view.
- Periodically - keep on updating after a certain time period.
- Lazily - doesn't get updated when base table is updated but gets updated when there is a need or someone needs access.

For a real-time system immediate updation should only be applied.

View updation standard term is — View Maintenence.

→ Since M.V is physical one can update the view and hence we need to update the base table also (because of Referential integrity constraints)

To avoid this →

① → simply reject updating the view. and allowing by asking him/her to update base table first & then the view.

② → even then if customer wants to update then just allow and bear the complexity. The DBA will have to deal complexity and update the base table.

Relational Algebra :-

* It presents the basic set of operations for relational model.
* It is a procedural language.

Unary Relational Operations

- SELECT (σ)
- PROJECT (π)
- RENAME (ρ)

Binary Relational Operations

- JOIN ($\bowtie$)
- DIVISION

SET THEORY:

- Union ($\cup$)
- Difference ($-$)
- Intersection ($\cap$)
- Cartesian Product ($\times$)

Select Operation (σ) :- (horizontal)

Notation - $\sigma_p(r)$

- $\sigma_{subject = "database"}(Books)$

output - Selects tuples from books where subject is 'database'.

Project Operation (π) :- (vertical)

- Notation - $\pi_{A1, A2, An}(r)$

where A1, A2, An are attribute names of relation r.

Rename (ρ) :-

- $\rho(C(1 \to sid1, 5 \to sid2), S1 \times R1)$

Syntax: $\rho(R(F), E)$

* Cardinality = $m * n$
* Degree = $p + n$

→ To allow updation of M.V then 4 rules have to be followed:

① The fromclause has only one database relation.
② The select clause contains only attribute names of relation, & does not have any expressions, aggregates.
③ Any attribute not listed in select clause can be set to null.
④ The query does not have a group by or having clause.

→ Unit - 3.

Normalisation :- (schema refinement) complex

It is simply decomposing a relation into less complex relations.

* We decompose because the larger table/relation may contain Data Redundancy which leads to Data Inconsistency and Modification Anamolies.

Insertion Anamolies, Deletion Anamolies, Updation Anamolies

Conditions during normalisation:
→ Lossless decomposition (must) compulsory
→ Dependency preserving (should) not compulsory

- While decomposing,
  → Define foreign key in old table.
  → Refer it new table as primary table.
- Redundancy comes because of functional dependencies, & because of multi-valued F.D.

Ex:
1 NF ← C1
2 NF ← C2
3NF ← C3
BCNF ← C4

* If a relation is in any NF it means it satisfies the condition imposed on it.
* NF's are cummulative i.e. if a relation is in 3 NF, then it satisfies all conditions of 1NF & 2NF & 3NF also.
* To be in NF, a relation must satisfy a condition.

- 1NF = C1
- 2NF = 1NF + C2
- 3NF = 2NF + C3
- BCNF = 3NF + C4

where C1 : Domain of all attributes must be atomic.
C2 : No Partial FD.
C3 : No transitive FD.
C4 : Every attribute must depend on super key only.

C1 implies => No repetitive groups, no multivalued attributes, ~~and no composite attributes.~~

* All components of candidate (composite) key are called prime attributes.
  Ex: if {AC} is C.K then A, C are prime attributes and rest are called non-prime attributes.
* If Non-prime depends on Prime i.e P ← N.P, then it's Partial F.D.
* Only if there is composite key, then prime & non-prime attributes arise.
  ∴ 1NF, only multivalued are not allowed (Composite attributes allowed).

Ex: Given a table R(ABCDEF). AC → D, AB → E, CD → F, CDF → B, BF → D.
Sol: $\{AC\}^+ = \{A, B, C, D, E, F\}$. ✓ C.K.

$\{ADC\}^+ = \{A, C, D, F, B, E\}$ ✓ C.K.
P : ABCD
NP : EF

While checking for 2 NF, ABD → E is not partially dependent because ABD as a whole of C.K even though AB is part of C.K.
i.e ABD ∉ {(ABC, ADC)}.
But CD → F is a partial key (Also CD ⊂ ADC).
↳ So this creating problem, so create table.

