DBMS Basic Concepts

BY: RICHA JAIN

DBMS

- Database Management System
- Term Database requires understanding of data and information
- **Data:** It can be anything like name, place or number, etc. Data usually refers to raw data, or unprocessed data.

- **Information:** It is organized or classified data so that it has some meaningful values to the receiver.
  - Information is the processed data on which decisions and actions are based.

Difference between Data and Information?

Data	Information
Data is raw facts and figures	Information is a processed form of data
For example: 12 is data	For example: When 12 is stored in row column form as shown it is information. Age 12
Data are atomic level pieces of information	Information is a collection of data
Data does not help in decision making	Information helps in decision making

Database

- A database is a shared collection of logically related data designed to meet the information needs of an organization
- The related information when placed is an organized form makes a database.
- The organization of data/information is necessary because unorganized information has no meaning.

Purpose of DBMS

An example

- University Database:

Data about students, faculty, courses, research-laboratories, course registration/enrollment etc. Reflects the state of affairs of the academic aspects of the university.

***Purpose*:** To keep an accurate track of the academic activities of the university.

Purpose of DBMS

Before DBMSs were introduced, organizations usually stored information in file processing system which has a number of disadvantages:

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation
- Integrity problems
- Atomicity problems
- Concurrent-access anomalies
- Security problems

Database Management System

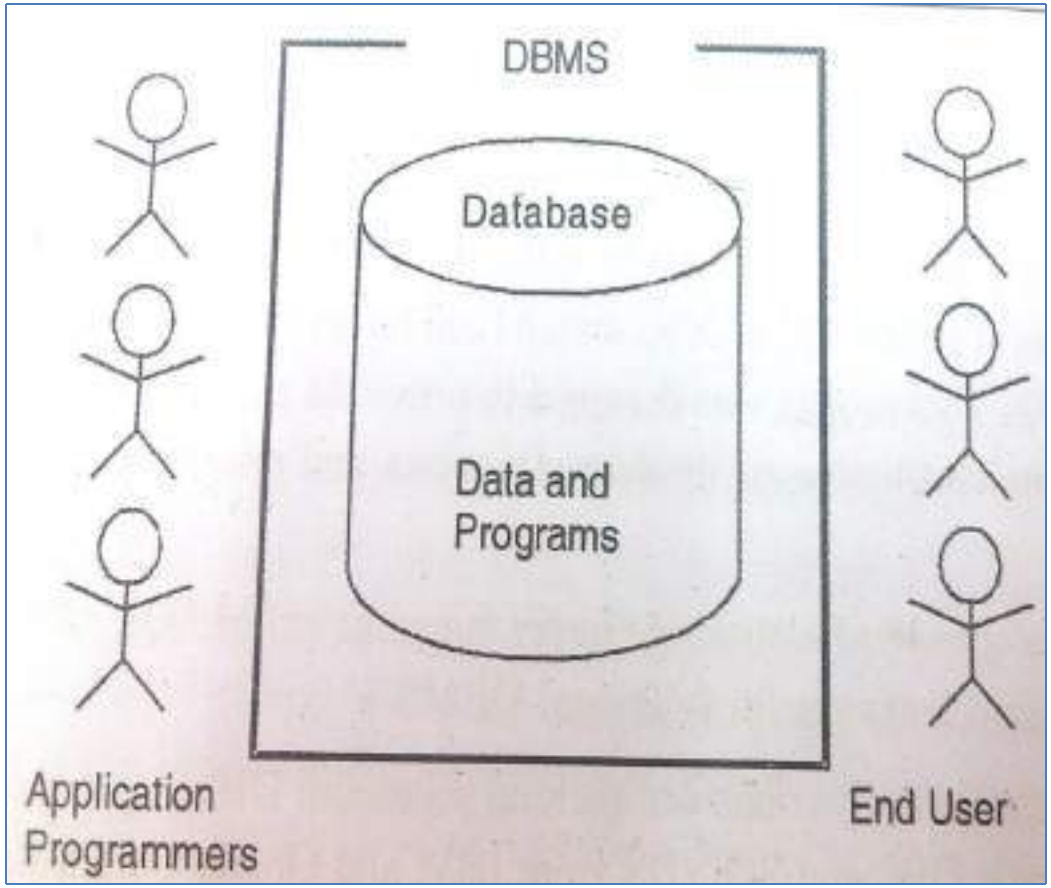
- **DBMS** A database management system is the software system that allows users to define, create and maintain a database and provides controlled access to the data.
- A database management system (DBMS) is basically a collection of programs that enables users to store, modify, and extract information from a database as per the requirements.

Operations on databases

- To add new information
- To view or retrieve the stored information
- To modify or edit the existing
- To remove or delete the unwanted information
- Arranging the information in a desired order etc.

Components of Database

- Five major components in database system environment:
  - Hardware
  - Software
  - Data
  - Users
  - Procedures



Components of Database System

- **Hardware:** It is the actual computer system used for keeping and accessing the database. DBMS hardware consists of secondary storage devices like hard disks.
- **Software:** It is the actual DBMS. Between the physical database itself and the users of system is a layer of software, called DBMS.
- **Data:** Data acts as the bridge between the machine components and user components.

Components of Database System

- **Users:** There are number of users who can access or retrieve data on demand using the applications and the interfaces provided by DBMS. The users can be:
  - Naïve users
  - Online users
  - Application Programmers
  - Sophisticated Users
  - Data base Administrator (DBA)

Components of Database System

- **Procedures:** It refers to the instructions and rules that govern the design and the use of the database. The users of the system and the staff that manage the database requires documented procedures on how to use or run the system.

Applications of DBMS

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Data models, Schemas, and Instances

- **Data model:-**A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Schema:-** The overall description of the database is called the Database Schema.
  - A schema is defined as an outline or a plan that describes the records and relationships existing at the particular level.
- **Instance:-** Data in the database at a particular moment in time.

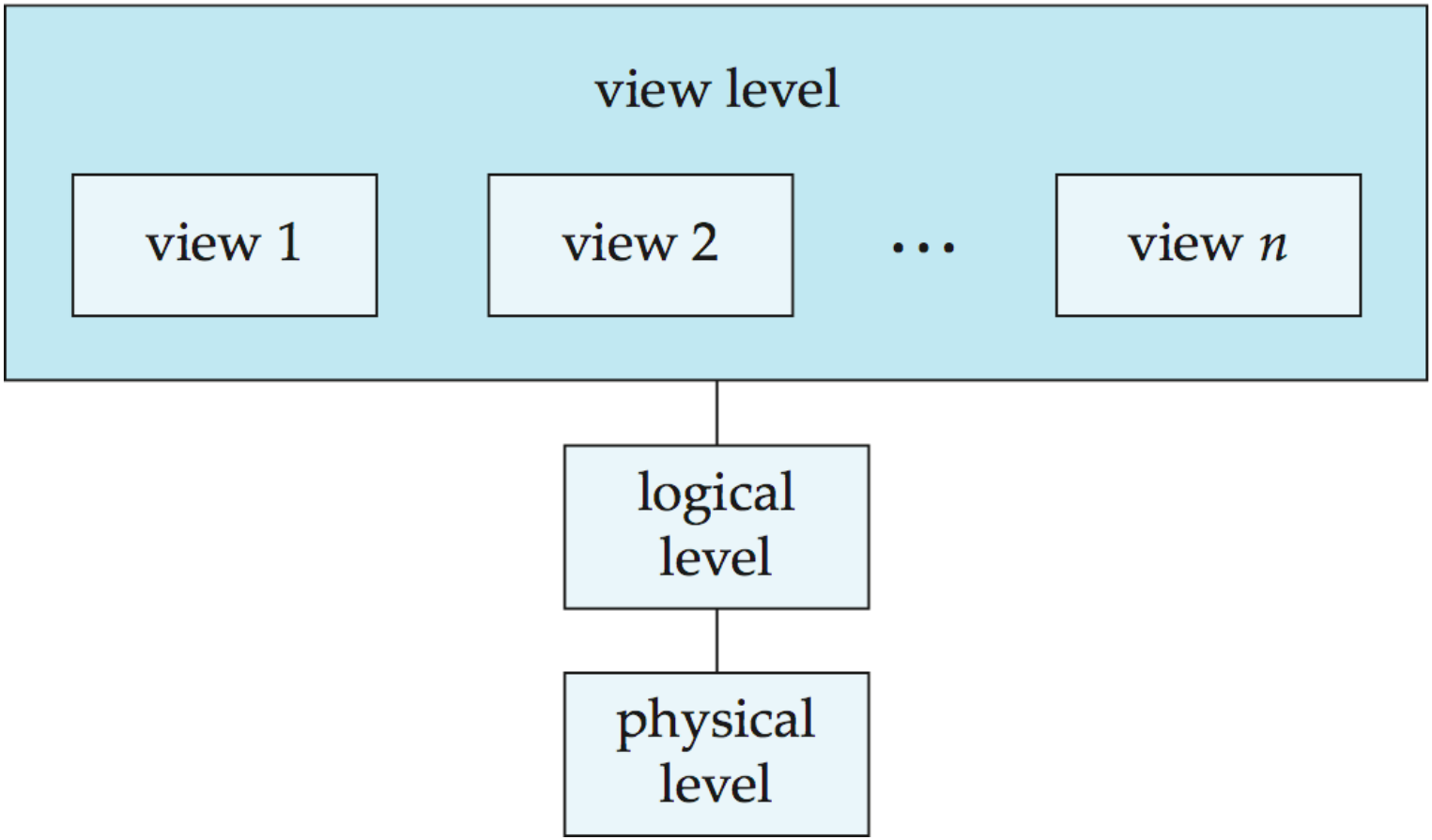
Data abstraction

- A major purpose of database system is to provide user with an abstract view of data. That is, system hides certain details of how the data are stored and maintained.

Levels of Abstraction(view of data)

- **Physical level:** describes how a record (e.g., customer) is stored.
- **Logical level:** describes what data stored in database, and the relationships among the data. DBA, who decides what information to keep in the database, use the logical level of abstraction.
- **View level:** describe only part of database. application programs hide details of data types. Complexity remain due to variety of information stored. Views can also hide information (such as an employee's salary) for security purposes.

