Unit III part I: Relational Data Model

Relational model terminology

Domain:

In data management and database analysis, a data domain refers to all the values which a data element/ an attribute may contain.

Attribute	Domain
name	Person Name
Job	Job Name
course	course Name

A-Z / a-z

Teaching, accad

Bba, bca, bcom

Figure 1: domain example

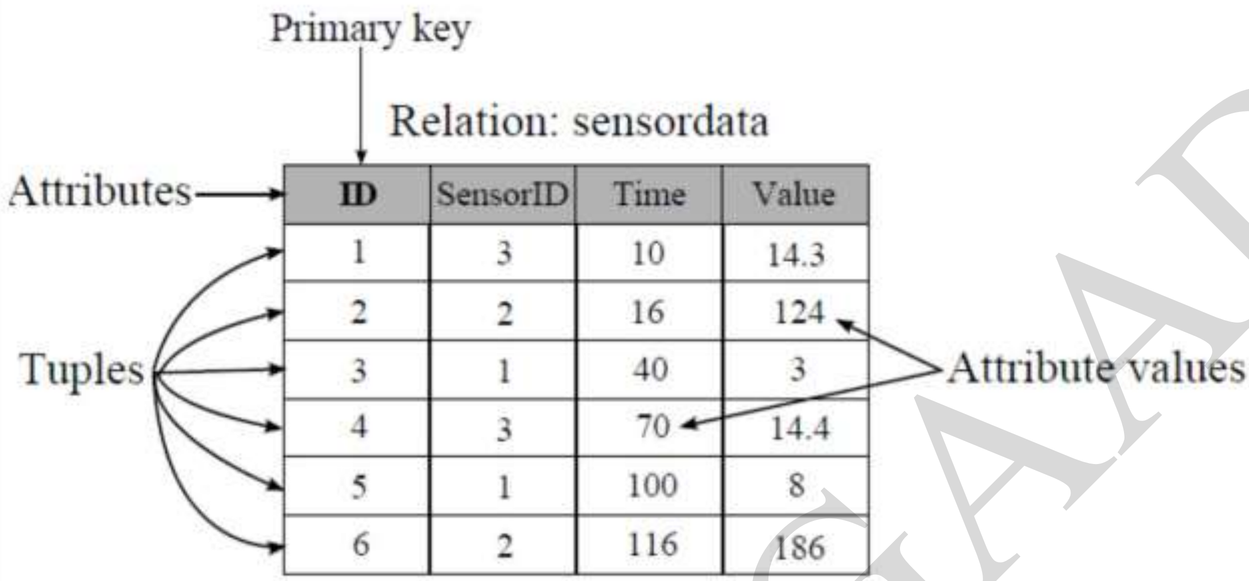
Attributes:

Attribute: Attribute names (or simply attributes) are properties of entity types. An attribute is a property or characteristic of an entity type that is of interest to an organization. It defines property of an entity. Some attributes of common entity types include the following: Example:

STUDENT = {Student ID, SSN, Name, Address, Phone, Email, DOB} ORDER = {Order ID, Date of Order, Amount of Order} ACCOUNT = {Account Number, Account Type, Date Opened, Balance} CITY = {City Name, State, Population}

Tuples

A single row of a table, which contains a single record for that relation, is called a tuple.



ID	SensorID	Time	Value
1	3	10	14.3
2	2	16	124
3	1	40	3
4	3	70	14.4
5	1	100	8
6	2	116	186

<u>Figure 2: tuples</u>

Relations

A database relation is a predefined row/column format for storing information in a relational database. Relations are equivalent to tables.
Also Known As: Table

Characteristics of relations

• No Duplicate Tuples – A relation cannot contain two or more tuples which have the same values for all the attributes. i.e., In any relation, every row is unique.

• Tuples are unordered – The order of rows in a relation is immaterial.

• Attributes are unordered – The order of columns in a relation is immaterial.

• Attribute Values are Atomic – Each tuple contains exactly one value for each attribute.

Constraints:

What are Database Constraints in DBMS ??

Database constraints are restrictions on the contents of the database or on database operations. It is a condition specified on a database schema that restricts the data to be inserted in an instance of the database.

Need of Constraints:

Constraints in the database provide a way to guarantee that :

- the values of individual columns are valid.
- in a table, rows have a valid primary key or unique key values.

- in a dependent table, rows have valid foreign key values that reference rows in a parent table.

Relational Integrity Constraints.

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**.

Constraints allow us to restrict the domain of an attribute. For instance,a constraint can restrict a given integer attribute to values between 1 and 10.

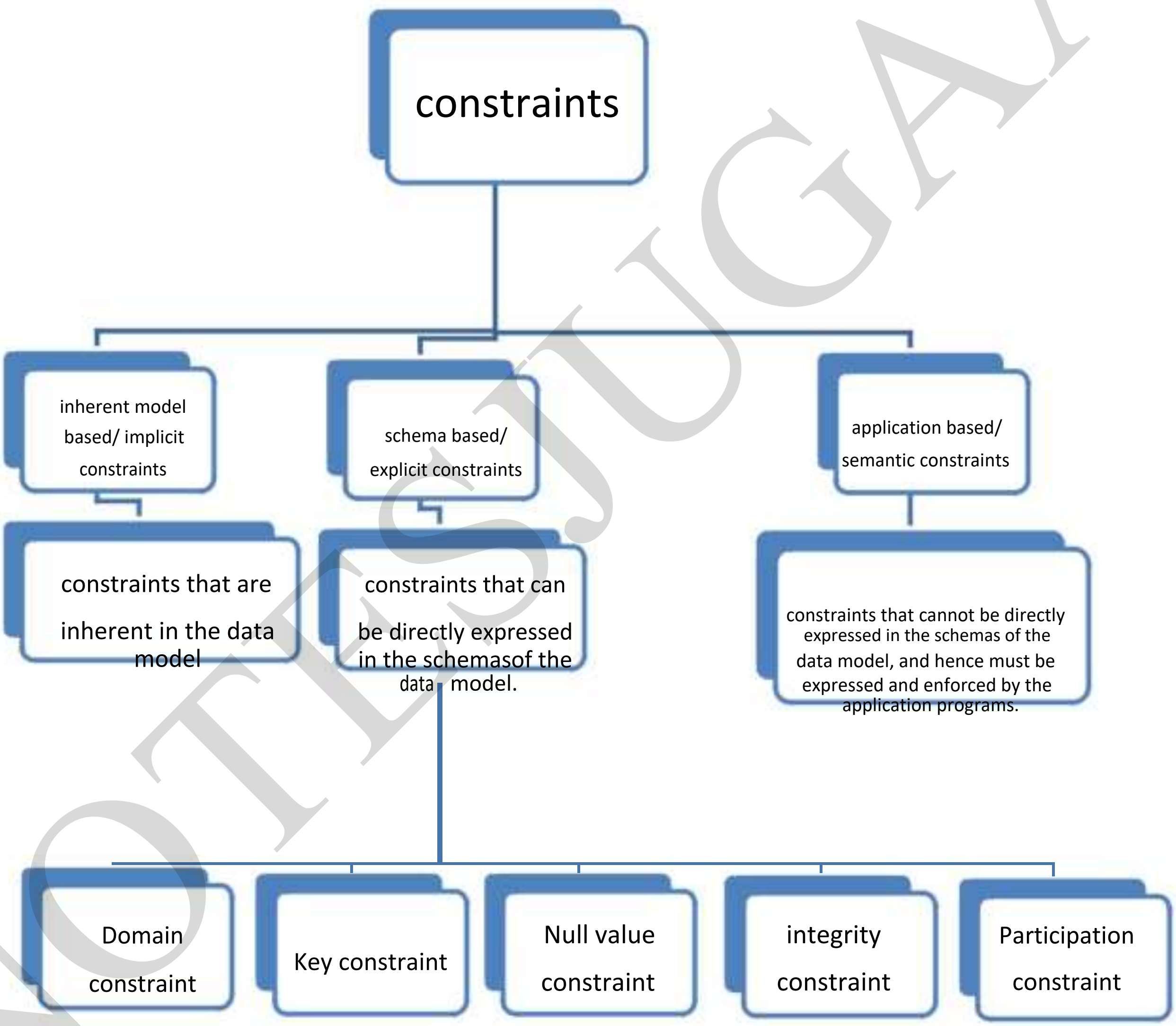


Figure 3: types of constraints

1. inherent model based/ implicit constraints

For example, duplicate tuples are not allowed in a relation

2. schema based/ explicit constraints

For example, films have only one director

3. application based/ semantic constraints

For example, this year's salary increase can be no more than last year's

There are 5 types of schema based constraints:

. Key constraint
. Null value constraint
. Domain constraint
. Entity integrity constraint
. Referential integrity constraint

Integrity Constraints

. Participation constraints
 - total participation
 - Partial participation

Key constraint

Keys are attributes or sets of attributes that uniquely identify an entity within its entity set. An Entity set E can have multiple keys out of which one key will be designated as the primary key. **<u>Primary Key must have unique and not null values in the relational table</u>**.

Example of Key Constraints in a simple relational table –

SID	Name	Class (semester)	Age
8001	Ankit	1^{st}	19
8002	Srishti	1^{st}	18
8003	Somvir	4^{th}	22
8004	Sourabh	6^{th}	45
8002	Tony	5^{th}	23

Not allowed as Primary Key Values must be unique

<u>Figure 4: key constraint</u>

Null value constraint

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, by default it takes NULL value. By specifying not NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Domain constraint

Domain Constraints specifies that what set of values an attribute can take. Value of each attribute X must be an atomic value from the domain of X.
The data type associated with domains include integer, character, string, date, time, currency etc. An attribute value must be available in the corresponding domain. Consider the example below –

SID	Name	Class (semester)	Age
8001	Ankit	1st	19
8002	Srishti	1st	18
8003	Somvir	4th	22
8004	Sourabh	6th	A

Not Allowed. Because Age is an Integer Attribute.

Figure 5: domain constraint

Most imp.

Integrity Constraints:

It refers to the accuracy and correctness of data in the database.

All DBMS' must have some form of ICs to prevent invalid data from being entered.

- Domain constraints specify the set of values which may be used for each field.
- Other constraints, such as key or tuple, may limit which values from the domain can be used for a given field in a given instance.
- Key constraints require that each set of fields in the key be unique for each entry.

Types of Integrity Constraints:

Entity integrity (on primary key)

- No component of the Primary Key is allowed to accept nulls.

Referential integrity (on foreign key)

Master table: table having the original primary key which is being used as a foreign key in another table
Detail table- table having the primary key of some other table.

Referential integrity (on foreign key)

Refers to foreign key in which we:

- Cannot enter / insert a value in foreign key unless it exists in the master table having primary key.
- Can delete any value from detail table having foreign key.
- Cannot delete any value from master table if it exists in detail table.

For example, the titles table in the books database has a link to the publishers table because there is a logical relationship between books and publishers. The pub_id column in the titles table matches the primary key column of the publishers table. The pub_id column in the titles table is the foreign key to the publishers table.

Master table: PUBLISHERS
Detail table: TITLES

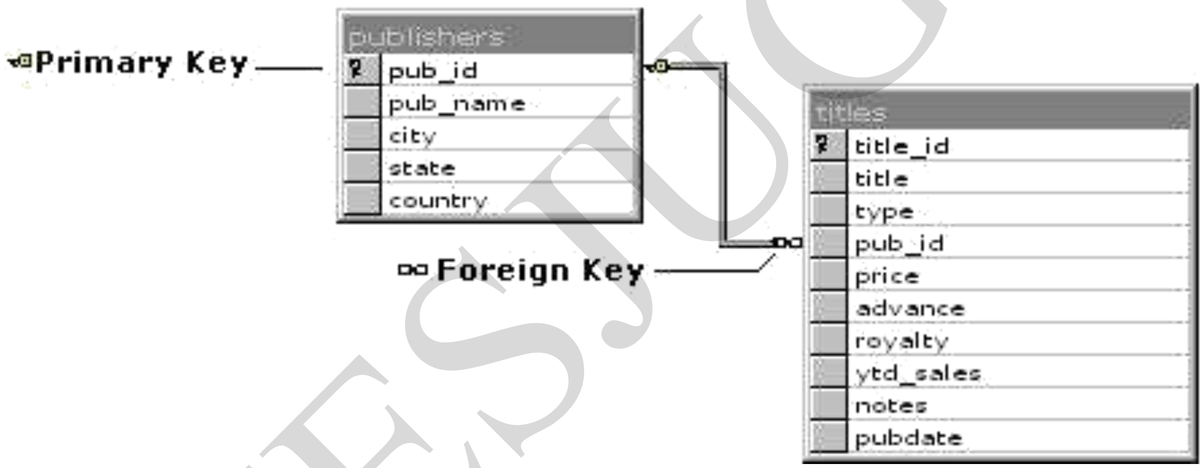


Figure 6 : referential integrity

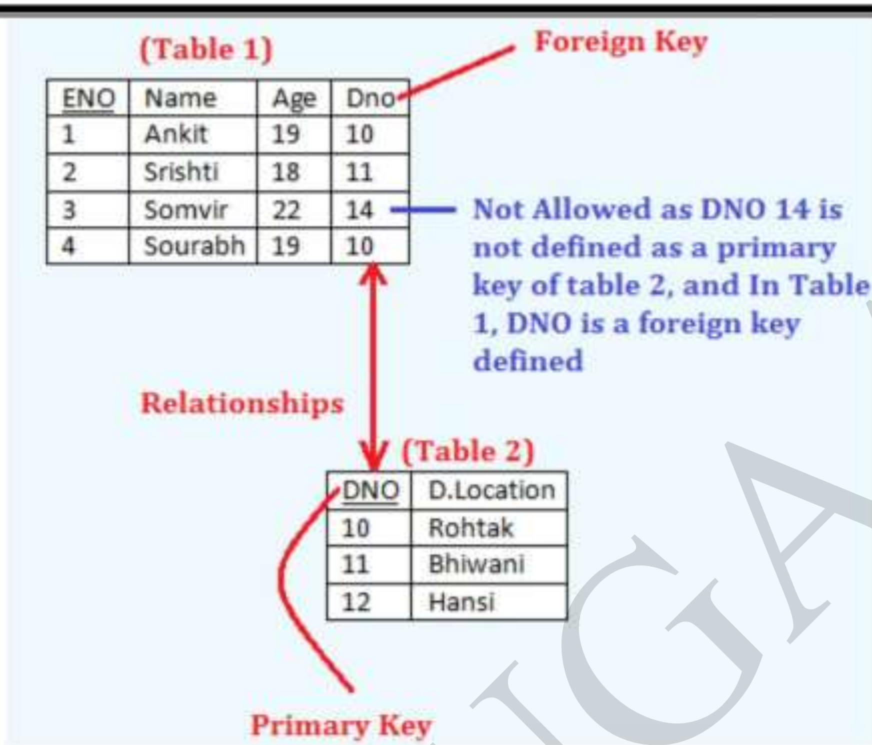


Figure 7:referential example2

Let the table in which the foreign key is defined is Foreign Table or details table i.e. Table 1 in above example and the table that defines the primary key and is referenced by the foreign key is master table or primary table i.e. Table 2 in above example. Then the following properties must be hold :

- Records cannot be inserted into a Foreign table if corresponding records in the master table do not exist.
- Records of the master table or Primary Table cannot be deleted or updated if corresponding records in the detail table actually exist.