∴ R_1(ABCDE) , R_2(CDF).

Q: R($F_1 F_2 F_3 F_4 F_5$).
$F_1 \to F_3$, $F_2 \to F_4$, $(F_1, F_2) \to F_5$.
Sol: $\{F_1, F_2\}^+ = \{F_1, F_2, F_3, F_4, F_5\}$. ✓ C.K.

$F_1 F_3$ | $F_2 F_4$ | $F_1 F_2 F_5$.

Relational Algebra :-

1. It is Procedural language.
2. It takes instances from one relation and produces another relation.
3. It is based on set theory (removes duplicates).
4. It uses certain operations to answer a query.
5. In R.A operations can only be unary / binary.

Ex: Give BID where BC > 5.

$\Pi_{BID}(Book(\sigma_{BC>5}))$

Rename (ρ):-

$\rho_{newtablename(NA_1, NA_2)}(R)$ — R(A, A2) → newtablename(NA1, NA2)

* $\rho_{(NA_2)}(R)$ → will rename only attribute.

ex: All BID where BC > 5. Rename BookID.

$$\rho_{\text{BookID}(\text{ID})}\left(\pi_{BID}\left(\sigma_{BC>5}(Book)\right)\right)$$

* Always $t \leq t_1 \times t_2$
  - t: no. of tuples in join
  - $t_1 \times t_2$: no. of tuples in cartesian product

* Join is a selection/filter over cartesian product.

* Inner Join
  - theta Join (θ)
  - Equi Join ($=$)
    - Natural Join

* Outer Join
  - Left OJ.
  - Right OJ
  - Full OJ.

* $\bowtie_{R_1.C_2 > R_2.C_3}$ ⇓ Theta Join $(R_1 \bowtie_\theta R_2)$

  $\bowtie_{R_1.C_2 = R_2.C_3}$ ⇓ Equi Join $(R_1 \bowtie_= R_2)$.

* $R_1 \bowtie_{R_1.C_1 > R_2.C_2} R_2$ → Theta Join

  $R_1 \bowtie_{R_1.SID = R_2.BID} R_2$ → Equi Join

  $R_1 \bowtie_{R1.SID = R_2.SID} R_2$ → Equi Join & Natural Join

Relational Calculus:-
- DRC (Domain)
- TRC (Tuple) ⇒ Tuple Domain Calculus.

* R.C is non-procedural.
* Query form is $\{T \mid F(T)\}$ → TRC:-
  - T: tuple variable
  - F(T): Formula

T is the set of tuples (variables) for the which formula F(T) evaluates to true.

① T is a variable (No projection allowed with T).
② Many T.V are allowed in the formula where T is the only free variable.

* A free variable doesn't include existence quantifiers like $\exists, \forall$.

③ F(T)
↳ Atomic Formula.
① $T \in$ Relation
ex: $S \in$ Sailor
② T.a op S.b ($<, >, <=, >=, =, \neq$).
ex: S.Rating > R.Rating
③ T.a op Constant
ex: S.Rating > 7.

↳ if F_1 & F_2 are formulas, then any logical combination of them is also a formula. ($F_1 \wedge F_2$, $F_1 \vee F_2$, --).

↳ if F(T) is a formula then $\forall T \in R(F(T))$

$\exists T \in R(F(T))$.

Domain variables: E N A R → ③
SI, SN, A, R → ①

Sailor(Sid, SName, age, rating)
Boat(Bid, BN, Color)
Reserve(Bid, Sid, Day).

Bid, Sid, D ③

Queries:
① Find a sailor with rating > 7
$\{S \mid S \in Sailor \wedge S.rating > 7\}$.
② Find a sailor whose age < 30.
$\{S \mid S \in Sailor \wedge S.age < 30\}$.
③ Find the name of the sailor with rating > 7
$\{S \mid \exists S_1 \in Sailor (S_1.rating > 7 \wedge S_1.Name = S.Name)\}$.