View of Data



An architecture for a database system

Database Languages

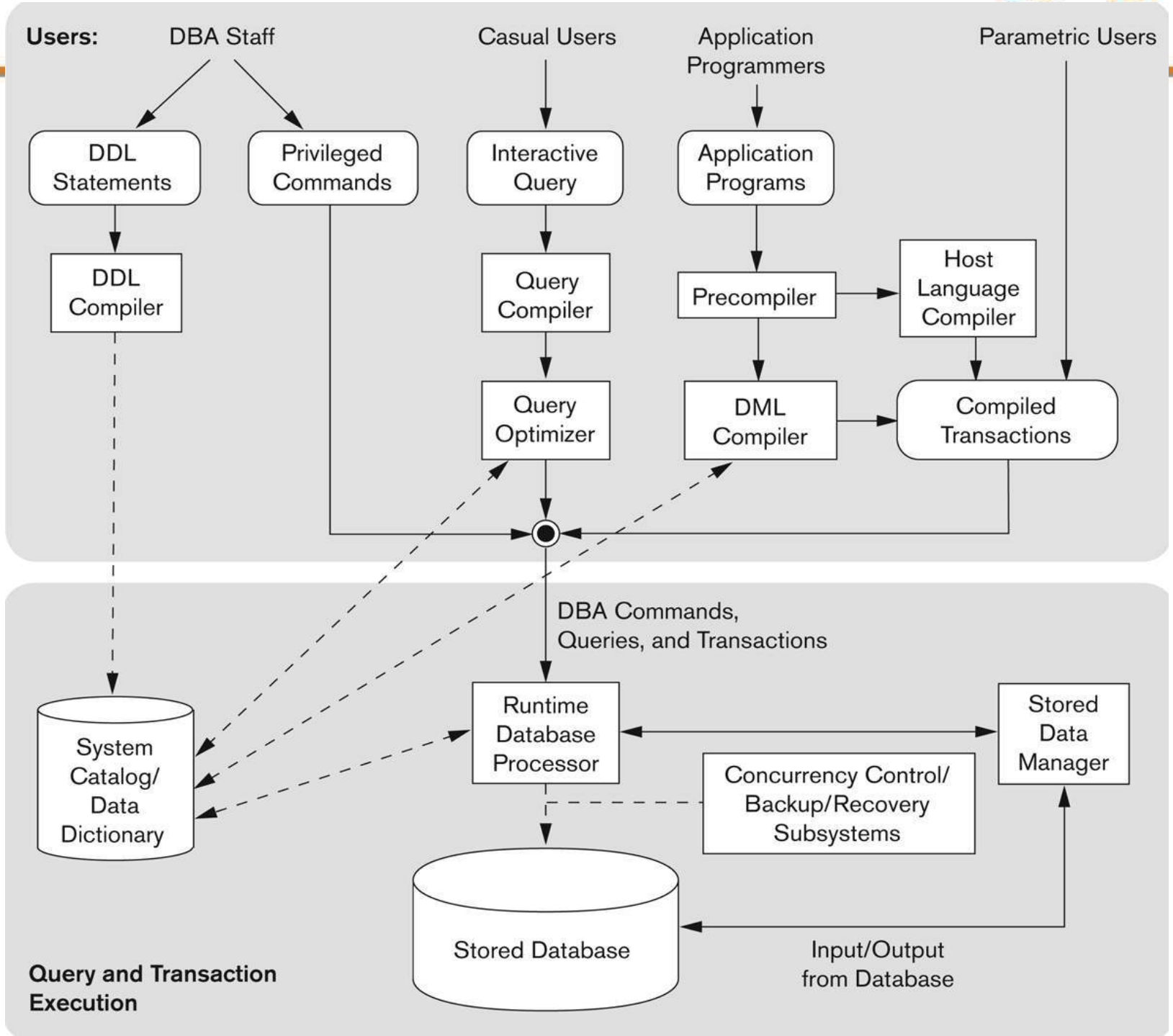
Database languages are used to create and maintain database on computer.

- **Data Definition Language(DDL):** It is a language that allows user to define data and their relationship to other types of data.
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE
  - RENAME
- **Data Manipulation Language(DML):**It provides a set of operations to support the basic data manipulation operations on the data held in databases. It allows user to insert, update, delete and retrieve data from the database.
  - DELETE
  - INSERT
  - SELECT
  - UPDATE

Database Languages

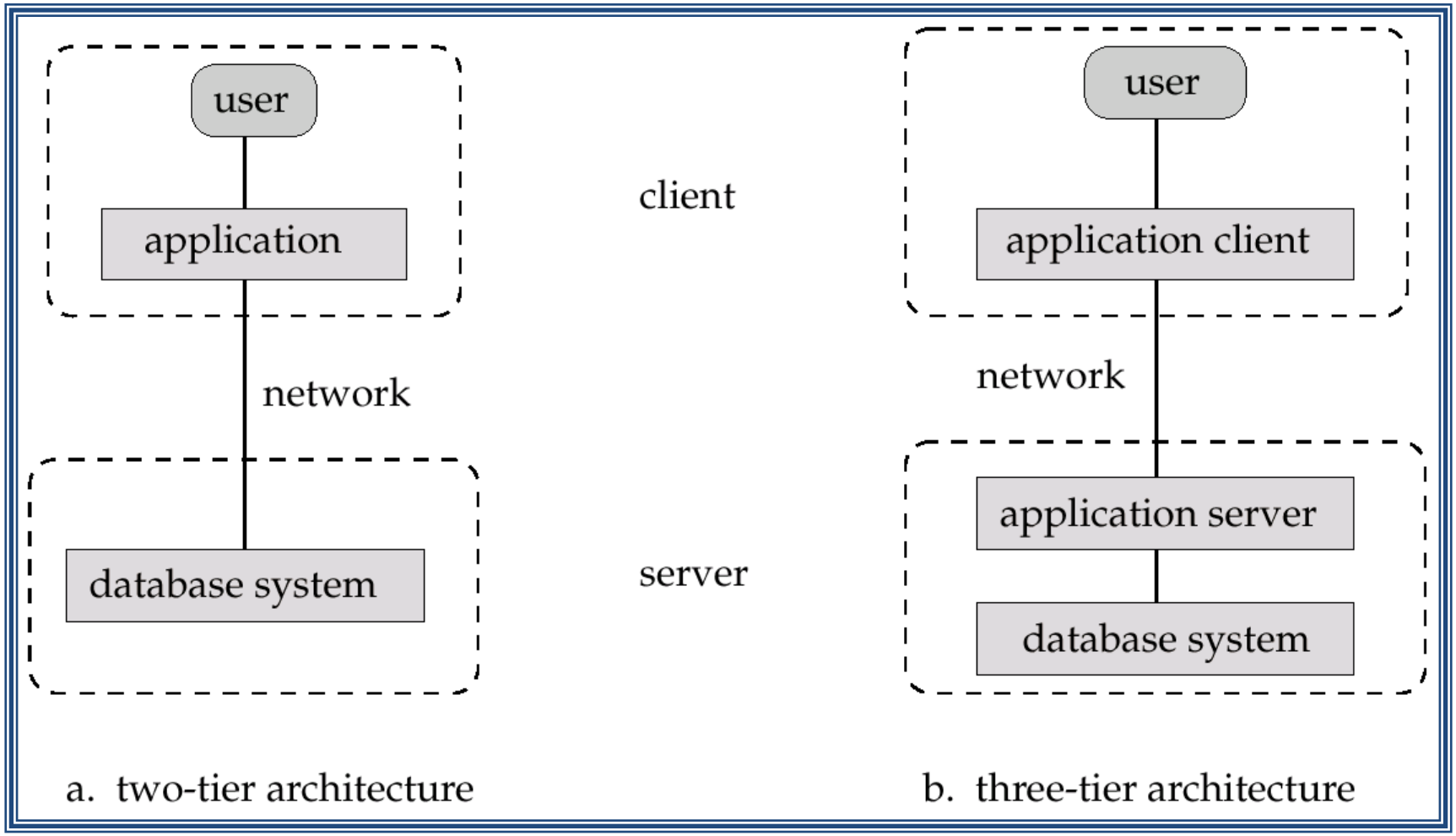
- **Data Control Language(DCL):** DCL statements control access to data and the database
  - GRANT
  - REVOKE
  - COMMENT
- **Transaction Control Language(TCL):** TCL statements manage the change made by DML statements, and group DML statements into transactions
  - COMMIT
  - ROLLBACK
  - SAVEPOINT
  - SET TRANSACTION