Example of referential integrity

Person (master table)

P_id	Name	Fname	Address	City

Orders (detail table) F.K

O_id	Orderno	P_id

Figure 8 :example of referential integrity

Referential integrity enforces the following rules:

1. We cannot add a new record in orders table unless P_id in orders table points to a valid record in persons table.
2. If the primary key for a record in the persons table changes, then all corresponding records in the orders table must be modified using a cascading update.
3. If a record in persons table is deleted, all corresponding records in the orders table must be deleted using a cascading delete.

Relational database schema

A relational database schema is the tables, columns and relationships that make up a relational database.

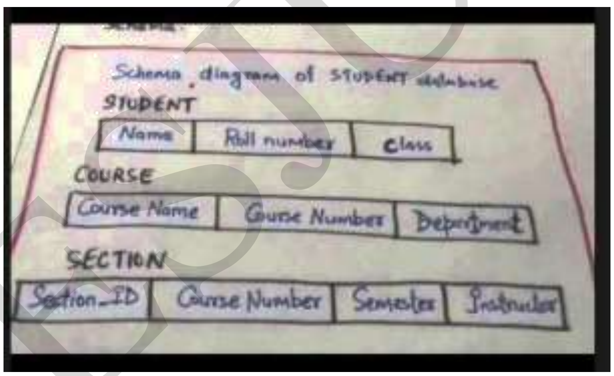


Figure 9: schema

The Purpose of a Schema

- A relational database schema helps you to organize and understand the structure of a database. This is particularly useful when designing a new database, modifying an existing database to support more functionality, or building integration between databases.

Creating the Schema

- There are two steps to creating a relational database schema:

1. creating the logical schema and
2. creating the physical schema.

1. The logical schema depicts the structure of the database, showing the tables, columns and relationships with other tables in the database and can be created with modelling tools or spreadsheet and drawing software.

2. The physical schema is created by actually generating the tables, columns and relationships in the relational database management software (RDBMS). Most modelling tools can automate the creation of the physical schema from the logical schema, but it can also be done by manually.

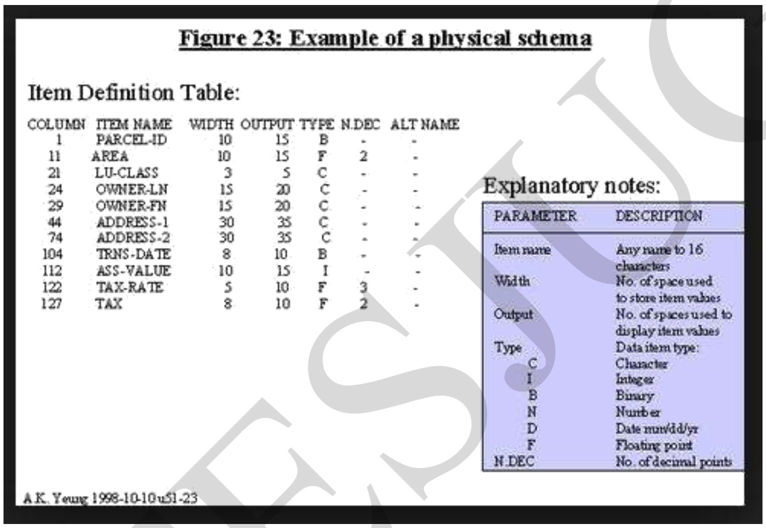


Figure 23: Example of a physical schema

Item Definition Table:

COLUMN	ITEM NAME	WIDTH	OUTPUT	TYPE	N.DEC	ALT NAME
1	PARCEL-ID	10	15	B	-	-
11	AREA	10	15	F	2	-
21	LU-CLASS	3	5	C	-	-
24	OWNER-LN	15	20	C	-	-
29	OWNER-FN	15	20	C	-	-
44	ADDRESS-1	30	35	C	-	-
74	ADDRESS-2	30	35	C	-	-
104	TRNS-DATE	8	10	B	-	-
112	ASS-VALUE	10	15	I	-	-
122	TAX-RATE	5	10	F	3	-
127	TAX	8	10	F	2	-

Explanatory notes:

PARAMETER	DESCRIPTION
Item name	Any name to 16 characters
Width	No. of space used to store item values
Output	No. of spaces used to display item values
Type	Data item type:
C	Character
I	Integer
B	Binary
N	Number
D	Date mm/dd/yr
F	Floating point
N.DEC	No. of decimal points

A.K. Yeung 1998-10-10u51-23

Codd's Rules (rules to convert DBMS to RDBMS)

Dr E.F codd developed the relational data model in 1970.In 1985,Dr Codd published a list of 12 rules that defines an ideal relational databases.

Rule Zero

• For a system to qualify as an RDBMS it must be able to manage its databases entirely through its Relational capabilities without using any external language.

• The other 12 rules derive from this rule

Rule 1: Information Rule

All data should be presented in table form.The table form gives ease and flexibilty to access data. Rule 2: Guaranteed Access Rule

All data should be accessible without ambiguity. Each unique piece of data (atomic value) should be accesible by : TableName + Primary Key (Row) +Attribute (Column)
Rule 3: Systematic Treatment of Null values

A field should be allowed to remain empty. This involves the support of a null value, which is different from an empty string or zero value.

Rule 4: Database Description Rule

A data dictionary should be present within RDBMS that is constructed from tables which can be examined using SQL

Rule 5: comprehensive data sub language rule

The database must support at least one clearly defined language that includes functionality for data definition ,data manipulation ,data integrity and database transaction control like SQL.

Rule 6: view updating rule

Data can be presented in different logical combination called views. Each view should support the same full range of data manipulation that has access to a table available. Rule 7: High level insert,update delete.

Data can be retrieved from a relational database in sets constructed of data from various row and/or multiple tables

Rule 8: Physical Data Independence

The user is isolated from the physical method of storing and retrieving information from database

Rule 9: Logical data Independence

When the table structure of the database changes,then data should not be changed

Rule 10: Integrity Independence

The database language like SQL should support constraints on user input that maintains database integrity
Rule 11: Distribution Independence
A user should be totally unaware of whether or not the database is distributed

Rule 12: Non subversion rule
There should be no way to modify the database structure other than through the multiple row database language like sql

Unit III part II:Relational algebra:

QUERY LANGUAGES:

Allow manipulation and retrieval of data from a database.

Formal Relational Query Languages

Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
Relational Algebra: More operational, very useful for representing execution plans.
Relational Calculus: Lets users describe what they want, rather than how to compute it (Non-operational, declarative.)

- ✓ A query is applied to relation instances, and the result of a query is also a relation instance.

Notation

The operations have their own symbols. The symbols are hard to write in HTML that works with all browsers, so I'm writing **PROJECT** etc here. The real symbols:

Operation	My HTML	Symbol
Projection	PROJECT	π
Selection	SELECT	σ
Renaming	RENAME	ρ
Union	UNION	$\cup$
Intersection	INTERSECTION	$\cap$

Operation	My HTML	Symbol
Cartesian product	X	$\times$
Join	JOIN	$\bowtie$
Left outer join	LEFT OUTER JOIN	⟕
Right outer join	RIGHT OUTER JOIN	⟖
Full outer join	FULL OUTER JOIN	⟗

Semijoin	SEMIJOIN	$\ltimes$

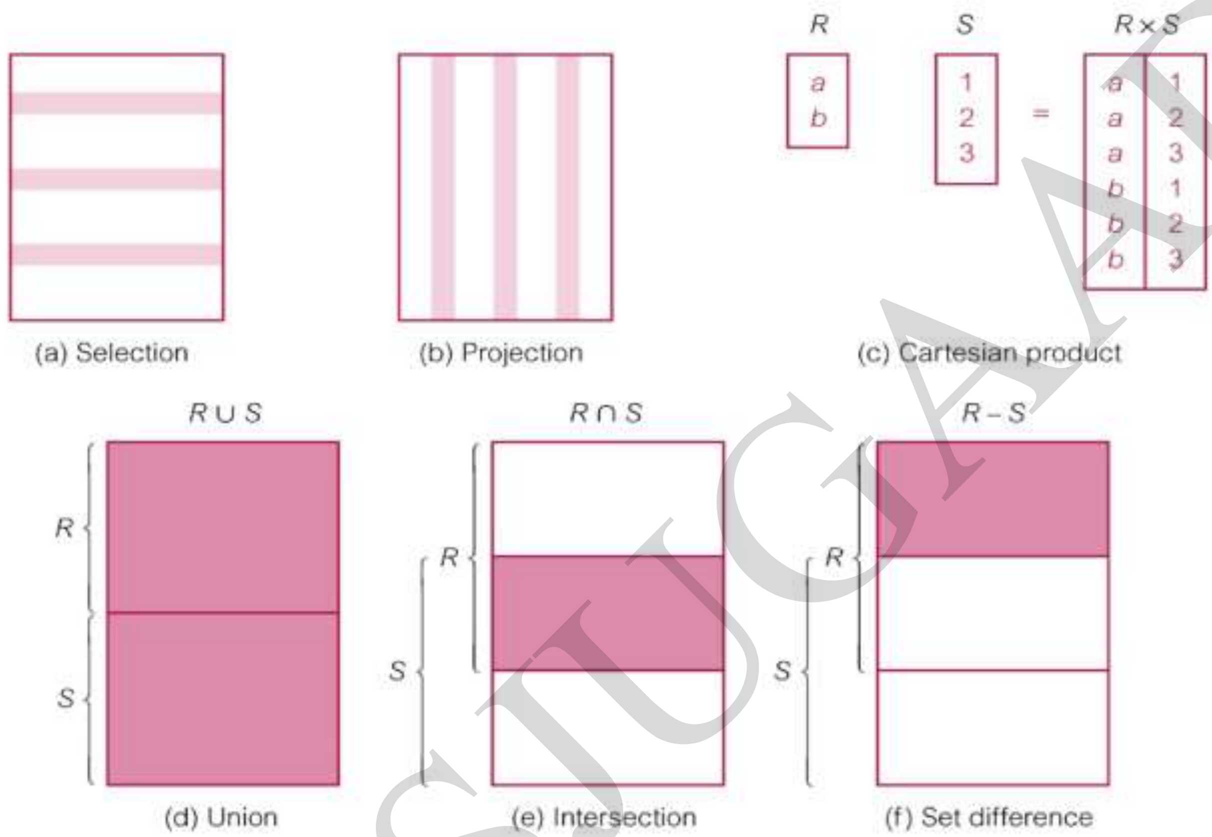


Figure 10: relational algebra

Set theoretic operations:

Intersection

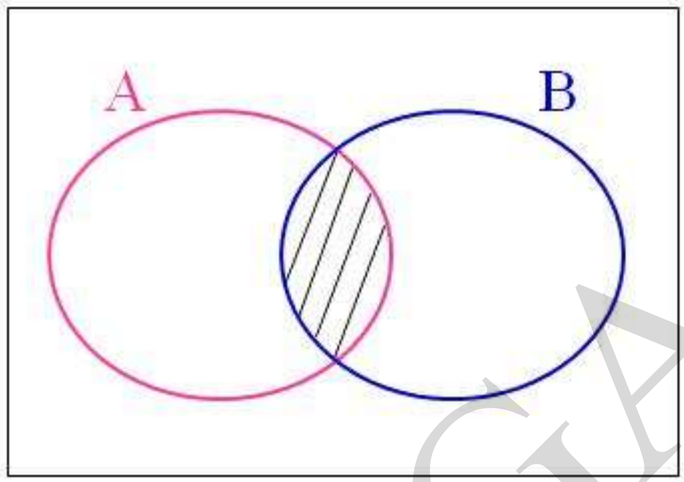


Figure 11 :intersection

Union

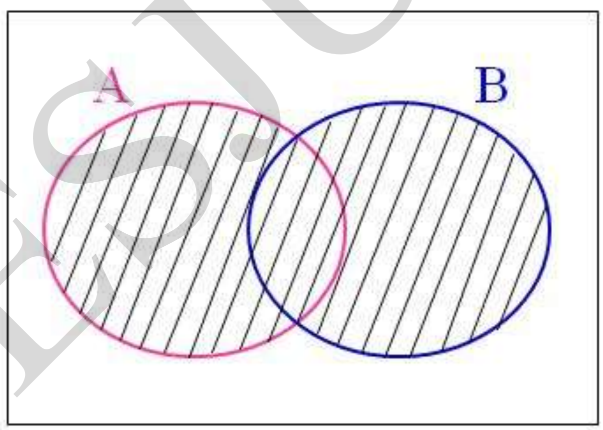


Figure 12: union

Set Difference

The *difference* of A and B (A–B)