④ Find age & Name of sailors having rating > 7.

$\{ S \mid \exists S_1 \in Sailor (S_1.rating > 7 \wedge S_1.Name = S.Name \wedge S_1.age = S.age) \}$.

⑤ Find sailors rating > 7 and who have reserved Bid = 103.

$\{ S \mid S \in Sailor \wedge S.rating > 7 \wedge (\exists R \in Reserve (R.Sid = S.Sid \wedge R.bid = 103)) \}$

⑥ Find sailor name who reserved red boat.

$\{ S \mid \exists S_1 \in Sailor (S_1.Name = S.Name \wedge (\exists R \in Reserve (R.Sid = S.Sid \wedge (\exists B \in Boat (B.Bid = R.Bid \wedge B.color = red))))) \}$

(or) → Simpler way

$\{ S \mid \exists S_1, \exists R, \exists B (S_1.Sid = R.Sid \wedge R.Bid = B.Bid \wedge B.Color = Red \wedge S_1.Name = S.Name) \}$

⑦ Find sailors who reserved red or green boat.

$\{ S \mid S \in Sailor \wedge \exists R \in Reserve (R.Sid = S.Sid \wedge (\exists B \in Boat (B.Bid = R.Bid \wedge (B.color = red \vee B.color = green)))) \}$.

⑧ Find sailors who reserved red and green boats.

$\{ S \mid S \in Sailor, \exists R \in Reserve, \exists B \in Boat (S.Sid = R.Sid \wedge R.Bid = B.Bid \wedge B.Color = red) \wedge (S.Sid = R.Sid \wedge R.Bid = B.Bid \wedge B.Color = Green) \}$

⑨ Find sailors who have reserved 2 different boats.

$\{ S \mid S \in Sailor, \exists R_1 \in Reserve, \exists R_2 \in Reserve (R_1.Sid = S.Sid \wedge R_2.Sid = S.Sid \wedge R_1.Bid \neq R_2.Bid) \}$.

DRC :-

Here the variable is domain of attributes belonging to a relation.

Query Format:

$\{ \langle x_1, x_2, \cdots, x_n \rangle \mid F(\langle x_1, x_2, \cdots, x_n \rangle) \}$

* Conditions are same as TRC.

Queries:

① Find sailors having rating > 7.

$\{ \langle SI, SN, R, A \rangle \mid \langle SI, SN, R, A \rangle \in Sailor \wedge R > 7 \}$.

② Find sid of sailors whose rating > 7.

$\{ \langle SI \rangle \mid \exists SN, R, A (\langle SI, SN, R, A \rangle \in Sailor \wedge R > 7) \}$.

③ Find the name of sailor who reserved boat of id 103.

$\{ N \mid \exists ITA (\langle INTA \rangle \in Sailor \wedge \exists I_R, B_R, D (\langle I_R, B_R, D \rangle \in Reserve \wedge I_R = I \wedge B_R = 103)) \}$.

④ Find the names of sailors who have reserved a red boat.

Sailor (I, N, T, A), Boat (B, BN, C), Reserve (I_R, B_R, D)

$\{ N \mid \exists ITA (\langle INTA \rangle \in Sailor \wedge \exists I_R B_R D (\langle I_R B_R D \rangle \in Reserve \wedge I_R = I \wedge \exists BBNC (\langle BBNC \rangle \in Boat \wedge B_R = B \wedge C = 'red'))) \}$.

Division Operator :-

Consider $R_1(X)$, $R_2(Y)$ where X & Y are set of attributes.

$R(Z) = R_1(X) \div R_2(Y)$

$X = Z \cup Y \leftarrow$ ① $Z = X - Y$

② T_R is a tuple from R_1 iff it is associated with all the tuples of R_2.