Structure and Components of DBMS

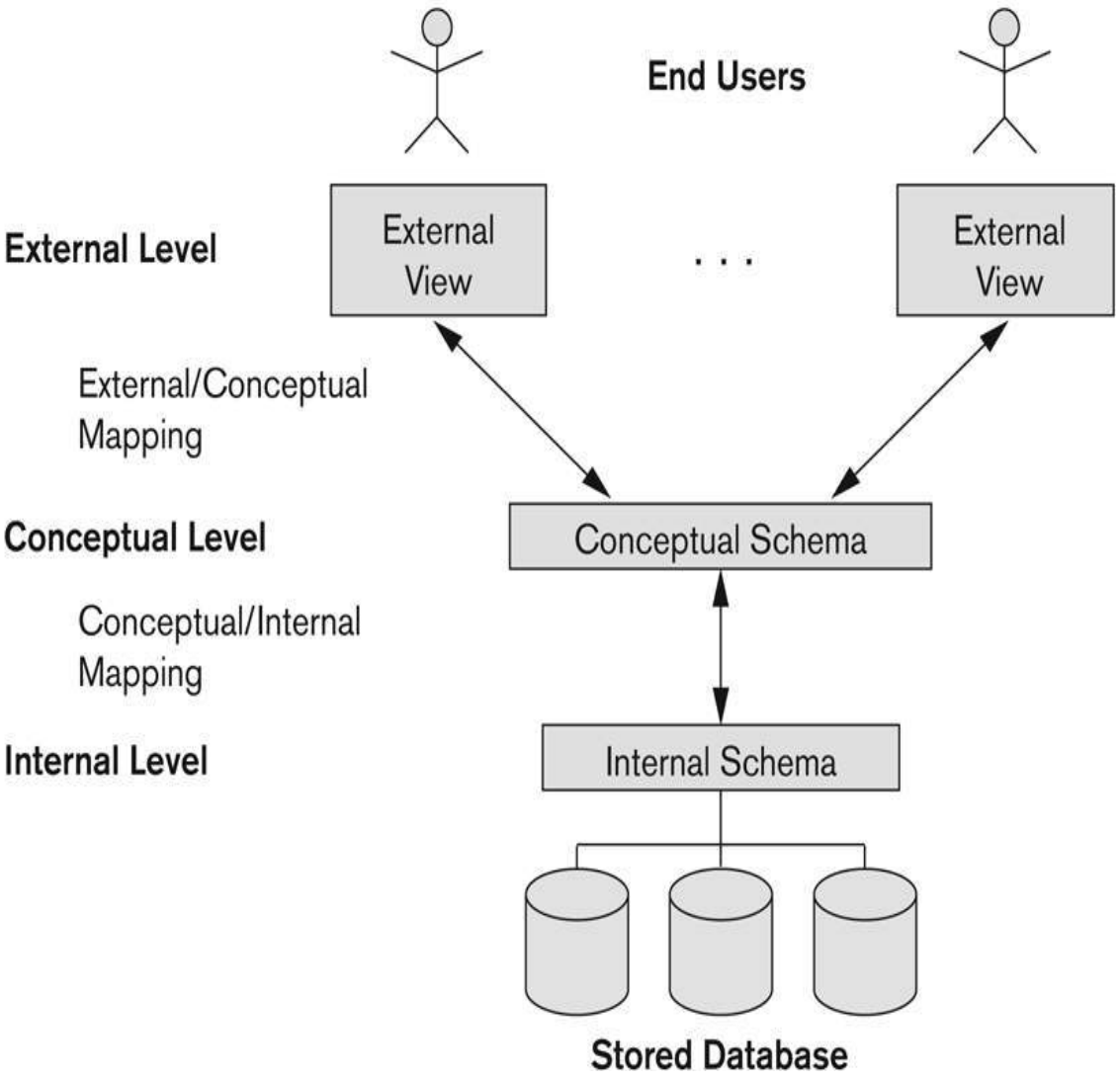

Users:
DBA Staff
Casual Users
Application Programmers
Parametric Users
DDL Statements
Privileged Commands
Interactive Query
Application Programs
DDL Compiler
Query Compiler
Precompiler
Host Language Compiler
Query Optimizer
DML Compiler
Compiled Transactions
DBA Commands, Queries, and Transactions
System Catalog/ Data Dictionary
Runtime Database Processor
Stored Data Manager
Concurrency Control/ Backup/Recovery Subsystems
Stored Database
Input/Output from Database
Query and Transaction Execution

Database Architecture

Two-tier and three-tier architecture



Three-tier architecture



- **External or View level:** It is the users' view of the database. This level describes that part of the database that is relevant to each user.
  - For example, one user may view dates in the form (day, month, year), while another may view dates as (year, month, day).
- **Conceptual or logical level:** It is the community view of the database. This level describes what data is stored in the database and the relationships among the data.
- It represents:
  - All entities, their attributes, and their relationships;
  - The constraints on the data;
  - Security and integrity information.
- **Internal or storage level:** It is the physical representation of the database on the computer. This level describes how the data is stored in the database.

Data Independence-Achievement of Layered Architecture of DBMS

- Two kinds of data independence:
  - Logical data independence
  - Physical data independence

Data Independence

- **Logical Data Independence**: The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence**: The capacity to change the internal schema without having to change the conceptual schema.

Data Independence

- The processes of transforming requests and results between the levels are called **mappings.**
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

Data Independence

- The processes of transforming requests and results between the levels are called **mappings.**
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

Data Models

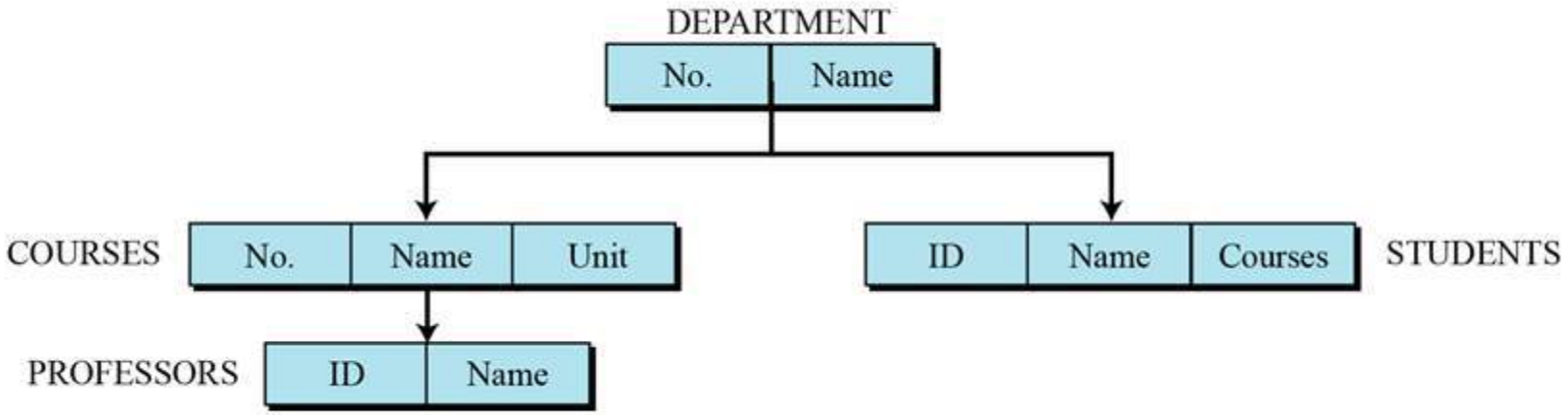
BY: Richa Jain

Data Model

- A database model defines the logical design of data. The model also describes the relationships between different parts of the data.
- Various types of data models are:
  - Object oriented model
  - Hierarchical data model
  - Relational data model
  - Network model

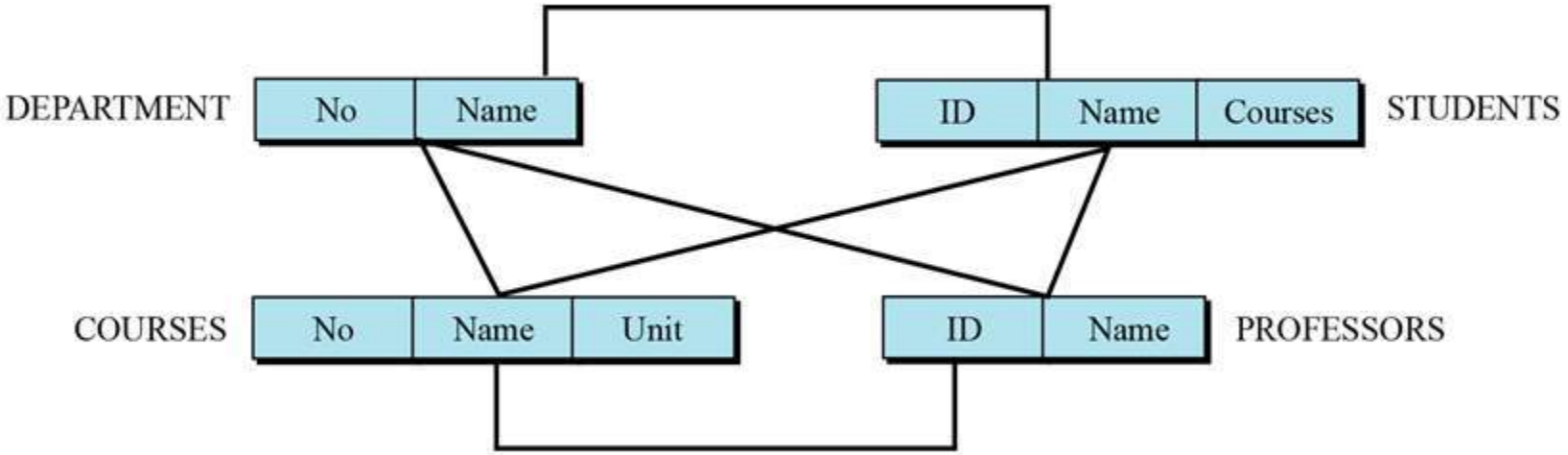
Data Models

- **Hierarchical data model:** This is called a **parent-child** relationship. In this model each entity has only one parent but several children. At the top of the hierarchy there is only one entity which is called **Root.**



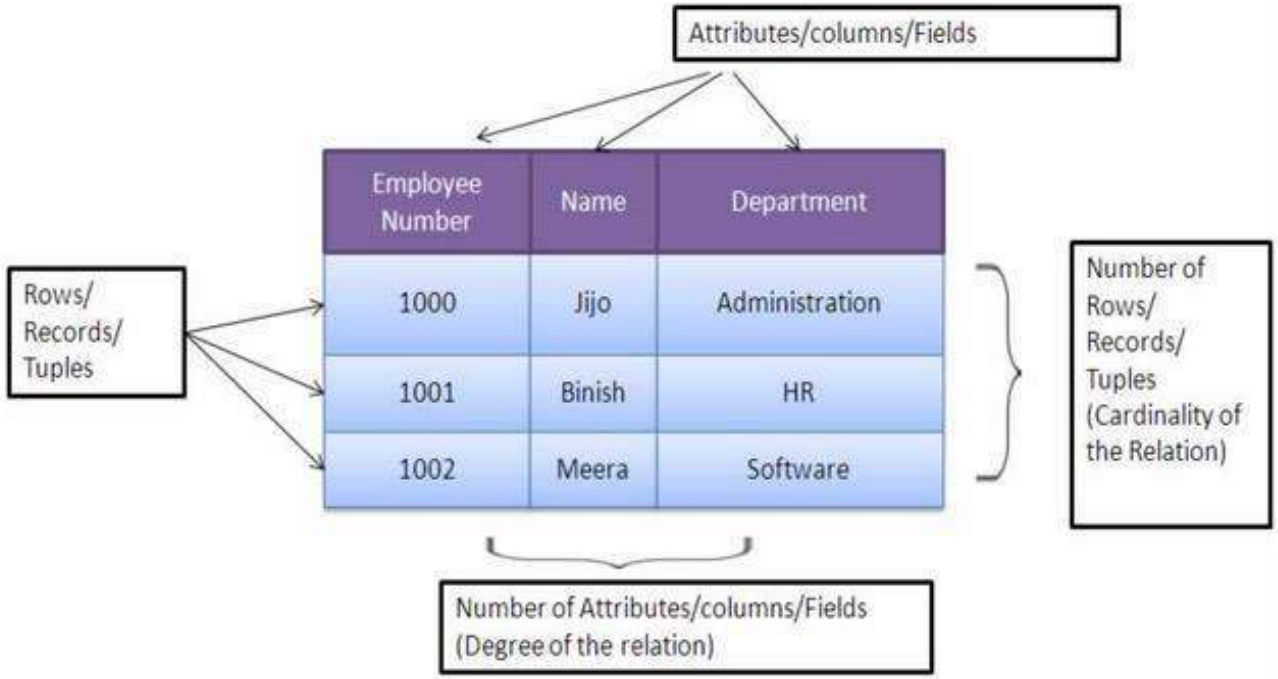
Data Models

- **Network model:** In a network DBMS every data item can be related to many others ones. The database structure is like a graph. This is similar to the hierarchical model and also provides a tree-like structure. However, a child is allowed to have more than one parent.