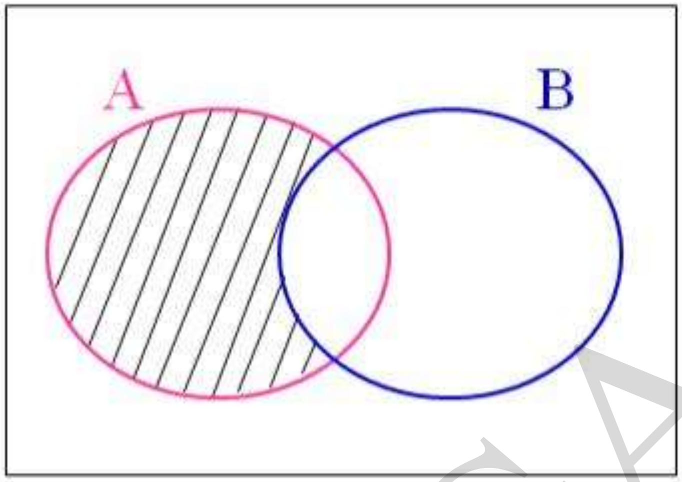


Figure 13:set difference

Example

R

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamil	456 Oak Rd., Brentwood	M	8/8/88

S

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

© CIS 4301 - Spring 2006 Lecture 15 9

Sample Operations

$R \cap S$	name	address	gender	birthdate
	Carrie Fisher	123 Maple St., Hollywood	F	9/9/99

$R \cup S$	name	address	gender	birthdate
	Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
	Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77
	Mark Hamil	456 Oak Rd., Brentwood	M	8/8/88

$R - S$	name	address	gender	birthdate
	Mark Hamil	456 Oak Rd., Brentwood	M	8/8/88

EXAMPLE FOR PROJECTION & SELECTION

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

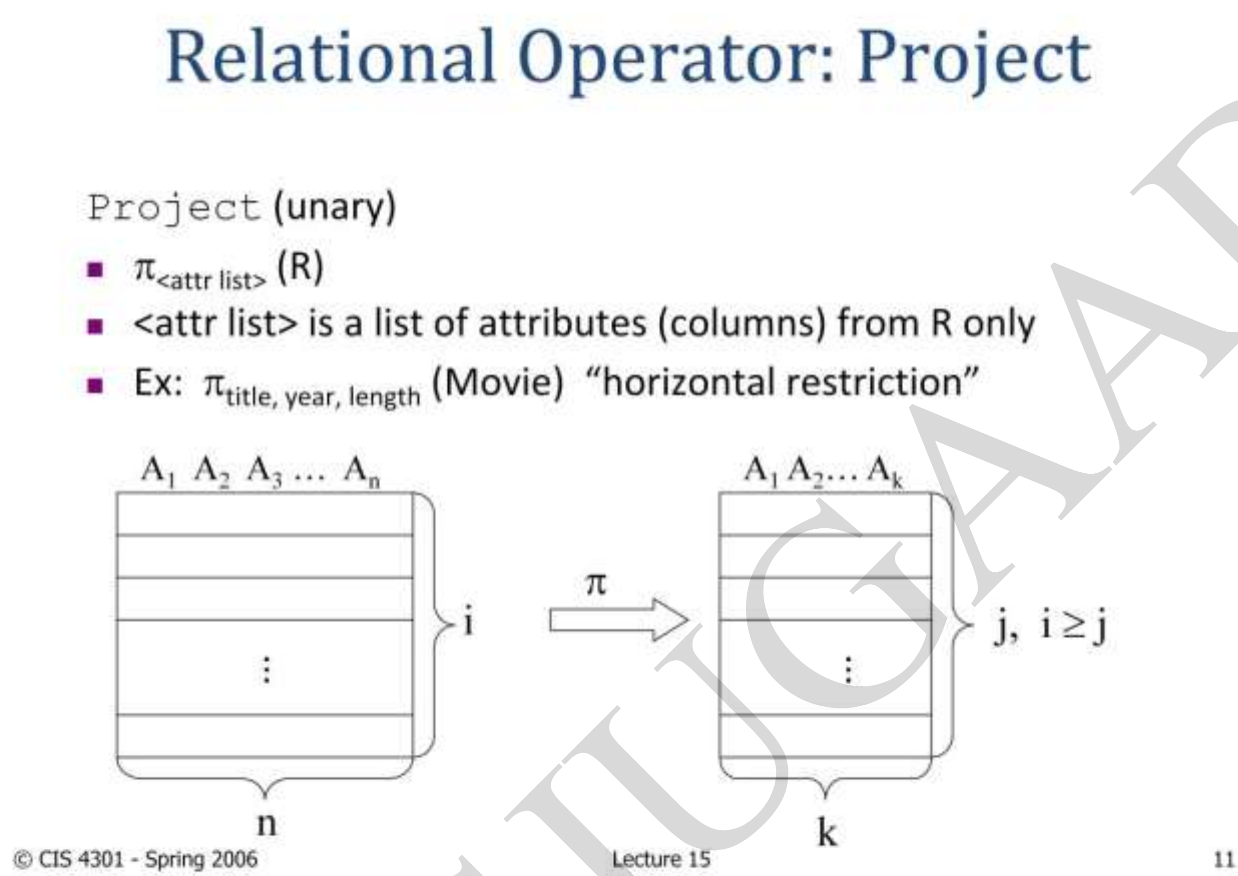
sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance *S2* of Sailors

basic operations

Projection

Deletes unwanted columns from relation.



Example: The table **E** (for **EMPLOYEE**)

nr	name	salary
1	John	100
5	Sarah	300
7	Tom	100

SQL	Result	Relational algebra
select salary from E	salary 100 300	**PROJECT**$_{salary}$(E)
select nr, salary from E	nr salary 1 100 5 300	**PROJECT**$_{nr, salary}$(E)

	7 100	

Note that there are no duplicate rows in the result.

Example of Projection

Employees

Surname	FirstName	Department	Head
Smith	Mary	Sales	De Rossi
Black	Lucy	Sales	De Rossi
Verdi	Mary	Personnel	Fox
Smith	Mark	Personnel	Fox

$\pi_{Surname,\ FirstName}$(Employees)

Surname	FirstName
Smith	Mary
Black	Lucy
Verdi	Mary
Smith	Mark

Another Example

Employees

Surname	FirstName	Department	Head
Smith	Mary	Sales	De Rossi
Black	Lucy	Sales	De Rossi
Verdi	Mary	Personnel	Fox
Smith	Mark	Personnel	Fox

$\pi_{Department,\ Head}$(Employees)

Department	Head
Sales	De Rossi
Personnel	Fox

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

Ques. Select the columns sname and rating.
as we need to select columns, we have to use projection.

sname	*rating*
yuppy	9
Lubber	8
guppy	5
Rusty	10

Figure 4.5 $\pi_{sname,rating}(S2)$

Selection Operation

The same table **E** (for **EMPLOYEE**) can be written as :

<table>
<tr><td>SQL</td><td>Result</td><td>Relational algebra</td></tr>
<tr><td>select *
from E
where salary < 200</td><td><table><tr><td>nr</td><td>name</td><td>salary</td></tr><tr><td>1</td><td>John</td><td>100</td></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>SELECT$_{salary < 200}$(E)</td></tr>
<tr><td>select *
from E
where salary < 200
and nr >= 7</td><td><table><tr><td>nr</td><td>name</td><td>salary</td></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>SELECT$_{salary < 200 \text{ and } nr >= 7}$(E)</td></tr>
</table>

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

Selection Example

Employees

Surname	FirstName	Age	Salary
Smith	Mary	25	2000
Black	Lucy	40	3000
Verdi	Nico	36	4500
Smith	Mark	40	3900

$\sigma_{Age<30 \vee Salary>4000}$ **(Employees)**

Surname	FirstName	Age	Salary
Smith	Mary	25	2000
Verdi	Nico	36	4500

Selection, Another Example

Citizens

Surname	FirstName	PlaceOfBirth	Residence
Smith	Mary	Rome	Milan
Black	Lucy	Rome	Rome
Verdi	Nico	Florence	Florence
Smith	Mark	Naples	Florence

$\sigma_{PlaceOfBirth=Residence}$ **(Citizens)**

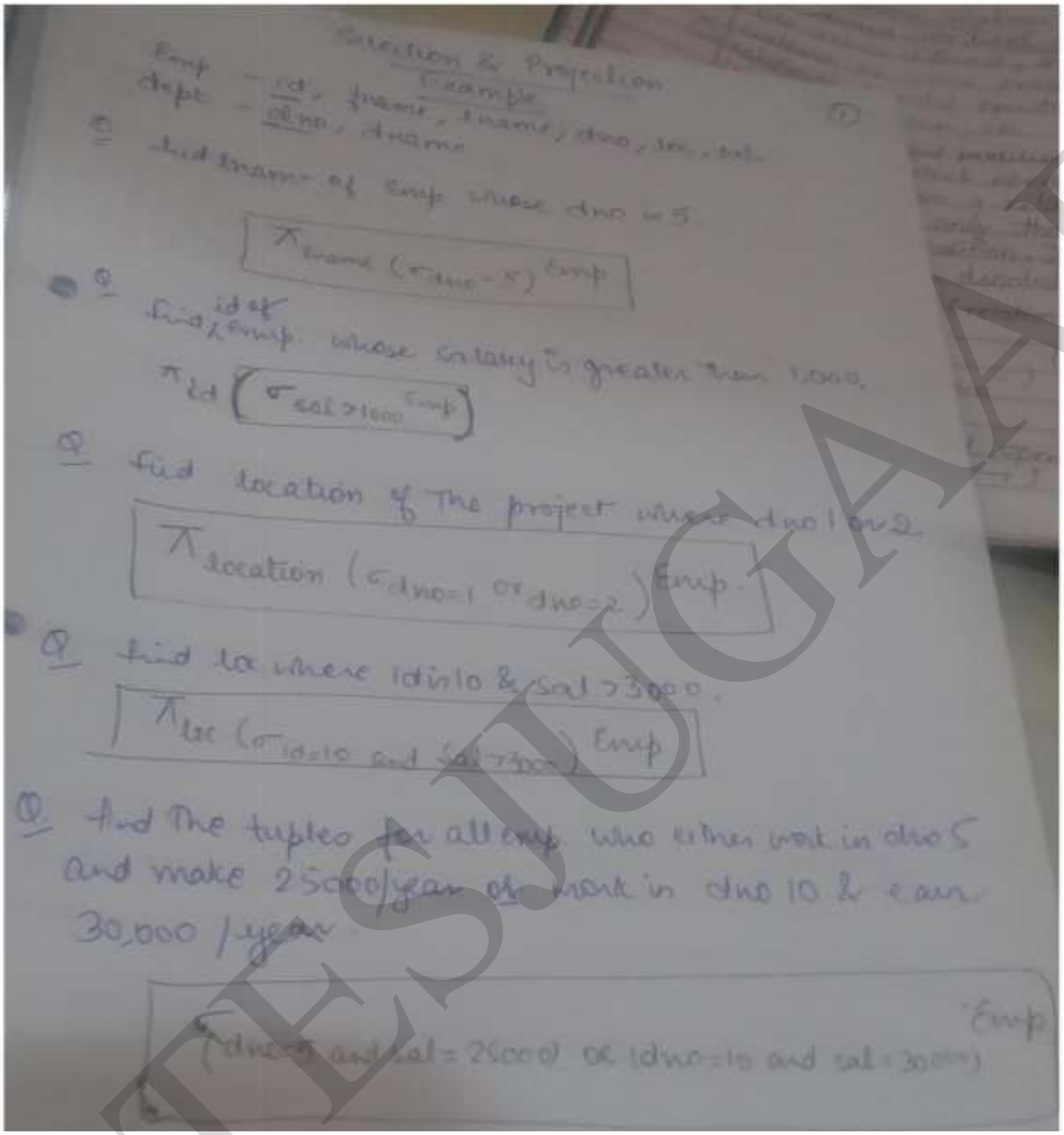
Surname	FirstName	PlaceOfBirth	Residence
Black	Lucy	Rome	Rome
Verdi	Nico	Florence	Florence

Ques. FIND THE RECORDS HAVING RATING GREATER THAN 8.

For this, we need to find the rows where value of rating is greater than 8.
So, Selection is to be used.

sid	sname	rating	age
28	yuppy	9	35.0
58	Rusty	10	35.0

Figure 4.4 $\sigma_{rating>8}(S2)$



Selection & Projection
Example

Emp – id, fname, lname, dno, sex, sal.
dept – dno, dname

Q find lname of emp whose dno is 5.

$\pi_{lname} (\sigma_{dno=5})$ Emp

Q find id of emp whose salary is greater than 1000.