Mathematically

$r \div s = \{ t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$.

ex:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	6
ε	1
β	2

r

$r \div s$:

B
1
2
3

s

A
α
β

division example in TRC:

① Find sailor who reserved all boats.

$\{ S \mid S \in Sailor \wedge$
$\forall B \in Boat (\exists R \in Reserve$
$(S.sid = R.sid \wedge B.Bid = R.Bid)) \}$

- Expressive Power (Theorem due to Codd):
- Every query that can be expressed in relational algebra can be expressed as a safe query in DRC/TRC; the converse is also true.
- SQL has implementation feature but R.A and R.C do not have so.

Canonical Cover of FD set / Minimal set of FDs / Irreducible set of FDs :-

FD set → F ⇒ F^+ → Canonical FD set.

F may have ① redundant FDs.
② Extra generous attributes.

ex:① R(WXYZ)

$X \to W$
$WZ \to XY$
$Y = WXZ$

Step 1
Decomposition rule

$X \to W$ ✓
$WZ \to X$ ✗
$WZ \to Y$ ✓
$Y \to W$ ✗
$Y \to X$ ✓
$Y \to Z$ ✓

Step 2.
Find essentials

$X \to W$
$WZ \to Y$
$Y \to X$
$Y \to Z$

Canonical cover

② R(ABC)

$A \to BC$
$B \to C$
$A \to B$
$AB \to C$

Step 1.

$A \to B$ ✗
$A \to C$ ✗
$B \to C$
$A \to B$
$AB \to C$

$A \to C$

Using shortcut.

$A \to B$
$B \to C$

↳ Canonical cover

Using process.

$A \to B$ ✗
$A \to C$ ✗
$B \to C$ ✓
$A \to B$ ✓
$AB \to C$ ✗

$A^+ = \{ACB\}$
$A^+ = \{ABC\}$.
$B^+ = \{BC\}$
$B^+ = \{B\}$
$A^+ = \{A\}$
$AB^+ = \{ABC\}$
$AB^+ = \{ABC\}$

$A \to B$
$B \to C$

↳ Canonical cover.

* Canonical cover is not unique.

③

$C \to B$
$CB \to AC$
$CAE \to FB$
$D \to E$
$CA \to B$

$C \to B$ ✗
$CB \to A$ ✗
$CB \to C$ ✗
$CAE \to F$ ✓
$CAE \to B$
$D \to E$ ✓
$CA \to B$

$C^+ = \{CBAC\}$
$C^+ = \{C\}$
$CB^+ = \{CBA\}$
$CB^+ = \{CB\}$
$CB^+ = \{CBA\}$
$CAE^+ = \{CAEBF\}$
$CAE^+ = \{CAEB\}$

$C \to B$
$CB \to A$
$CAE \to F$
$D \to E$

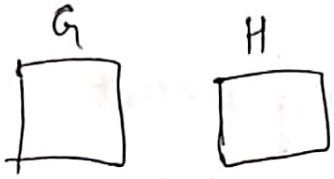
Now $CB^+ = \{CBA\}$.

$C^+ = \{CB\}$. } Hence B is extrageneous

$B^+ = \{B\}$.

$C \to B$
$C \to A$
$CE \to F$
$D \to E$

} $C \to AB$, $CE \to F$, $D \to E$ — Canonical cover.

* 

If both G and H cover each other then they are said to be equivalent.

ex:

F = {
$A \to B$
$AB \to C$
$D \to AC$
$D \to E$
}

G = {
$A \to BC$
$D \to AB$
}.

F

~~$A \to B$~~
~~$AB \to C$~~
~~$D \to A$~~
~~$D \to C$~~
~~$D \to E$~~

G

~~$A \to B$~~
~~$A \to C$~~
~~$D \to A$~~
~~$D \to B$~~

$\therefore F^+$

$A \to B$
$A \to C$
$D \to A$
$D \to E$

G^+

$A \to B$
$A \to C$
$D \to A$

Hence F covers G but G does not cover F.