Data Models

- **Relational data model:** In relational data model, data exists in two dimensional tables known as relations. A relation (table) consists of unique attributes (columns) and tuples (rows).



Data Models

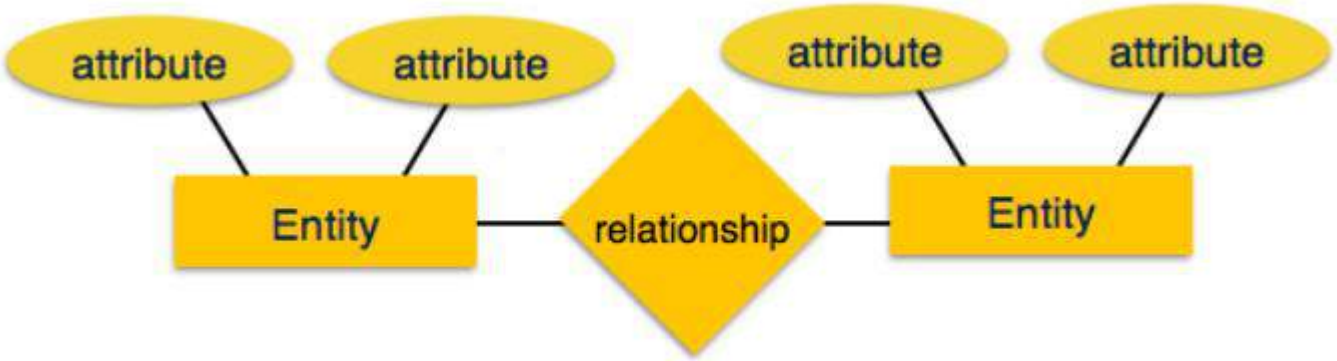
- **Object oriented model:** Object based data models use concepts such as entities, attributes, and relationships
- The **entity relational model(E-R model)** has emerged as one of the main techniques for modeling database design .

E-R Model

- ER Model is based on:

  Entities and their *attributes*

  Relationships among entities



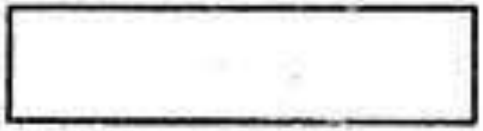
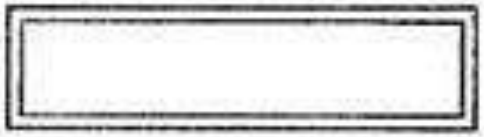
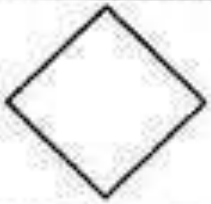
E-R Model components

- **Entity:** An entity in ER Model is real world entity, which has some properties called ***attributes***. Every attribute is defined by its set of values, called ***domain***. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age and class etc.
- An **entity set** is a collection of similar types of entities. For example, Students set may contain all the student of a school.

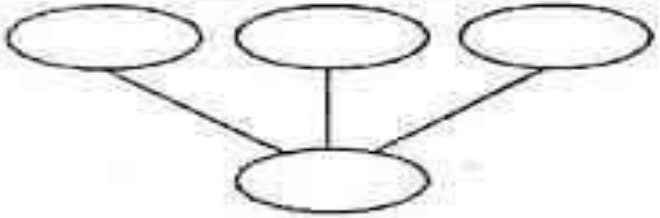
Attributes

- Attributes describe the properties of the entity of which they are associated.
- A particular instance of an attribute is a value. For example, "Ram" is one value of the attribute Name.
- We can classify attributes as following:
  - ¨ Simple
  - ¨ Composite
  - ¨ Single-valued
  - ¨ Multi-valued
  - ¨ Derived

E-R Model: Symbols and notations

Symbol	Meaning
	Entity Type
	Weak Entity Type
	Relationship Type
	Identifying Relationship Type
	Attribute

E-R Model: Symbols and notations

Symbol	Meaning
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E2 in R
	Cardinality Relation 1 : N for E1 : E2 in R

. Symbols used in E-R diagram

E-R Model

- **Relationship:** The association among entities is called relationship. For example, employee entity has relation works_at with department.
- Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Constraints

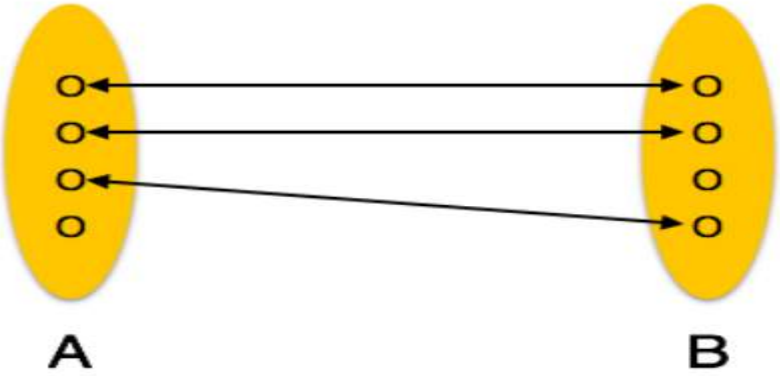
- Various types of constraints are:

1. Mapping cardinalities
2. Participation constraints
3. Keys

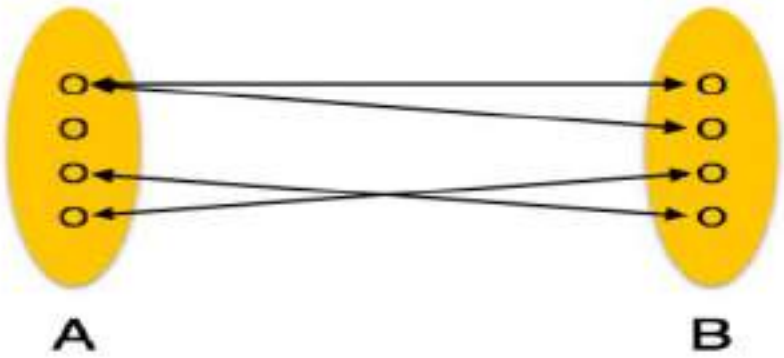
Mapping Cardinalities

- Mapping cardinality or cardinality ratios , express the number of entities to which another entity can be associated via a relationship set
- Mapping cardinalities:
  - one to one
  - one to many
  - many to one
  - many to many

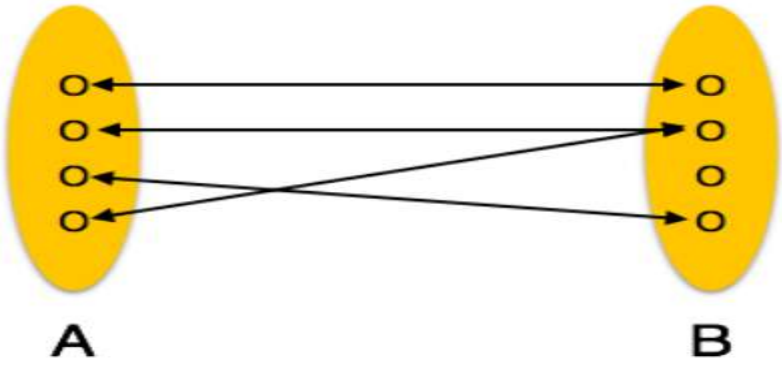
- **One-to-one:** one entity from entity set A can be associated with at most one entity of entity set B and vice versa.



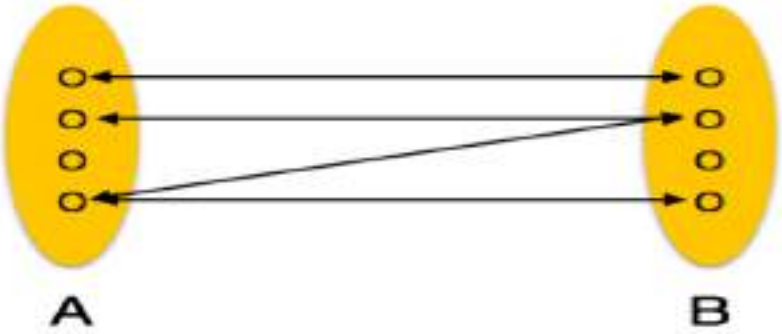
- **One-to-many:** One entity from entity set A can be associated more than one entities of entity set B but from entity set B one entity can be associated with at most one entity.



- **Many-to-one:** More than one entities from entity set A can be associated with at most one entity of entity set B but one entity from entity set B can be associated with more than one entity from entity set A.