$\pi_{id} (\sigma_{sal>1000}$ Emp$)$

Q find location of the project where dno 1 or 2.

$\pi_{location} (\sigma_{dno=1 \text{ or } dno=2})$ Emp.

Q find loc where id is 10 & sal >3000.

$\pi_{loc} (\sigma_{id=10 \text{ and } sal>3000})$ Emp

Q find the tuples for all emp who either work in dno 5 and make 25000/year or work in dno 10 & earn 30,000/year.

$\sigma_{(dno=5 \text{ and } sal=25000) \text{ or } (dno=10 \text{ and } sal=30000)}$ Emp

rename operation ρ (rho symbol)

rename operation is used to perform following functions

a. Rename relation

rho $_{\text{new name}}$ old name

b. Rename column

rho $_{\text{newcolname}}$ tablename

For example

emp-> id, name
change id to eno

rho $_{\text{eno,name}}$ emp

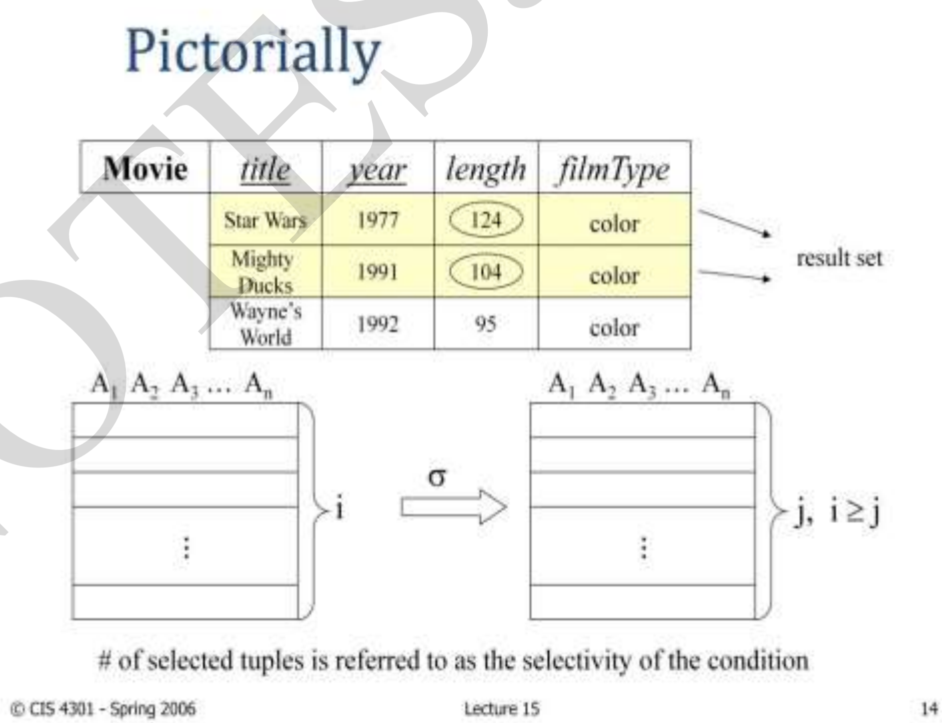
c. Rename table name as well as attribute name

ρ $_{\text{newtablename (new col name)}}$old tablename

For example

emp-> id, name
change id to eno and
emp to details

ρ $_{\text{details(eno,name)}}$ emp



Cartesian product

The *cartesian product* of two tables combines each row in one table with each row in the other table.

Example: The table **E** (for **EMPLOYEE**)

enr	ename	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

<table>
<tr><td>SQL</td><td>Result</td><td>Relational algebra</td></tr>
<tr><td>select *
from E, D</td><td>
<table>
<tr><td>enr</td><td>ename</td><td>dept</td><td>dnr</td><td>dname</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>B</td><td>Sales</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>C</td><td>Legal</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>B</td><td>Sales</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>B</td><td>Sales</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>C</td><td>Legal</td></tr>
</table>
</td><td>E X D</td></tr>
</table>

Seldom useful in practice.
Usually an error.
Can give a huge result.

Unit III part III

Join operations

SQL provides a convenient operation to retrieve information from multiple tables.

This operation is called **join**.

The join operation will **combine** the tables into one large table with all possible combinations (Math: Cartesian Product), and then it will filter the rows of this combined table to yield useful information.

JOIN V/S CARTESIAN PRODUCT

In JOIN , only combination of tuples satisfying the join condition appears in the result**, whereas in cartesian product,** all combination of tuples are included in the result.

The join keyword is used in SQL statement to query data from 2 or more tables,based on a relatonship between certain columns in these tables.

Note: there nust be atleast one common attribute in both tables.

Joins are of follwing types:

1. Equi join   inner join
2. Theta join
3. Natural join
4. left join or left outer join   outer join
5. right join or right outer join
6. full join

equi join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

A join where the only comparison operator is "=",is called an equi join.
For example

Employee

Fname	Ssn	Salary	Dno

Department

Dname	Dnumber	Mgr_ssn

<u>Q. Retrieve the name of the manager of each department.</u>

Ans:

To do so, we need to combine each department tuple with employee tuple whose SSN value matches Mgr_ssn value in department tuple.

Fname	Ssn	Salary	Dno
A	5	20,000	6
B	6	25,000	1
C	7	30,000	2
D	22	31,000	3

Dname	Dnumber	Mgr_ssn
HR	1	6
Mkt.	2	7
IT	3	22

Mgr $\leftarrow$ Department $\infty_{mgr\_ssn=ssn}$ Employee

mgr

Fname	Ssn	Salary	Dno	Dname	Dnumber	Mgr_ssn
B	6	25,000	1	HR	1	6
C	7	30,000	2	Mkt.	2	7
D	22	31,000	3	IT	3	22

as we need to find the name of employee , we have to use project operation.

$result \leftarrow \pi_{fname}(mgr)$

disadvantage of equi join :
after doing join operation, resultant table contains repeating values of attributes, which is not deesirable.

THETA JOIN

$R \bowtie_F S$

Defines a relation that contains tuples satisfying the condition F from the Cartesian product of R and S.

The predicate F is of the form R.ai q S.bi where q may be one of the comparison operators (<, >,>=,<=)

$relation\ 1 \bowtie_{condition} relation\ 2$

Example of Theta join

Car Model	Car Price
Car A	20000
Car B	30000
Car C	50000

Boat model	Boat Price
Boat 1	10000
Boat 2	40000
Boat 3	60000

Q. find the cars having rate mrethan bat price.

Sol.

$my\_car \leftarrow car \bowtie_{car\ price\ >\ boat\ price} Boat$

Car Model	Car Price	Boat Model	Boat Price
Car A	20000	Boat 1	10000
Car B	30000	Boat 1	10000
Car C	50000	Boat 1	10000
Car C	50000	Coat 2	40000

Natural join (*)

Invariably the JOIN involves an equality test, and thus is often described as an equi-join. Such joins result in two attributes in the resulting relation having exactly the same value. **A `natural join' will remove the duplicate attribute(s).**

- In most systems a **natural join will require that the attributes have the same name** to identify the attribute(s) to be used in the join. This may require a renaming mechanism.
- If you do use natural joins make sure that the relations do not have two attributes with the same name by accident.

example of natural join

deptt

dname	dno	mgrssn

project

pname	pno	ploc	dnum

after doing join operation ,we get

deptt_project

dname	dno	mgrssn	pname	pno	ploc	dnum

to use natural join, we need to firstly rename dnum to dno, so that the resulting table have one

occurrence for dnum
i.e

new_deptt_project ← rho $_{\text{pname,pno,ploc,dno}}$ (project)

now join deptt & new_deptt_project

final_result ← deptt * new_deptt_project

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

INNER JOIN

This join returns rows when there is at least one match in both the tables.

OUTER JOIN

This join returns all the rows from the left table in conjunction with the matching rows from the right table. If there are no columns matching in the right table, it returns NULL values.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

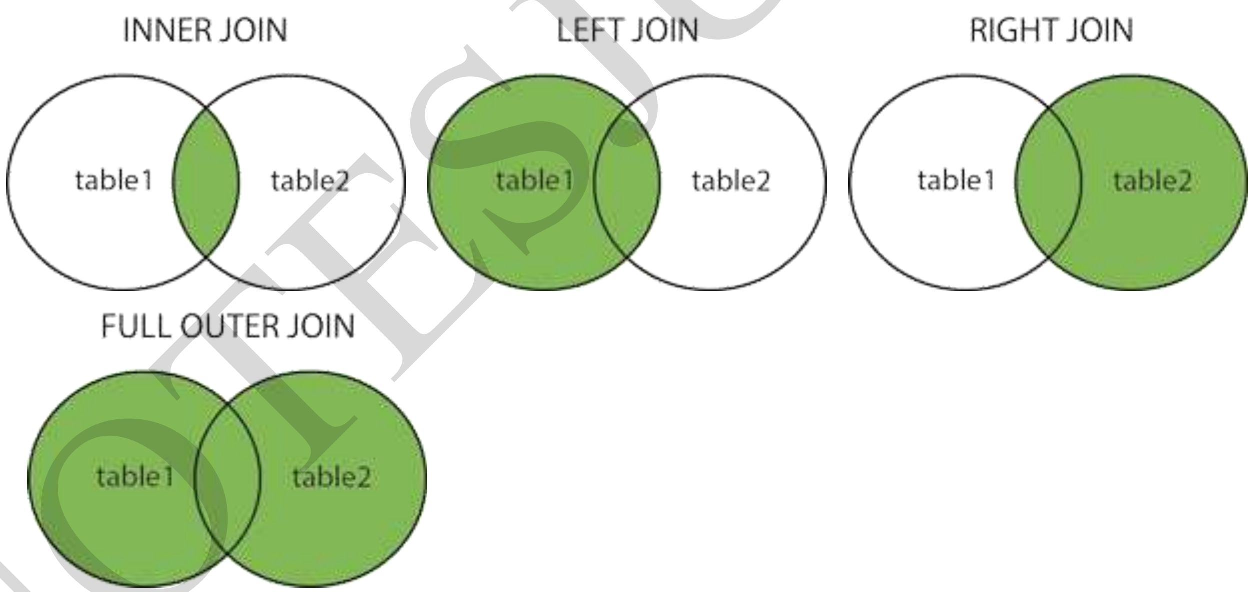
OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996

10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common

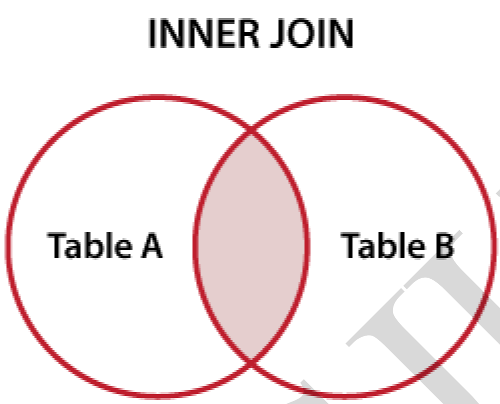
field will be same.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

- This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

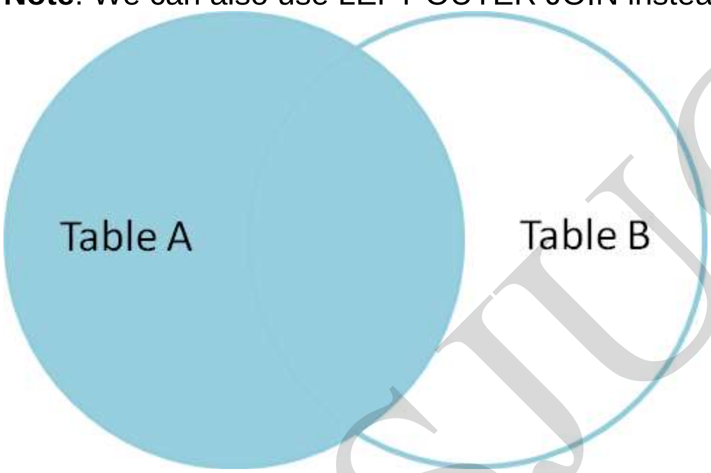
11. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there

is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.**Syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

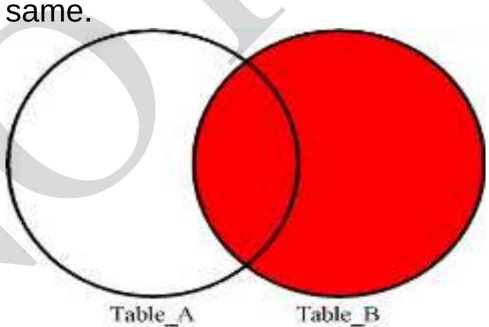
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

21. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.**Syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
```

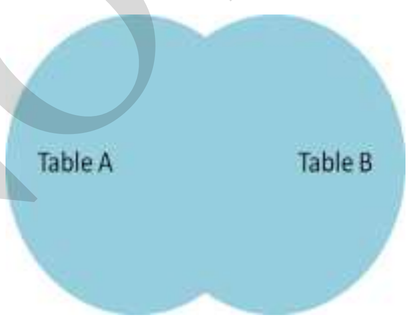
```
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