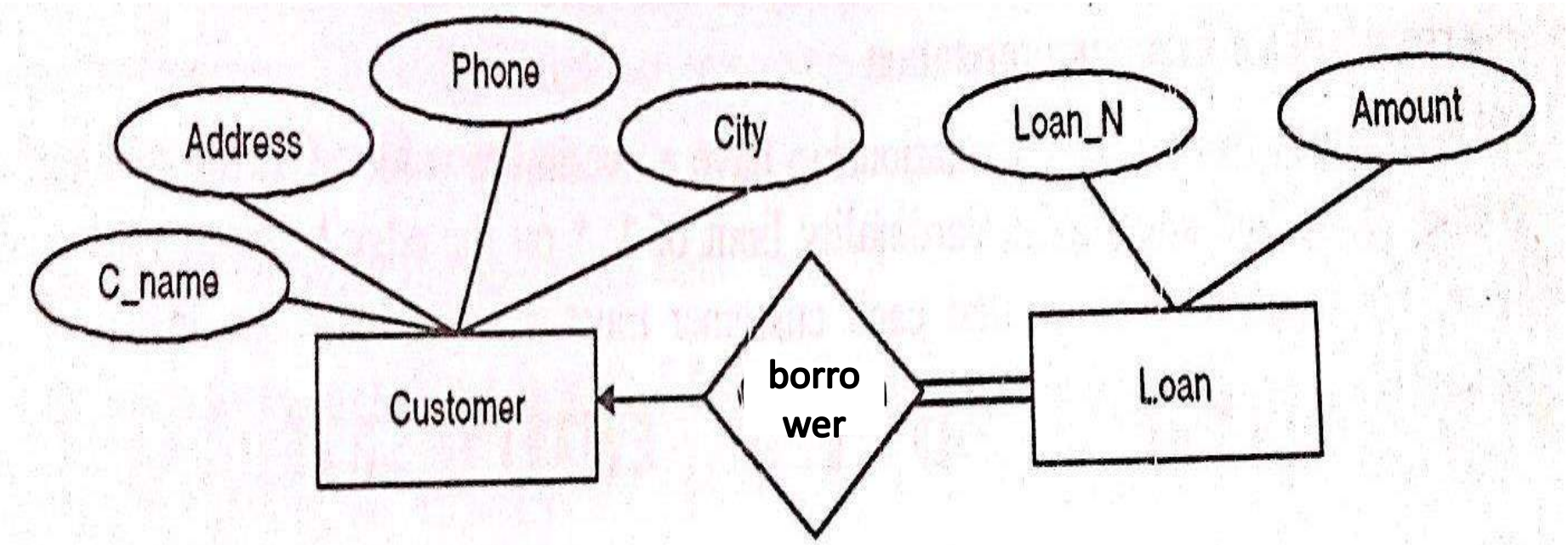
- **Many-to-many:** one entity from A can be associated with more than one entity from B and vice versa.



Participation Constraints

- The participation of an entity set E in a relationship set R is said to be **total** if every entity in E participates in at least one relationship in R.
- If only some entities in E participate in relationships in R, the participation is said to be **partial**.

Example



Total participation of Loan entity

Keys

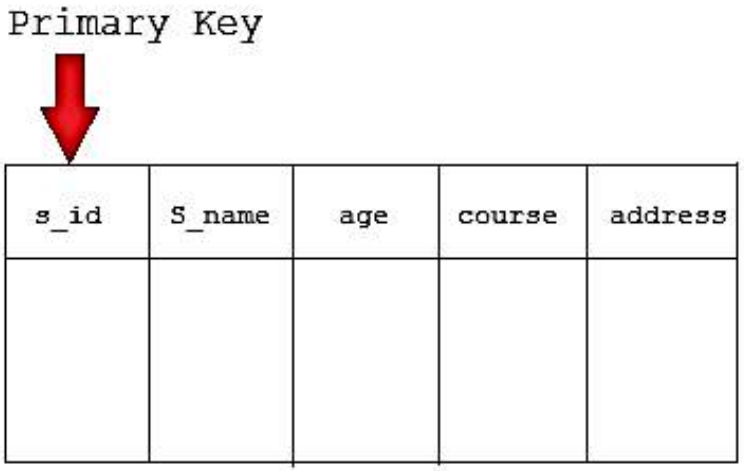
- The key is defined as the column or attribute of the database table.
- They are used to establish and identify relation between tables.
- They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

Types of Keys

1. **Superkey:** An attribute (or combination of attributes) that uniquely identifies each row in a table. It is a super set of candidate key.
2. **Candidate key :** An attribute (or set of attributes) that uniquely identifies a row. Let K be a set of attributes of relation R. Then K is a candidate key for R if it possess the following properties:
    1. Uniqueness – No legal value of R ever contains two distinct tuples with the same value of K
    2. Irreducibility- No proper subset of K has the uniqueness property

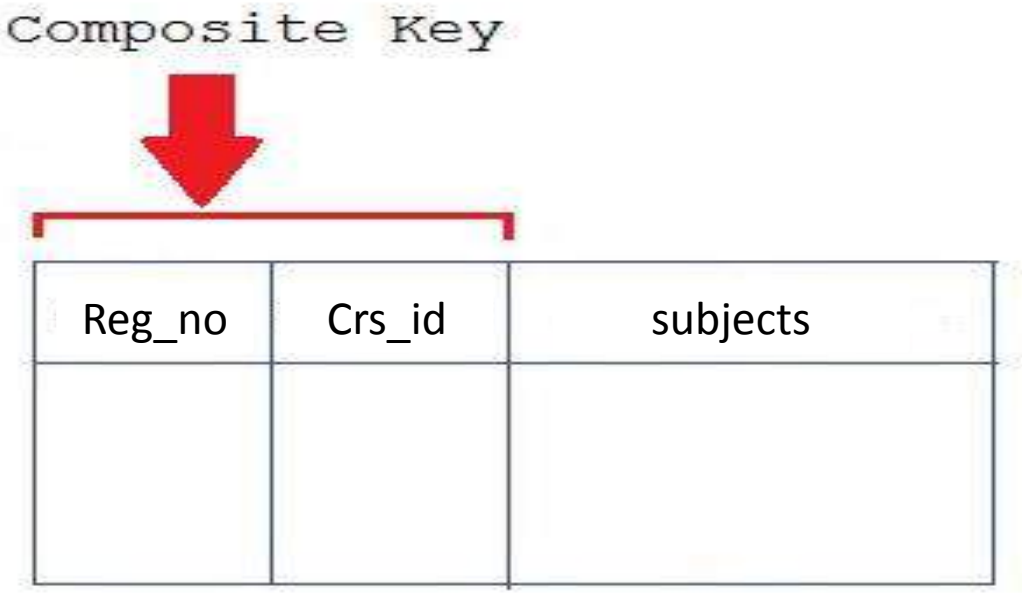
Types of Keys

3. Primary key : is the candidate key which is selected as the principal unique identifier. It is a key that uniquely identify each record in the table. Cannot contain null entries.



Types of Keys

4. Composite Key: Key that consist of two or more attributes that uniquely identifies an entity occurrence is called composite key.

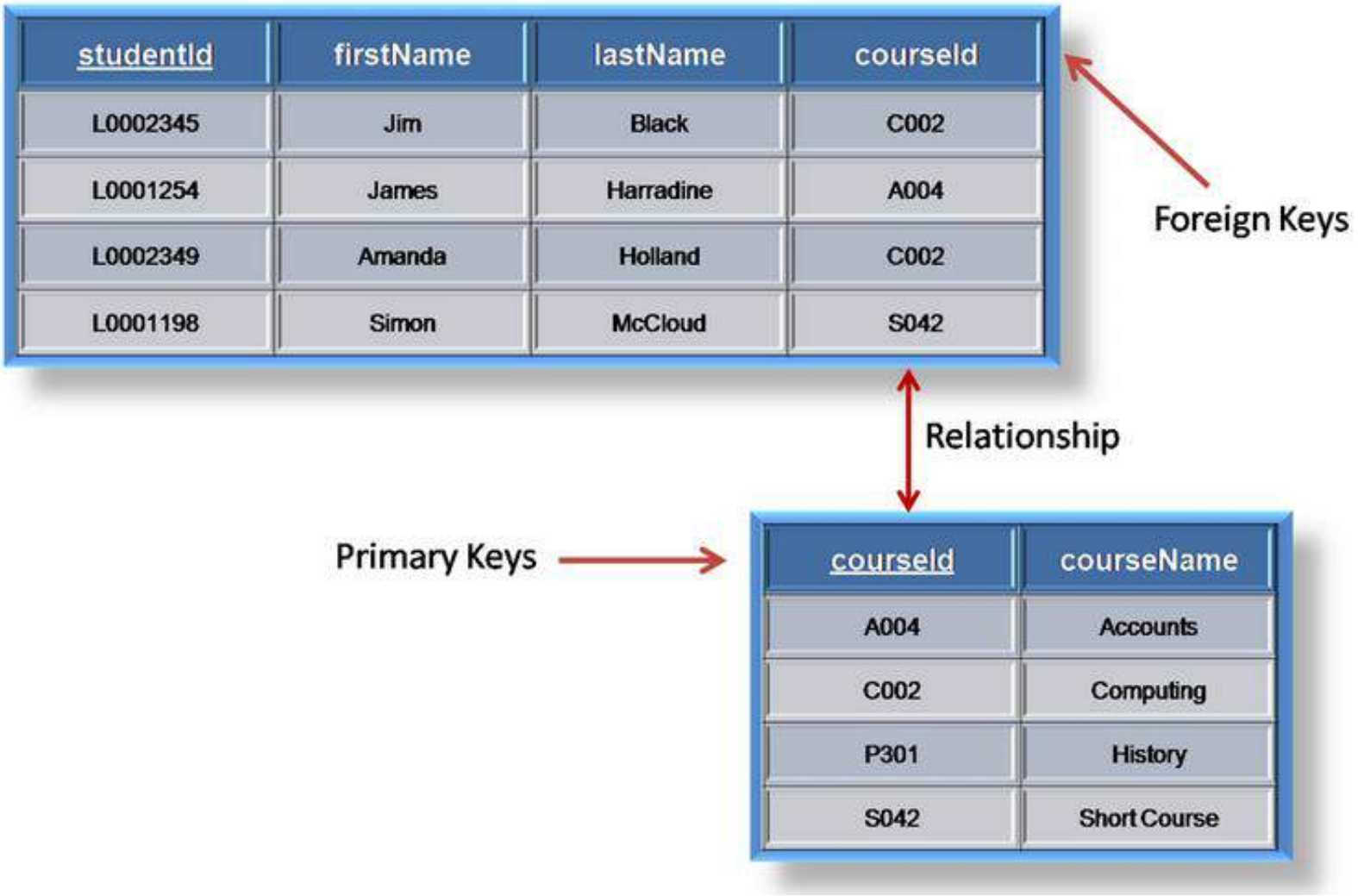


Types of Keys

5. Foreign Key: A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.

In other words, if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

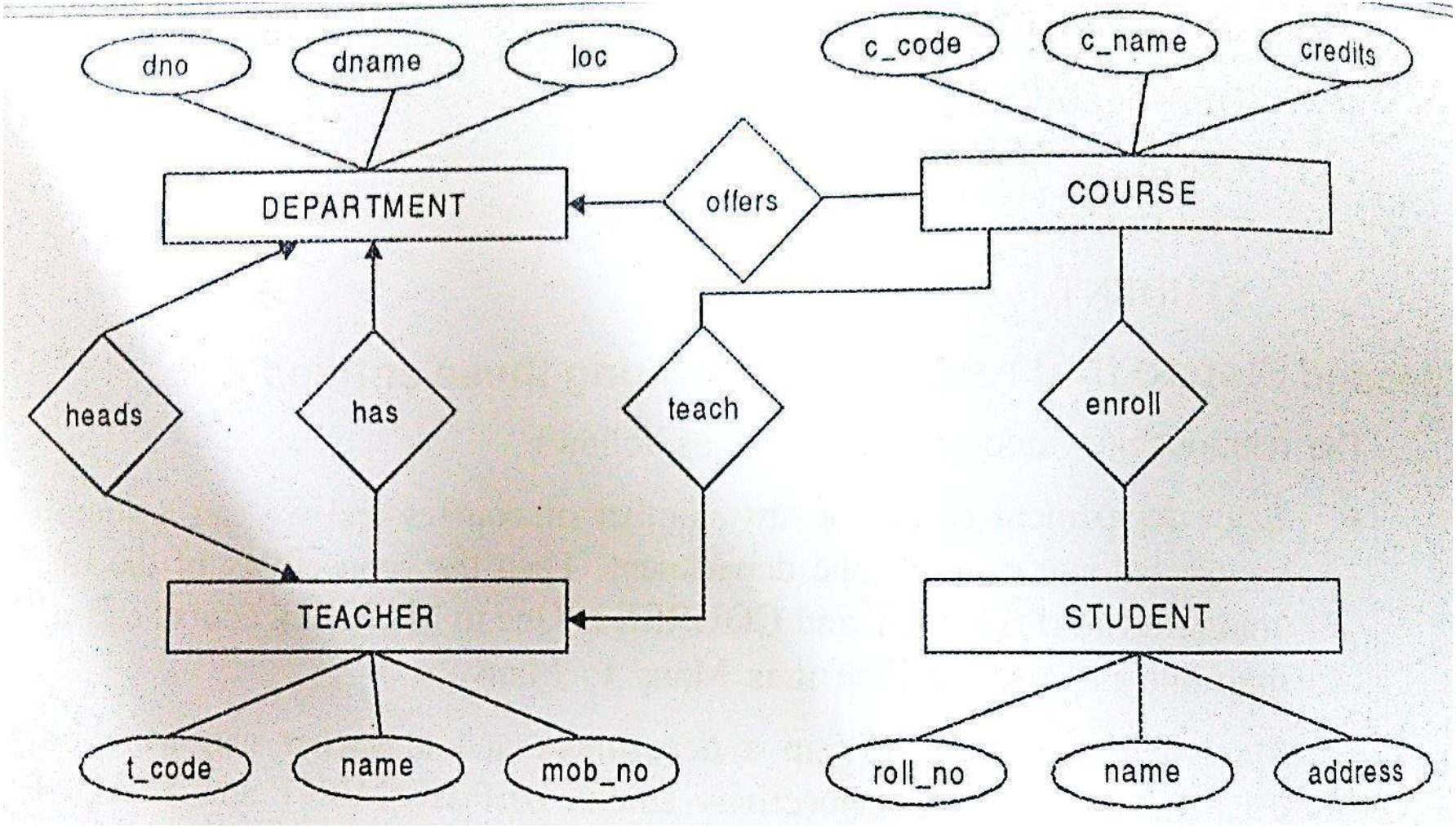
Example


studentId
firstName
lastName
courseId
L0002345
Jim
Black
C002
L0001254
James
Harradine
A004
L0002349
Amanda
Holland
C002
L0001198
Simon
McCloud
S042
Foreign Keys
Relationship
Primary Keys
courseId
courseName
A004
Accounts
C002
Computing
P301
History
S042
Short Course

Steps in designing E-R Diagram

- There are following steps:
  - Identify the entities from the requirement sheets
  - Find relationships among those entities
  - Identify the key attributes for every entity
  - Identify other relevant attributes
  - Draw complete E-R diagram with all attributes including Primary key
  - Review your results with your Business users

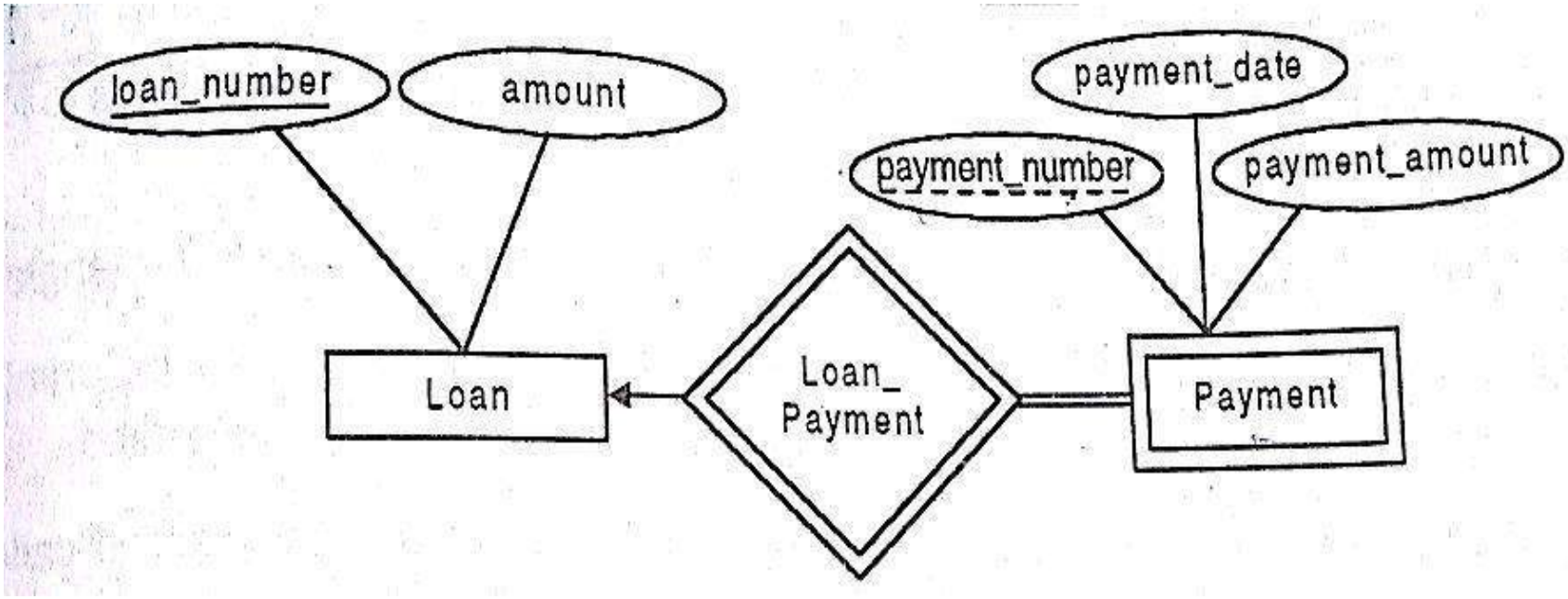
Example: University Management System



Strong and Weak entity sets

- The entity set which does not have sufficient attributes to form a primary key is called a Weak entity set.
- The entity set which has the primary key is called as Strong entity set.
- A member of a strong entity set is called dominant entity and member of a weak entity set is called subordinate entity.
- Weak entity does not have a primary key but we need a mean to distinguish among other entities. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made. Ex: payment_number.

Example


loan_number
amount
payment_date
payment_number
payment_amount
Loan
Loan_
Payment
Payment

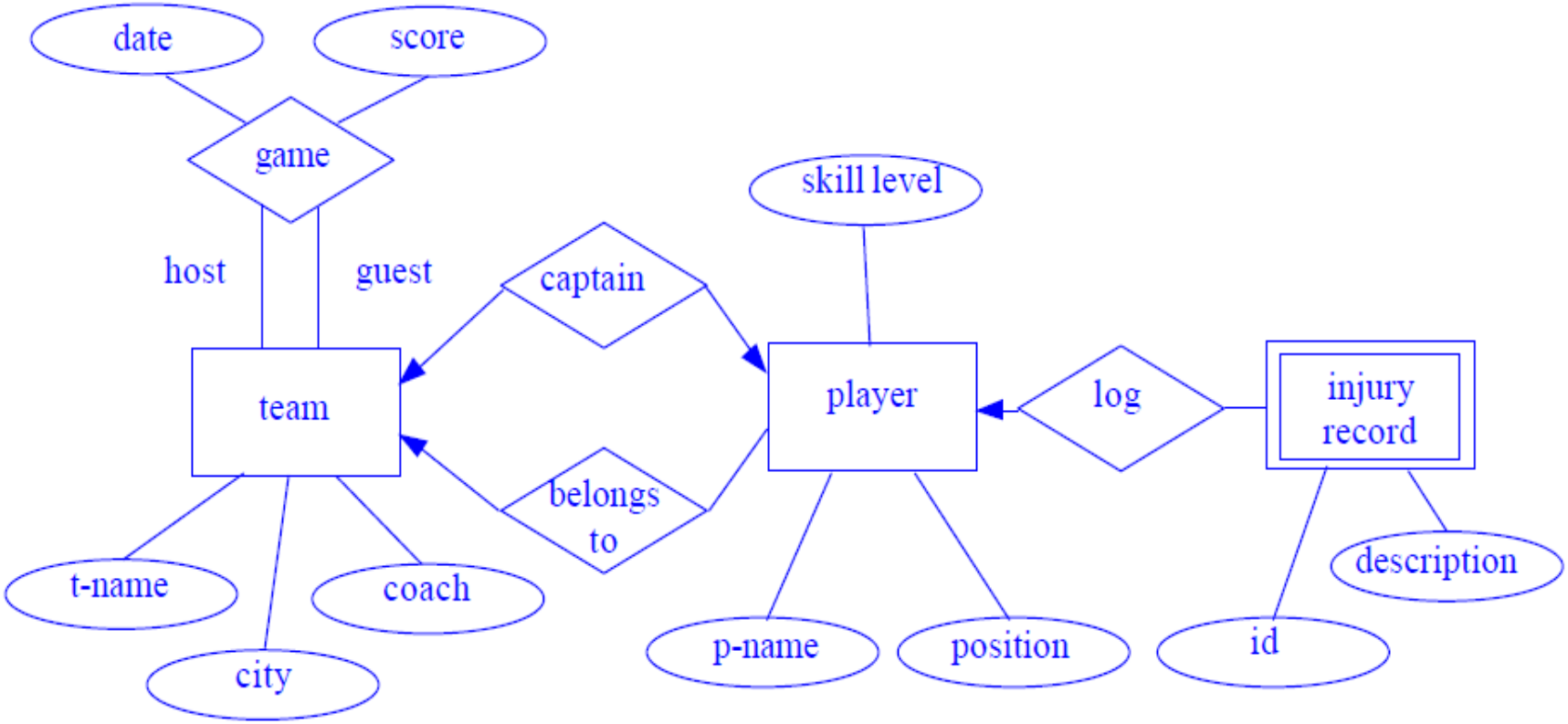
Example

Suppose you are given the following requirements for a simple database for the National Hockey League (NHL):

- the NHL has many teams,
- each team has a name, a city, a coach, a captain, and a set of players,
- each player belongs to only one team,
- each player has a name, a position (such as *left wing* or *goalie*), a skill level, and a set of injury records,
- a team captain is also a player,
- a game is played between two teams (referred to as host_team and guest_team) and has a date (such as *May 11th, 1999*) and a score (such as *4 to 2*).