31. **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.**Syntax:**

```
32.   SELECT table1.column1,table1.column2,table2.column1,....
33.   FROM table1
34.   FULL JOIN table2
35.   ON table1.matching_column = table2.matching_column;
36.
37.
38.   table1: First table.
39.   table2: Second table
40.   matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
```

```
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

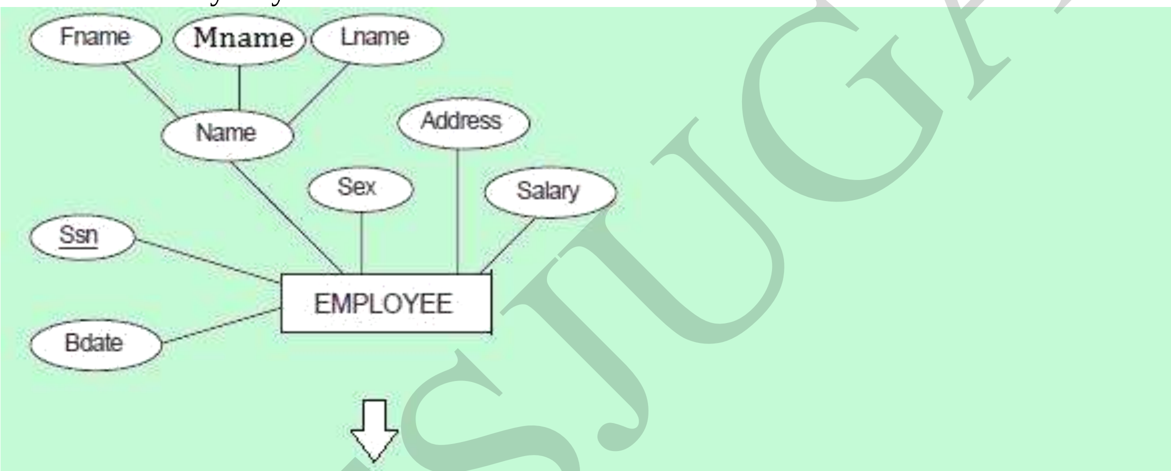
Unit III part IV :ER to relational Mapping:

To convert the er model to relational model there are 7 Steps to be followed, which are

Conversion of Strong Entities –
Conversion of Weak Entities
Conversion of one to one Relationships
Conversion of One to Many Relationships
Conversion of Many to Many Relationships
Conversion of n-ary Relationships
Conversion of Multivalued Attribute

1. Conversion of Strong Entities –

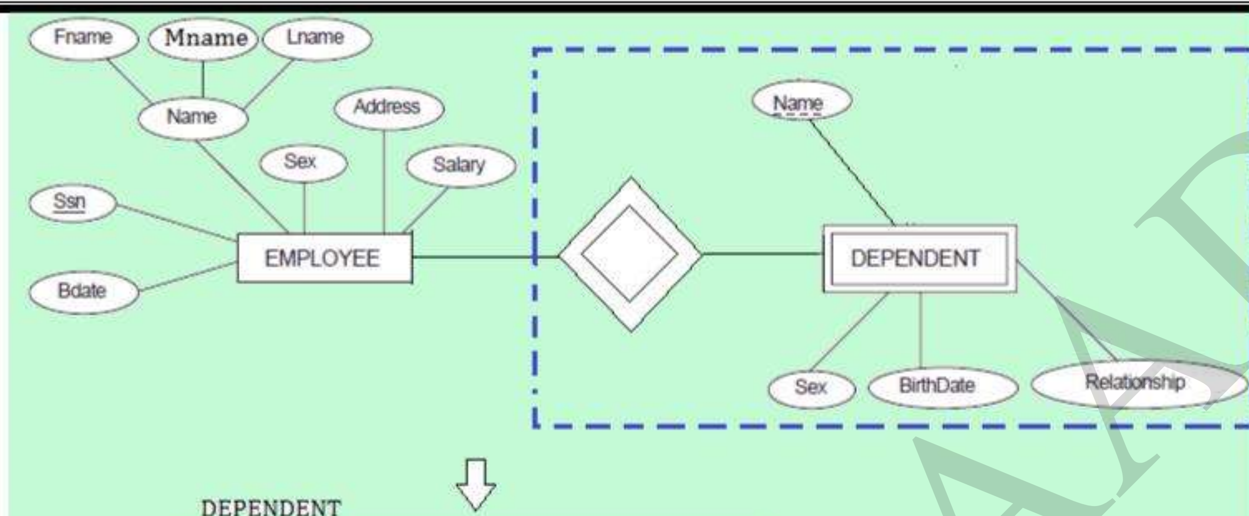
For each strong entity in ERD, create a separate table with the same name.
Create all simple Attributes
Break the Composite attributes into simple attributes and create them.
Choose a Primary Key for the table



SSN	Fname	Mname	Lname	Sex	Bdate	Address	Salary

2. Conversion of Weak Entities –

For each weak entity, create a separate table with the same name.
Include Primary Key of the strong entity as a foreign key in the table.
Select the Primary Key attributes of strong entity and the partial Key attribute of the weak entity, and declare them as primary key



DEPENDENT

SSN	Dependent Name	Sex	Birthdate	Relationship

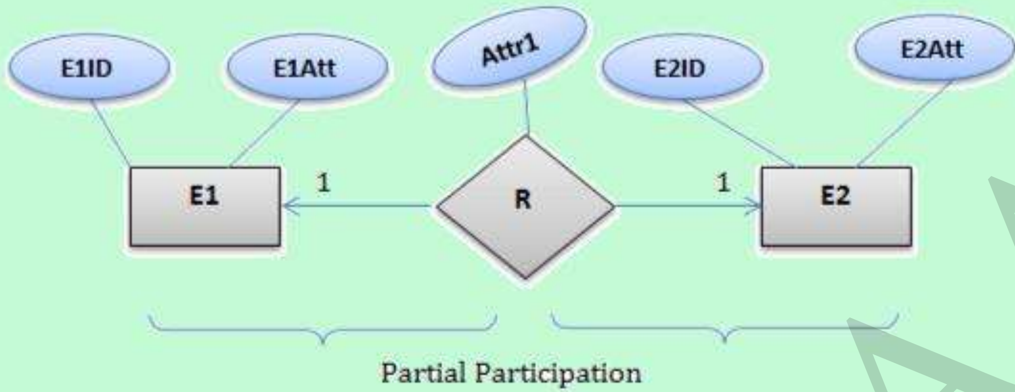
Primary Key : SSN Dependent Name

3. Conversion of One to One Relationships –

There are two possible approaches on the basis of Participation Constraints –

Partial Participation on Both Sides – For each One to One Cardinality between E1 and E2 with partial participation on both sides, modify either E1 or E2 to include the primary key of other table as a foreign key.
So, 1:1 cardinality with partial participation on both sides can be minimised into two relations only.

One to One Conversion with Partial Participation on Both Sides -



E1ID	E1Attr
1	x
2	x
3	y

E1ID	E2ID
1	1003
2	1001

E2ID	E2Attr
1001	p
1002	q
1003	q

PK : Primary Key
FK : Foreign Key

Modification Can be done as -

E1ID	E1Attr	E2ID
1	X	q
2	X	p
3	Y	Null

PK : E1ID
FK : E2id

E2ID	E2Attr
1001	p
1002	q
1003	q

PK : E2ID

(OR)

E2ID	E2Attr	E1ID
1001	p	2
1002	q	Null
1003	q	1

PK : E2ID
FK : E1ID

E1ID	E1Attr
1	x
2	x
3	y

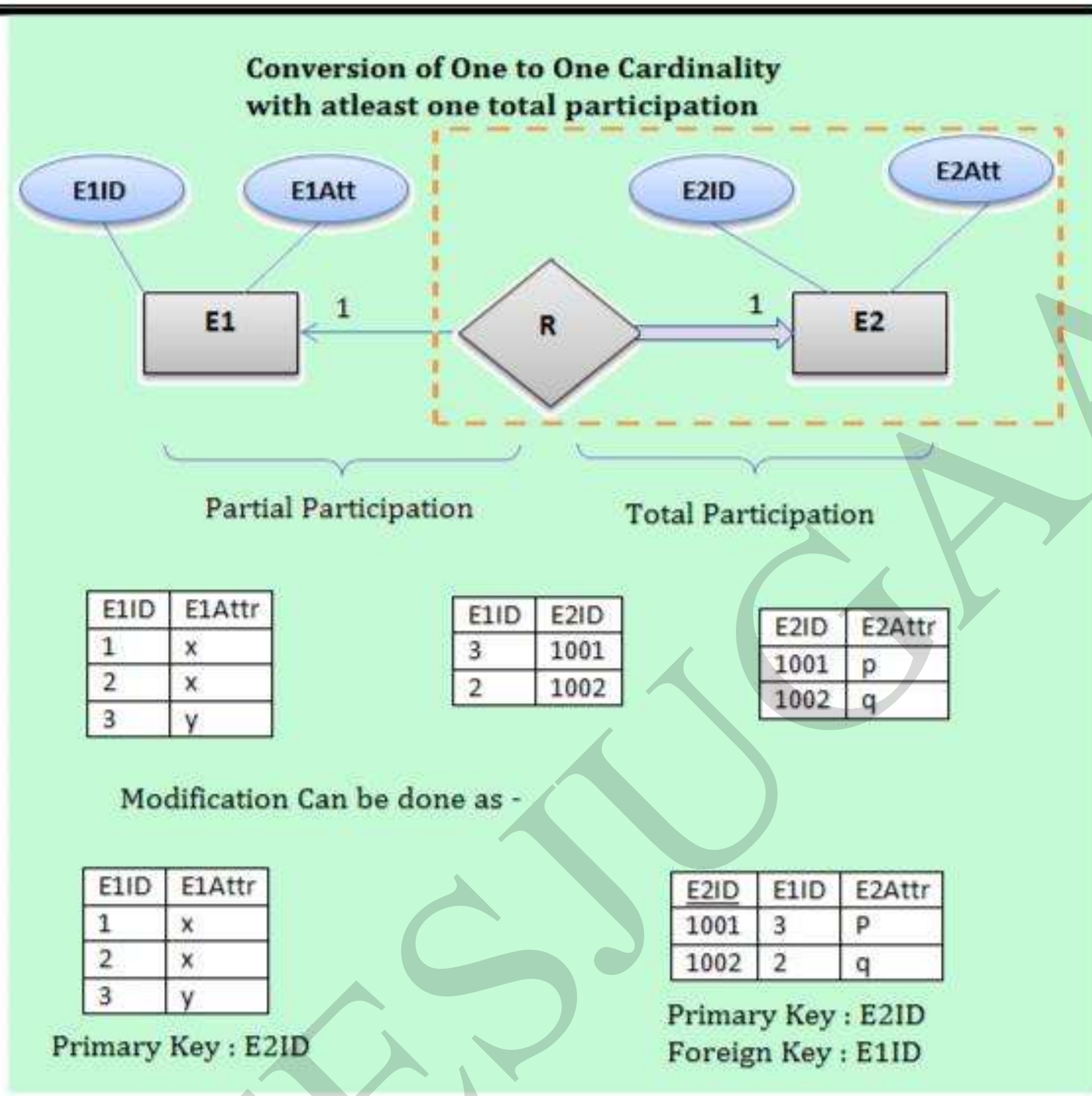
PK : E1ID

If we try to minimize the above ERD in a single table, i.e. E1RE2, then it contains too many NULL values, and therefore, we are not be able to select a primary key. For example,

E1ID	E1Atr	E2ID	E2Attr
1	x	1003	q
2	x	1001	p
3	y	Null	Null
Null	Null	1002	q

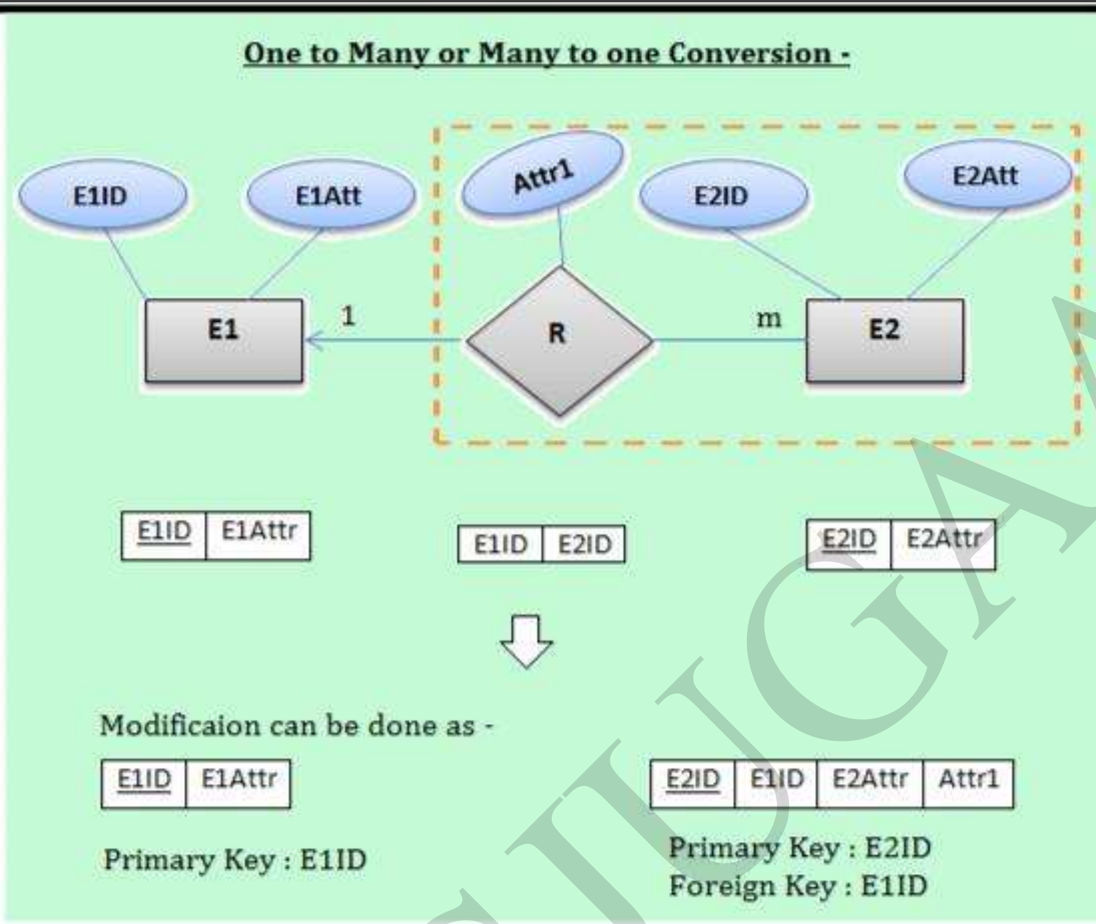
Unable to select a Primary Key, because all attribute have some null values. Therefore, can't be able to identify

Cardinality with atleast one Total Participation – For each One to One Cardinality between E1 and E2 with atleast one total participation, modification is done only on total participation side. So, One to One Cardinality with atleast one Total Participation can be minimized into a single relation.



4. Conversion of One to Many or Many to One Relationship –

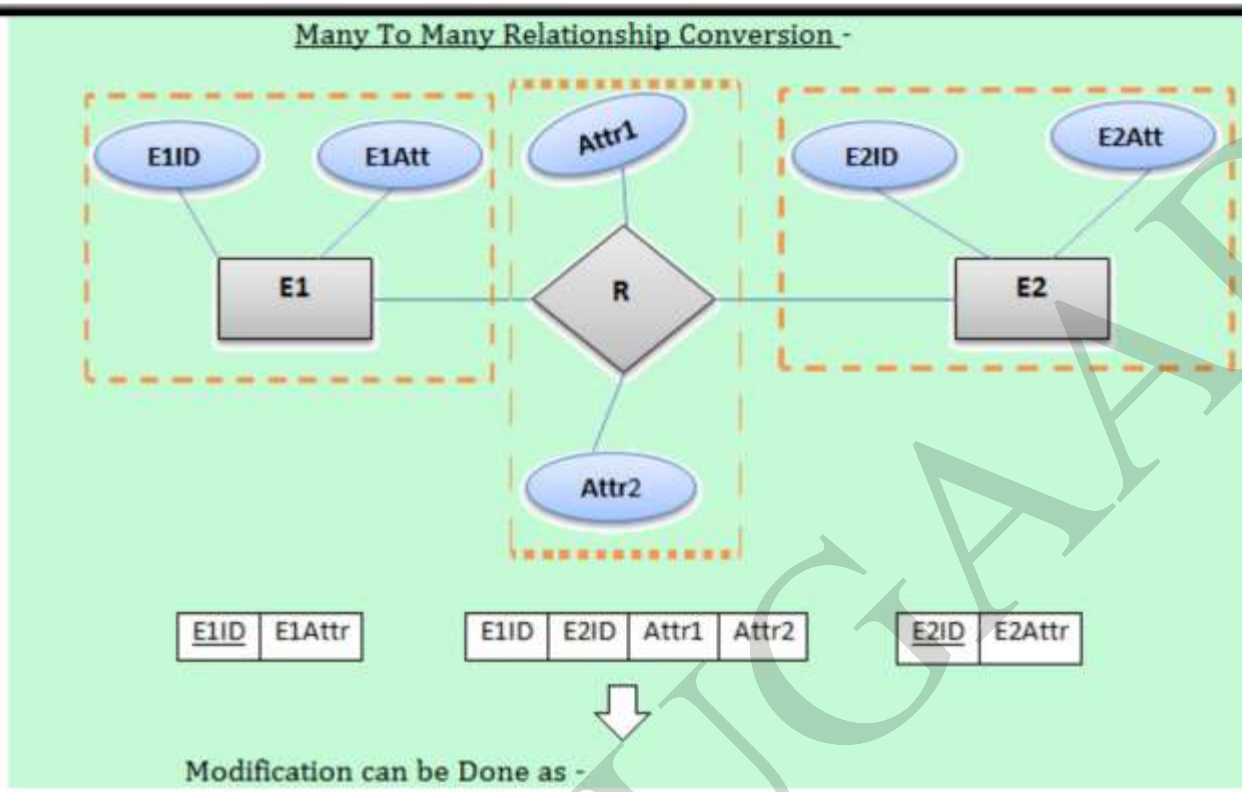
For each one to many relationship between E1 and E2, modify many side relation to include from one side as a Foreign Key.



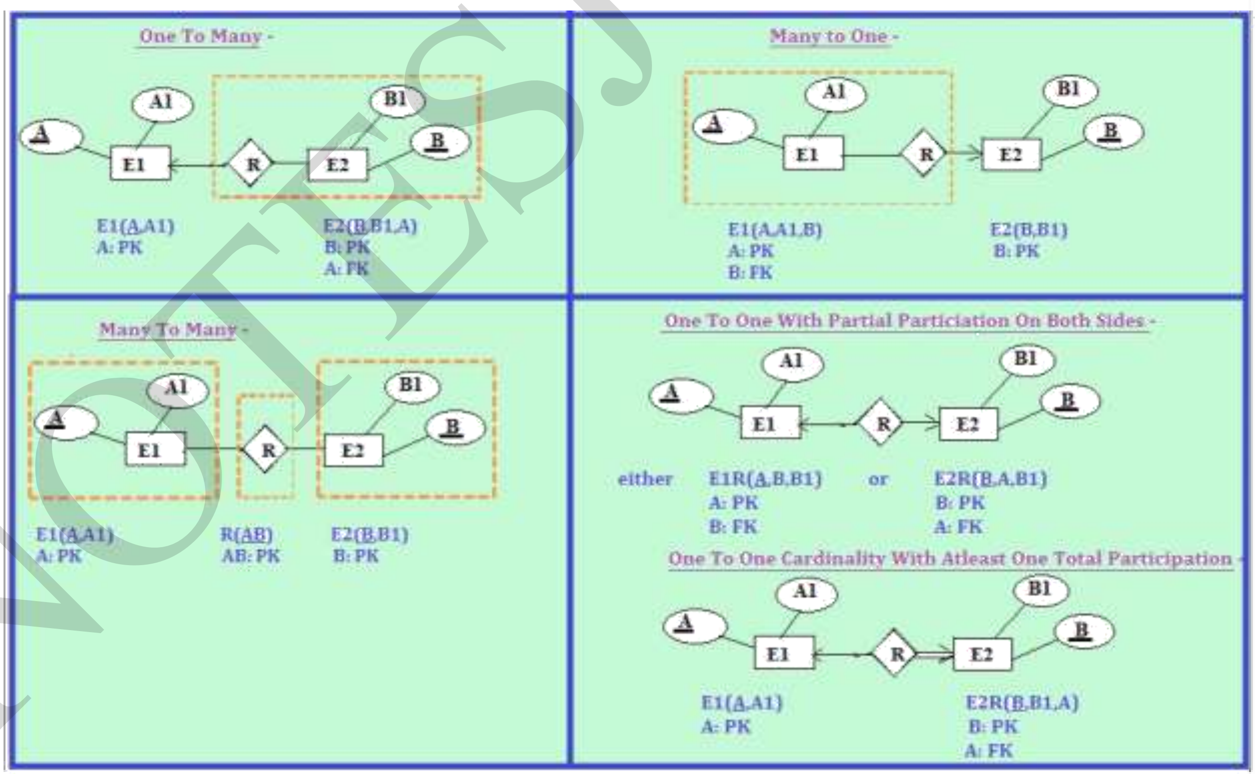
5. Conversion of Many to Many Relationship –

For each one to many relationship between E1 and E2, create a separate table and include primary key of both the tables as a Foreign Key.

If relationship is having one or more attributes, then these must also be included in the table.

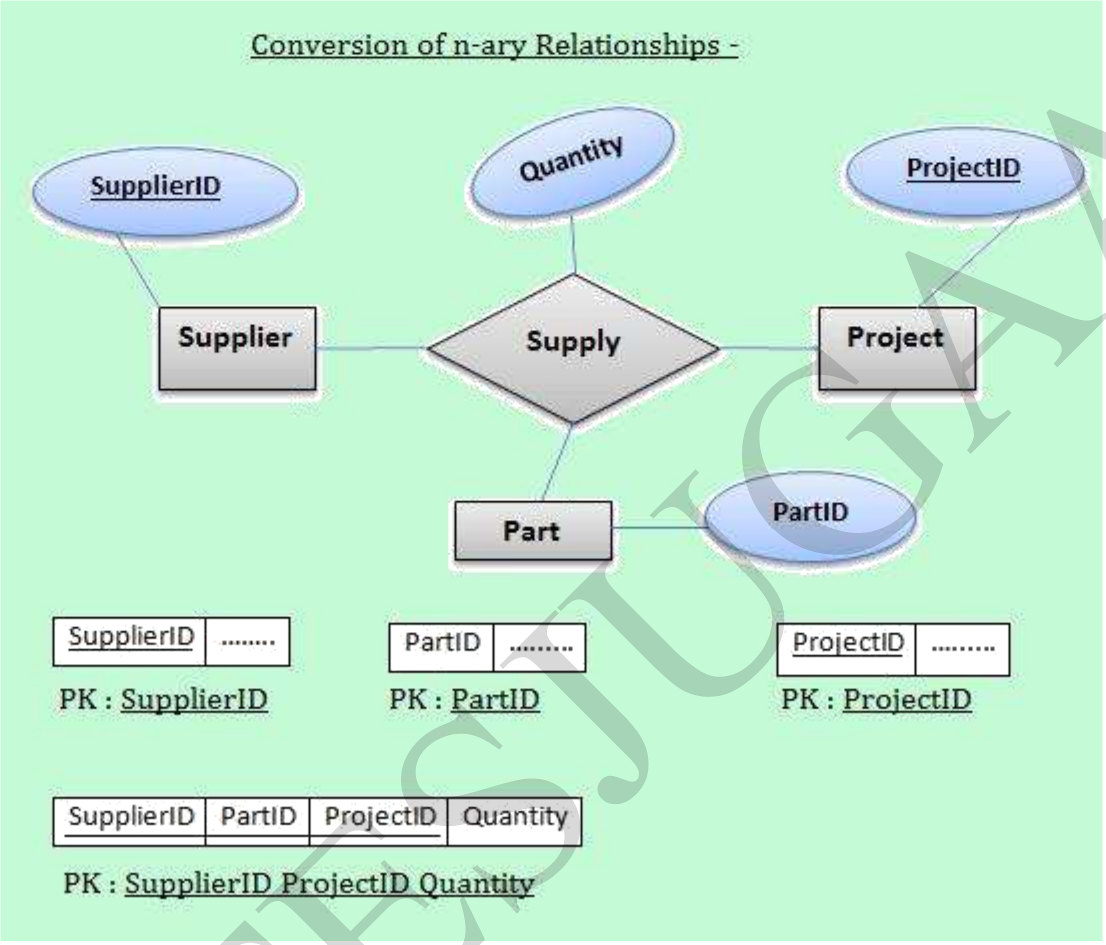


In short, The Conversion of Relationships will be done as –



6. Conversion of n-ary Relationship –

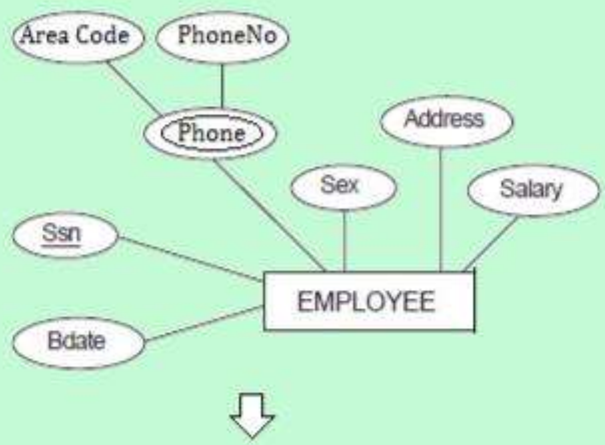
For each n-ary relationship, Create a separate table and include primary keys of all other entities as a foreign key.
If the relationships has some attributes, then these must also be included in the table.



7. Conversion of multivalued Attributes –

For each multivalued attributes, create a separate table, then include all of its simple attributes.
Include the primary key of the original table as a foreign key.

Conversion of Multivalued Attribute -



EMPLOYEE

SSN	Bdate	Sex	Address	Salary

Phone

SSN	Area_Code	Phone_No.

All the steps for conversion of er model to relational model defined above, are necessary steps. If you follow all the steps for the conversion, then you will definitely convert er model to relational model.

Unit III Part V: Data Normalization:

Normalization

The Process of Normalization

Normalization is a data analysis technique to design a database system. It allows the database designer to understand the current data structures in an organization. Furthermore, it aids any future changes and enhancements to the system.

Normalization is a technique for producing relational schema with the following properties:

1. No Information Redundancy
2. No Anomalies

Significance of Normalization

1. Improves update efficiency,
2. Removes many causes of anomalous data dependencies