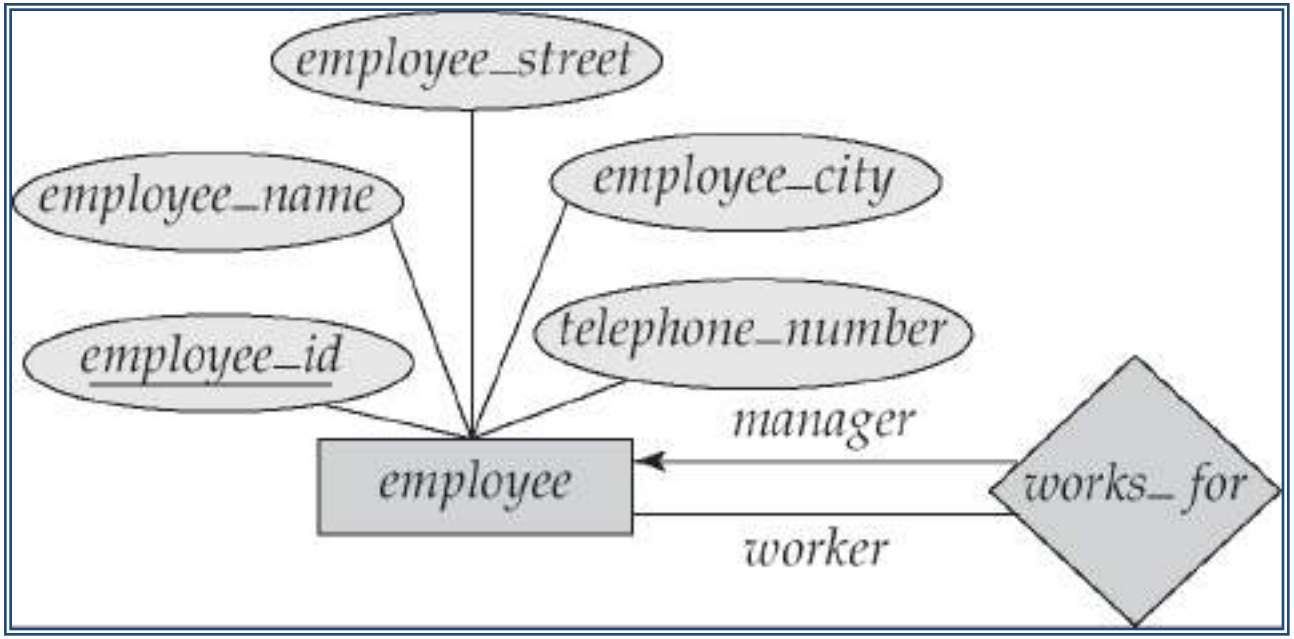
Construct a clean and concise ER diagram for the NHL database

E-R Diagram


date
score
game
host
guest
captain
skill level
team
player
log
injury record
belongs to
t-name
coach
city
p-name
position
id
description

Roles

- Entity sets of a relationship need not be distinct
- The labels “manager” and “worker” are called **roles**; they specify how employee entities interact via the works_for relationship set.
- Roles are indicated in E-R diagrams by labeling the lines that connect diamonds to rectangles.
- Role labels are optional, and are used to clarify semantics of the relationship



Removing Redundant attributes in entity set

- Once the entities and their corresponding attributes are chosen, the relationship sets among the various entities are formed.
- These relationship sets may result in a situation where attributes in the various entity sets are redundant and need to be removed .
- Consider the entity sets *instructor* and *department*
  - Instructor includes the attributes ID, name, de*pt_name, and salary with ID as primary key*
  - *Department includes the attributes dept_name, building, and budget with dept_name as primary key*

Removing Redundant attributes in entity set

- Attribute dept_name appears in both entity sets. It is primary key for entity department, it is redundant in the entity set instructor and needs to be removed.
- If both entities have one to one relationship then we can remove it from instructor table as it will get added up in the relational schema of department.
- But if an instructor has more than one associated department, the relationship between the entities is recorded in a separate relation inst_dept

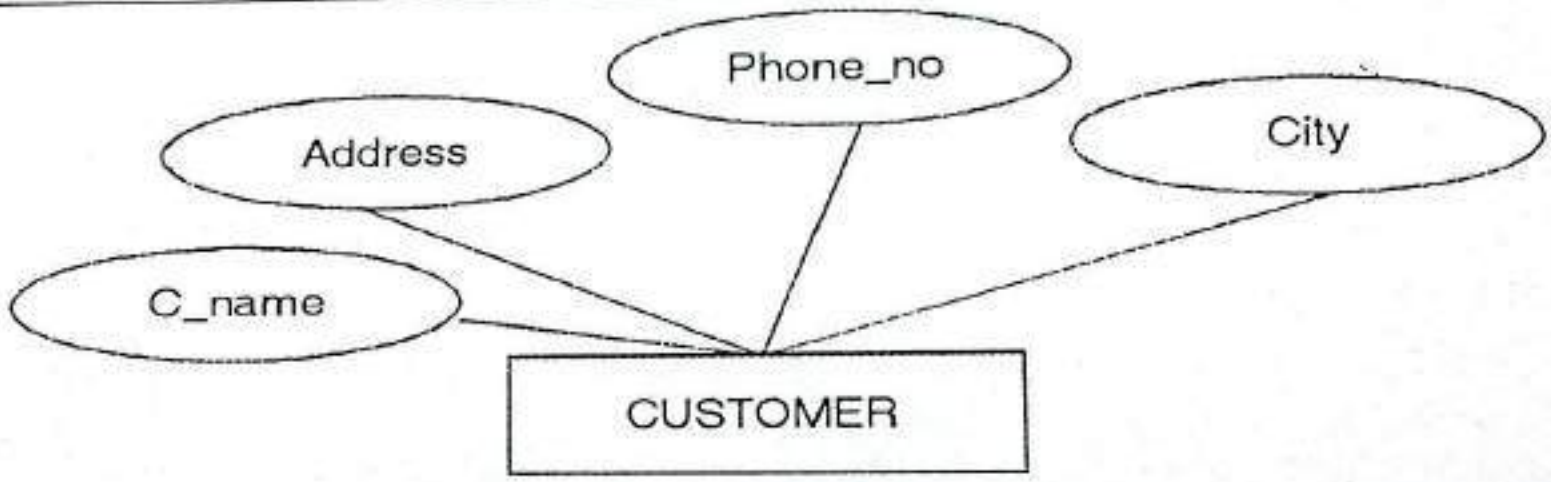
Reduction to relational schema

- Both E-R model and relational database model are abstract, logical representations of real-world enterprises.
- Because the two models employ similar design we can convert an E-R design to relational design
- E-R schema can be represented by relation schema in following ways:
  - Representing of strong entity sets with Simple Attributes
  - Representing of strong entity sets with Complex Attributes
  - Representation of weak entity sets
  - Representation of relationship sets

Representing of strong entity sets with Simple Attributes

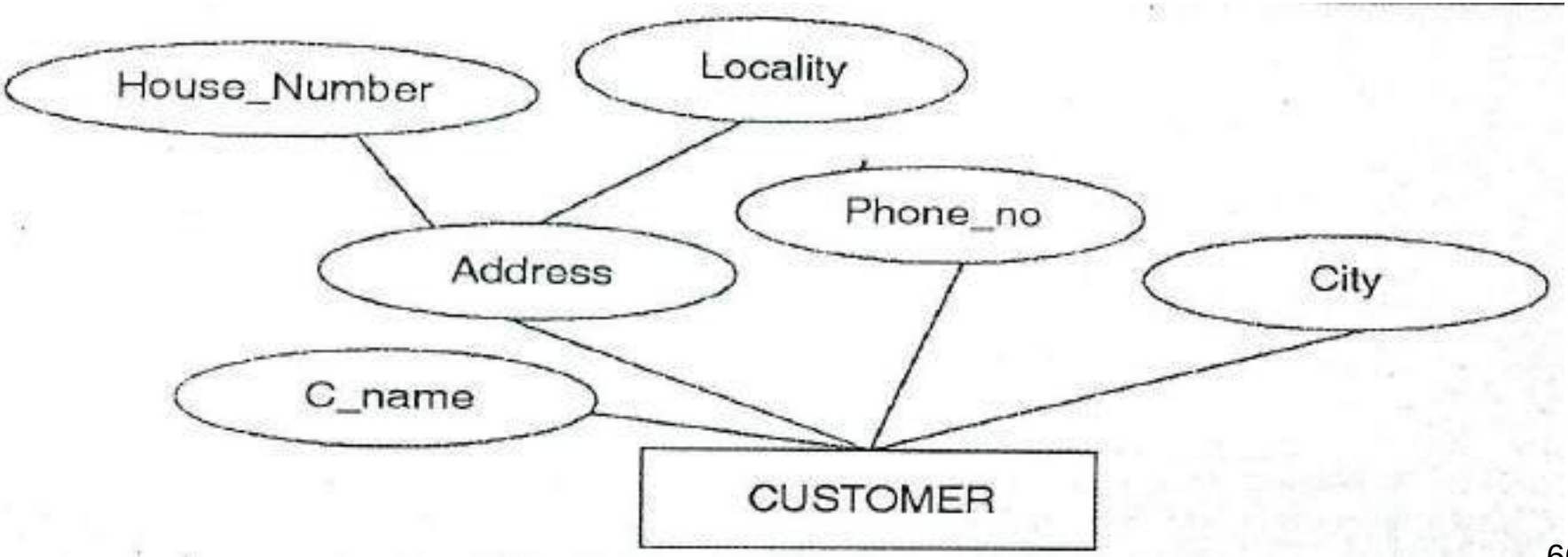
- A strong entity set reduces to a table with the same attributes as represented in E-R diagram