3. Is (usually) better for query handling.
4. Allows better checks for consistency.

Normalization is also significant due to following reason

1. To make feasible representation of any relation in the database

2. To obtain powerful relational retrieval using relational operator

3. To free relation from undesirable insertion, update and deletion anomalies

4. To reduce the need for restructuring the relations as new data types are introduced

The end result of normalization is a set of entities, which removes unnecessary redundancy (ie duplication of data) and avoids the anomalies

DISADVANTAGES OF NORMALIZATION

The following are disadvantages of normalization.

1. You cannot start building the database before you know what the user needs.
2. On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.

3. It is very time consuming and difficult process in normalizing relations of higher degree.

In short,

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

Anomalies

These are just the contradictions.

Anomalies are inconvenient or error-prone situation arising when we process the tables. There are three types of anomalies:

1. Update / modification Anomalies
2. Delete Anomalies
3. Insert Anomalies

Update Anomalies

This type of anomaly occurs when a change of a single attribute in one record requires changes in multiple records.
An **Update Anomaly** exists when one or more instances of duplicated data is updated, but not all.

Example 1:

consider Jones moving address - you need to update all instances of Jones's address.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

Example 2:

If there is updation in the fee from 5000 to 7000, then we have to update **FEE** column in all the rows, else data will become inconsistent.

SID	Sname	CID	Cname	FEE	
S1	A	C1	C	~~5k~~	7k
S2	A	C1	C	~~5k~~	7k
S1	A	C2	C	10k	
S3	B	C2	C	10k	
S3	B	C2	JAVA	15k	

> Costly Operation

⇩

More IO Cost

Delete Anomalies

A **Delete Anomaly** exists when certain attributes are lost because of the deletion of other attributes.

example 1:
consider what happens if Student S30 is the last student to leave the course - All information about the course is lost.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

example 2:

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

Deleting student S3 will permanently delete the course B.

Insert Anomalies

An **Insert Anomaly** occurs when certain attributes cannot be inserted into the database without the presence of other attributes.

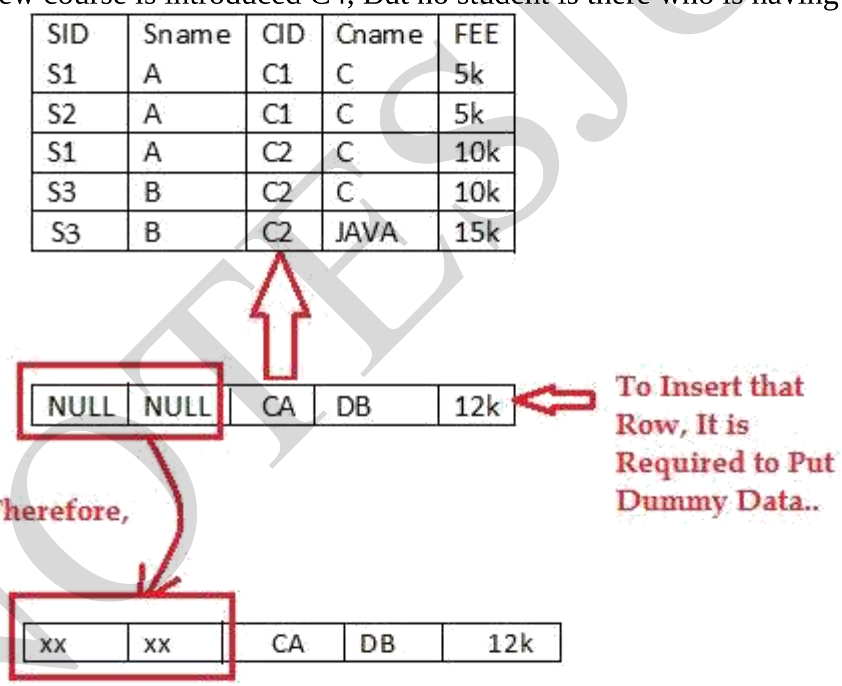
example 1:

we can't add a new course unless we have at least one student enrolled on the course.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

example 2:

New course is introduced C4, But no student is there who is having C4 subject.



Terms related to normalization

Functional dependency:

It is an association between two attributes of the same relational database table. One of the attributes is called determinant and the other attribute is called determined. For each values of determinant there is associated only one value of the determined.

If A is determinant and B is determined then we say that A functionally determines B and is represented as A → B i.e. for each value of attribute A, there is exactly one value of attribute B. If value of A is repeating in tuples then value of B will also repeat. In our example, Employee Address has a functional dependency on Employee ID, because a particular Employee ID value corresponds to one and only one Employee Address value. (Note that the reverse need not be true: several employees could live at the same address and therefore one Employee Address value could correspond to more than one Employee ID. Employee ID is therefore not functionally dependent on Employee Address.) An attribute may be functionally dependent either on a single attribute or on a combination of attributes.

Employee Table

Employee ID	Employee Name	Employee Address	Salary
1	Ankit	Delhi	10000
2	Amit	Mumbai	20000
3	Amit	Delhi	10000

A **functional dependency** occurs when the value of one (a set of) attribute(s) determines the value of a second (set of) attribute(s) in the same table:

Functional Dependencies are (EmployeeID is unique):

a) Employee ID → Employee Name
b) Employee ID → Employee Address
c) Employee ID → Salary

Student Table

RollNo	Name	Address	Semester	Course	Batch
12	Rita	Delhi	First	BCA	2006
23	Amit	Mumbai	Second	BBA	2007
33	Ankit	Delhi	First	BBA	2008
34	Amit	Pune	Second	MCA	2006

Functional Dependencies are (RollNo is unique):

a) RollNo → Name
b) RollNo → Address
c) RollNo → Course
d) (RollNo, Course) → Semester

e) (RollNo, Course, Semester) → Batch

S.No	P.No	Quantity
1	10	10

Types of functional dependency

a) Trivial functional dependency: A trivial functional dependency is a functional dependency of an attribute on a superset of itself.
E.g. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.

b) Full functional dependency: An attribute is fully functionally dependent on a set of attributes X if it is:

- functionally dependent on X, and
- Not functionally dependent on any proper subset of X.

E.g.{Employee Address} has a functional dependency on {Employee ID, EmployeeName}, but not a full functional dependency, because it is also dependent on {Employee ID}.

X → Y

X is determinant
Y is determined

Example:

1. Roll no. determines student name

   roll no → Student name

2. Social Security Number determines employee name

   SSN → ENAME

3. Project Number determines project name and location

   PNUMBER → {PNAME, PLOCATION}

4. Employee SSN and project number determines the hours per week that the employee works on the project

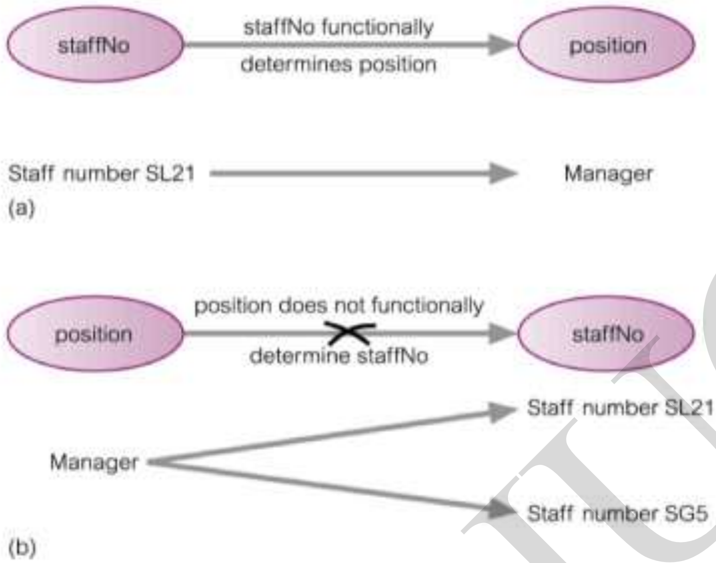
{SSN, PNUMBER} → HOURS

5. If, ExtendedPrice = Quantity X UnitPrice , then we can say quantity and unit price determines ExtendedPrice.

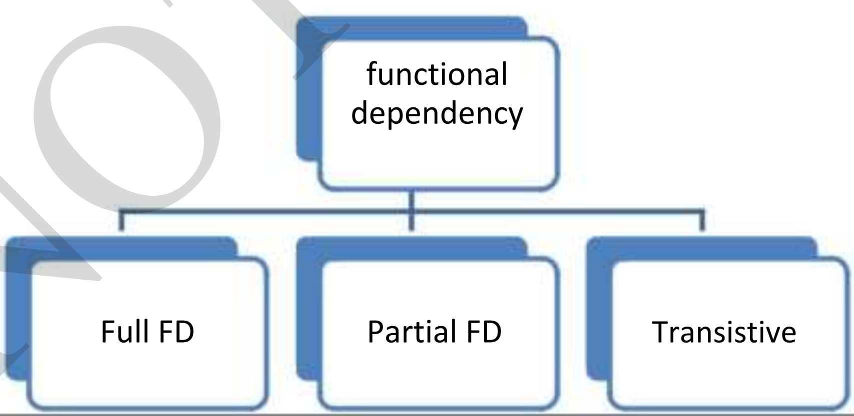
(Quantity, UnitPrice) → ExtendedPrice

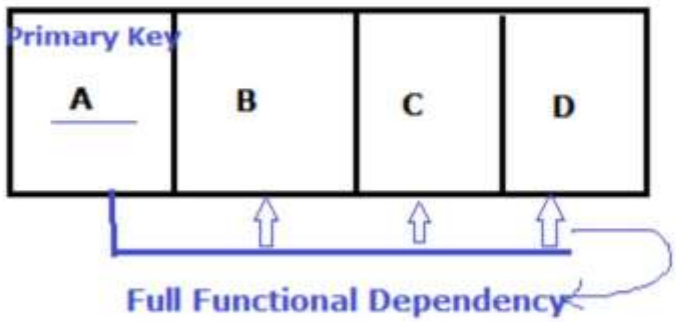
One of the attributes is called as the **determinant** and the other is called as the **determined**.

- The attribute (or attributes) that we use as the starting point (the variable on the left side of the equation) is called a *determinant*



Types of Functional dependency



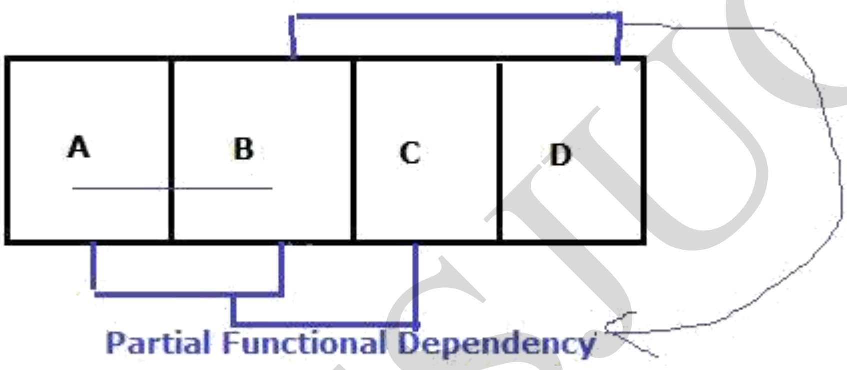


$A \rightarrow B$

$A \rightarrow C$

$A \rightarrow D$

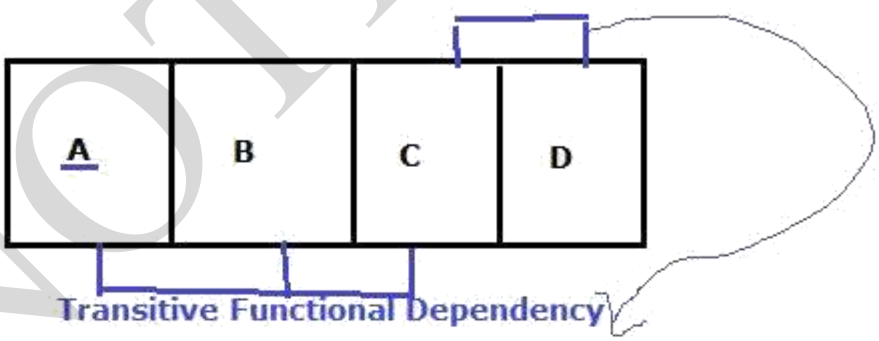
every non key attribute is dependent on primary key(A).



primary key: (A,B)

$A,B \rightarrow C$

$B \rightarrow D$ (part of primary key determines a non key attribute)



Primary key: A

$A \rightarrow B$

$A \rightarrow C$

C $\rightarrow$ D (non-key attribute determines another non key attribute)

Full FD:

Every non key attribute is dependent on primary key

key $\rightarrow$ non key attribute

If any attribute is removed from LHS, the FD does not hold any more.

i.e. $\{X - (A)\} \nrightarrow Y$

Example 1:

Rno	Marks	Name	course

Rno marks
Rno name
Rno course

example 2:

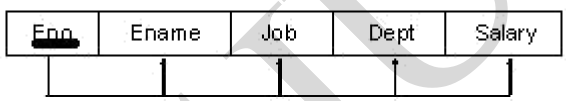


Fig. 6.2 Functional dependency of relation EMPLOYEE

example 3:

Shipment Table

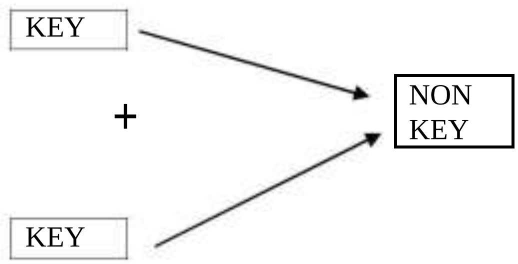
S.No	P.No	Quantity
1	10	10
1	12	12
3	10	15

In the above table
{S.No, P.No} → Quantity is Full Functional Dependency.

determinant determined

Partial FD:

Some non key attribute is dependent on primary key and some are dependent on other non key attribute i.e there are more than 1 PK in a relation



In this case,

{X –(A) } $\longrightarrow$ Y , holds

Example 1 :

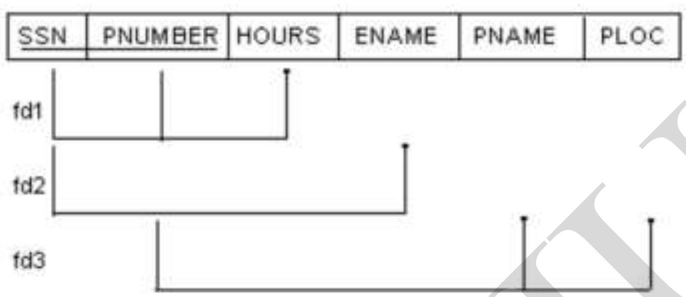


Fig. 7.3 A relation schema of Emp-Project relation

In the above figure, the dependency
{SSN, PNUMBER} $\rightarrow$ ENAME is partial because SSN $\rightarrow$ ENAME holds

FFD VS PFD

Consider following relationship Emp_proj with composite key(eno,pno)

<u>eno</u>	<u>pno</u>	hours	ename	pname	ploc

eno,pno $\rightarrow$ hours (full FD)

If we remove eno or pno from the above FD, then it will not hold.

i.e.

eno $\rightarrow$ hours
pno $\rightarrow$ hours } $\longrightarrow$ Not Valid

so, {X –(A) } → Y , is true

eno,pno → ename
eno → ename
pno → ename
} → VALID

so, {X –(A) } → Y , is true

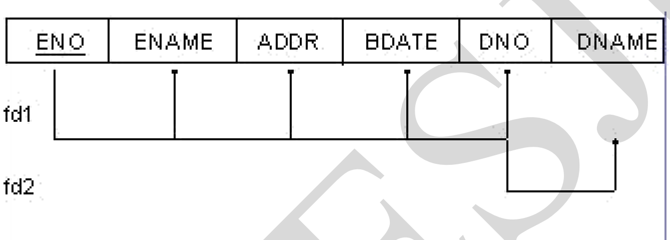
i.e partial FD

Transitive FD: Non key attributes are dependent only on non-key attributes.

Transitive dependencies occur when a *determinant* affects the values of more than one business object.

NON KEY → NON KEY

Example 1 :



In the above figure, DNO is dependent on key attribute ENO and DNAME is dependent on DNO which we can denote as,

ENO ->{ENAME, ADDR, BDATE, DNO}
DNO -> DNAME

From the above dependencies we can say that DNAME is indirectly related to key attribute ENO. So, DNAME is transitively dependent on ENO.

Closure of set of dependency

The set if all dependencies that include F (Functional dependency) as well as all dependencies that can be inferred from F is called the closure of F. It is denoted by F^+.

E.g. Dep_ID → Manager_Eno
Manager_Eno → Manager_PhNo

Then the two dependencies together implies that Dep_ID → Manager_PhNo. This is an inferred F.D and need not be explicitly stated.

i.e. F^+ = { Dep_ID → Manager_Eno, Manager_Eno → Manager_PhNo and Dep_ID → Manager_PhNo }

Armstrong's Inference Rules

a) Reflexivity Rule: If X is as set of attributes and Y is a subset of X, then X→Y holds true. It is also known as Trivial functional dependency:

E.g. {Employee ID, Employee Address} → {Employee Address} {D_No, D_Location} → {D_Location }

b) Augmentation Rule: If X→Y holds and W is a set of attributes, then WX→WY holds true.
E.g. RollNo → Name
{Class, Marks} are set of attributes W then
{RollNo, Class, Marks} → {Name, Class, Marks}

c) Transitivity Rule: If X→Y and Y→Z holds then X→Z holds true.
E.g. RollNo → City and City → Status implies RollNo → Status

d) Union Rule: If X→Y and X→Z holds then X→YZ holds true
E.g. RollNo → Name and RollNo → PhoneNo implies RollNo → (Name, PhoneNo)

e) Decomposition Rule: If X→YZ holds then X→Y and X→Z holds true.
E.g. RollNo → (Name, PhoneNo) implies RollNo → Name and RollNo → PhoneNo

f) Pseudotransitivity Rule: If X→Y and WY→Z holds then WX→Z holds true.

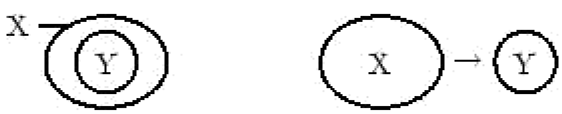
E.g. {Class, Marks} are set of attributes W then
RollNo → Name
{Class, Marks, Name} → Address implies {Class, Marks, RollNo} → Address

Specify rules for reasoning about dependency functions:

1. **Reflexive rule:**

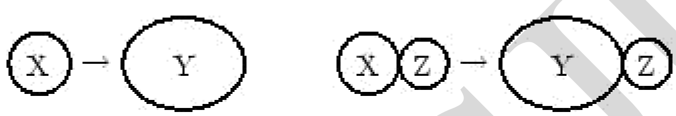
If A is a set of attributes such that B is a subset of A,

then A → B.



2. **Augmentation Rule**

if A → B, then AC → BC holds true.



3. **Transitive rule:**

If A → B & B → C, then A → C holds true.

Armstrong inference rules are

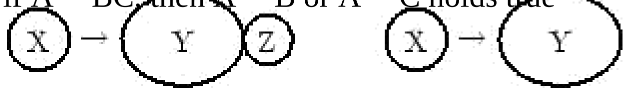
a. Sound: Produce only functional dependencies belonging to the closure
b. Complete: Produce all the functional dependencies in the closure

Additional Inference Rules

Armstrong's inference rules may be augmented with the following ones, without affecting their power.

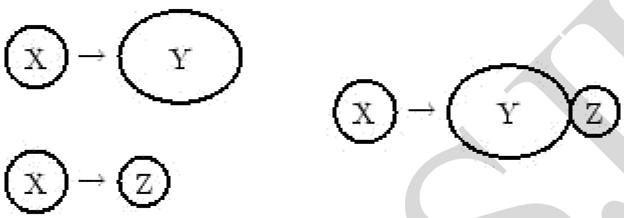
4. **Decomposition**:

if $A \rightarrow BC$, then $A \rightarrow B$ or $A \rightarrow C$ holds true



5. **Union:**

If, $A \rightarrow B$ & $A \rightarrow C$ then $A \rightarrow BC$ holds true

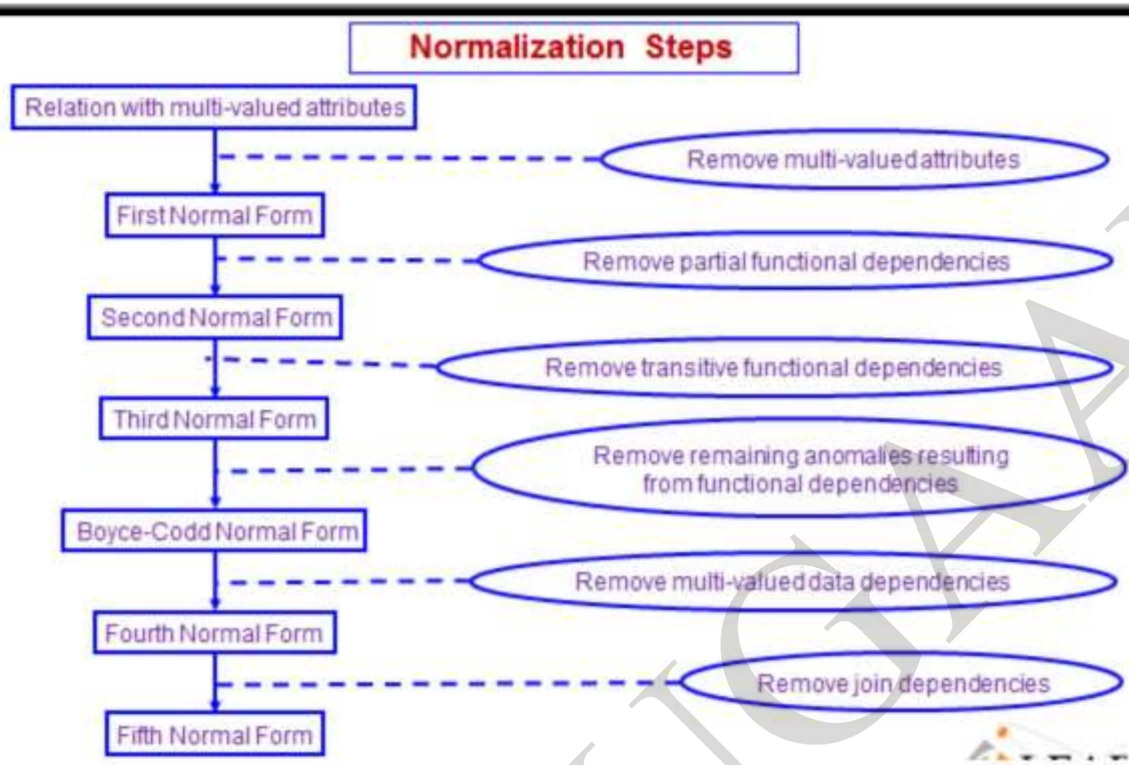


6. **Pseudotransitive rule**

if $A \rightarrow B$, $\gamma B \rightarrow C$, then $\gamma A \rightarrow C$ holds true.

Normalization Process

The normalization process was first proposed by Codd (1972), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form.



First normal form:

definition :

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values.

Achieving 1NF: Remove the repeating groups & ensure that all the entries of the resulting table have at most a single value & a key must be identified.

Which type of table need to be normalized:
Tables with multivalued entries, create an un-normalized table.

<u>Example:</u>
Following relation is in un-normalized form (UNF) as here are multiple occurrences of rows under each key Emp-Id.

pno	pname	eno	ename	job	salary	no.of days
1	insurance	101,103,105,106	rahul, amit,rakesh,rohit	a/cs,mgr, clerk, clerk	1000, 4000,500,500	30.15.30.25
2.	health	101,103,107,109	rahul, amit, ruche,suruchi	a/cs, mgr,hr,clerk	1000,4000,500,500	30,10,30,30

approaches to 1NF

a. Flattening of table
b. Decompose the table

Flattening of table

Step 1. In this method, we remove the repeating groups by filling the missing entries of each incomplete row of the table.

The above table in UNF can be processed to create the following table in 1NF.

pno	pname	eno	ename	job	salary	no.of days
1	insurance	101	rahul	a/cs	1000	30
1	insurance	103	amit	manager	4000	15
1	insurance	105	rakesh	clerk	500	30
1	insurance	106	rohit	clerk	500	25
2	health	101	rahul	a/cs	1000	30
2	health	103	amit	manager	4000	10
2	health	107	ruchi	HR	500	30
2	health	109	suruchi	clerk	500	30

Step 2.

Identify the PK of this table now.
since pno is no longer uniquely identifying any row , we take a composite PK i.e (pno, eno)

Decompose the table

in this approach , the table is broken down into multiple relation having the following structures.
1st table: PK + non-repeating attributes (eno, ename, job, salary, no. of days)
2nd table: PK + Repeating attributes(pno, pname)

1st table : emp_details

eno	ename	job	salary	no. of days

2nd table: project_details

pno	pname

Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF **<u>and each and every attribute fully depends on the primary key of the relation</u>**. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

2NF test:

1. If the PK contains only a single attribute then relation is in 2NF. No need for any check of 2NF.
2. If **PK contains more than one attribute**, then the relation must check if there is any partial dependency and then remove it.

Let us explain.

consider the company table

pno	pname	eno	ename	job	salary	no.of days

firstly check what type of dependencies exist:

pno,eno → pname, ename, job,salary,noofdays

pno → pname

eno → ename,job,salary

partial dependency

steps to 2NF

1. Write LHS of each partial dependency on a separate line and also write the original PK.

a. Pno
b. Eno
c. Pno,eno

2. Divide the relation into 3 tables and add dependent attributes

a. Project(pno,pname)
b. Employee(eno,ename,job, salary)

c. Wotks(pno,eno,noofdays)

now the relation is in 2NF.

Third Normal form(3NF)

For a table in third normal form

- It should already be in Second Normal Form.
- There should be no transitive dependency, i.e. we shouldn't have any non-key column depending on any other non-key column.

Again we need to make sure that the non-key columns depend upon the primary key and not on any other non-key column.

relation in 2NF

Employee(eno,ename,job, salary)

the non prime attribute salary is transitively dependent on PK as follows

eno → job

job → salary

which implies,

eno → salary

hence the relation is in 2NF but not in 3NF.

steps to convert to 3NF

1. Create a new relation for FD that cause transitive dependency.

In this case job → salary is the problem that causes transitive dependency.
So, decompose it & create a new relation

employee (original table to be converted in 3NF)

eno	ename	job	salary

FD1: eno → ename,job,salary

FD2: job → salary

step 2:

create separate table for each FD

emp

eno	ename	job

eno → ename   eno → job

job_salary

job	salary

job → salary

3. **Boyce-Code Normal Form (BCNF)/ 3.5 NF**

Boyce-Codd normal form (BCNF)

A relation is in BCNF, if and only if, every determinant is a candidate key.

The difference between 3NF and BCNF is that for a functional dependency A → B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key, whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key.

A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every dependency [determinant] is a candidate key. A relation in 3NF is not always in BCNF.

Example 2:Normalization