C_Name	Address	Phone_No	City
Rahat	TU	2444566	Patiala
Ruhani	NIT	4547235	Jalandhar
Raja	Model Town	3445432	Amritsar

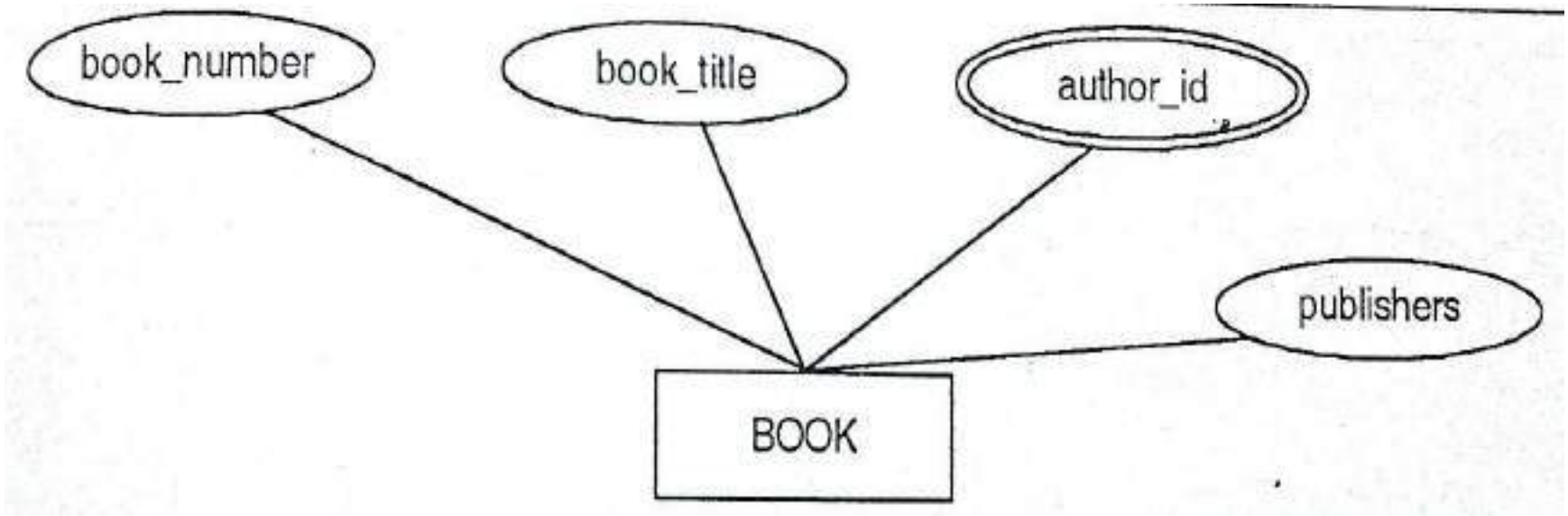


Representing of strong entity sets with Complex Attributes

- In order to convert an entity having composite attributes, the composite attributes are flattened out by creating a separate attribute for each component attribute.



- Multivalued attributes are treated differently from other attributes. New relation schemas are created for these attributes.
- Derived attributes are not explicitly represented in the relational model.



BOOK entity set with author_id as multi-value attribute

Instances of BOOK entity

Book_Number	Book_Name	Author_Id	Publisher
B1	DBMS	A1	Kalyani
B1	DBMS	A2	Kalyani
B1	DBMS	A3	Kalyani
B2	Oracle	C1	Kalyani
B2	Oracle	C2	Kalyani

Book_Author

Book_Number	Author_Id
B1	A1
B1	A2
B1	A3
B2	C1
B2	C2

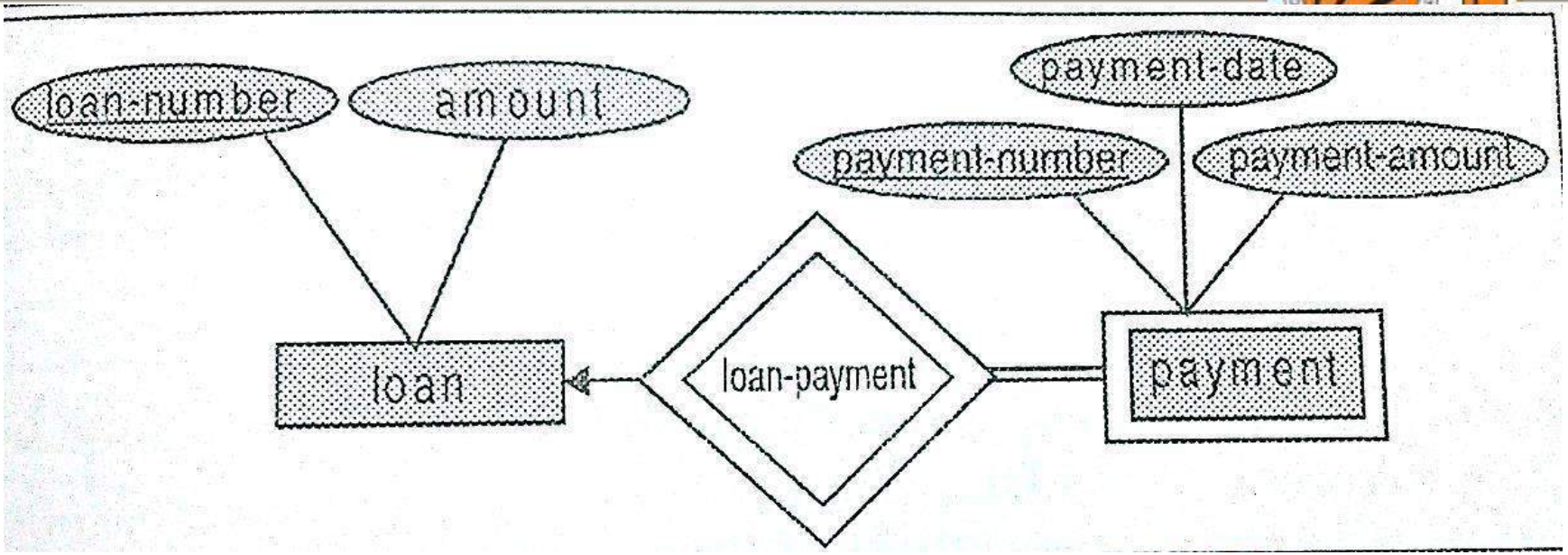
Book_Info

Book_Number	Book_Name	Publisher
B1	DBMS	Kalyani
B2	Oracle	Kalyani

Representation of weak entity sets

- For schemas derived from a weak entity set, the combination of the primary key of the strong entity set and the discriminator of the weak entity set serves as primary key of the schema.

  payment = (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)



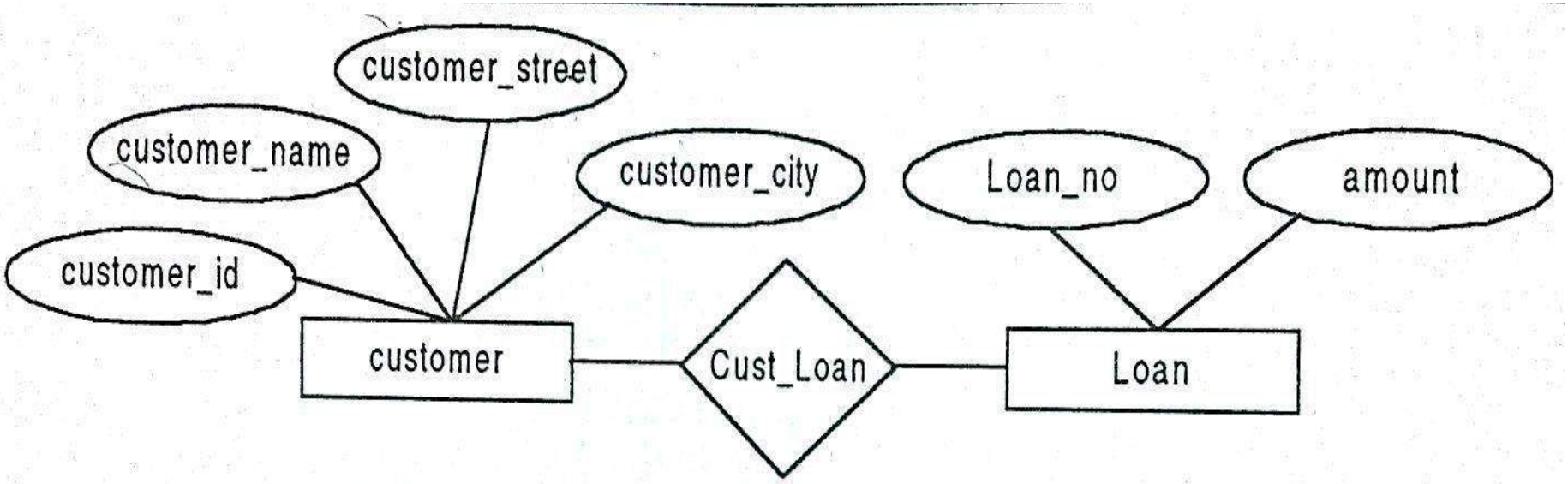
loan-number	payment-number	payment-date	payment-amount
L-11	53	7 June 2001	125
L-14	69	28 May 2001	500
L-15	22	23 May 2001	300
L-16	58	18 June 2001	135
L-17	5	10 May 2001	50
L-17	6	7 June 2001	50
L-17	7	17 June 2001	100
L-23	11	17 May 2001	75
L-93	103	3 June 2001	900
L-93	104	13 June 2001	200

Representation of relationship sets

- There are different strategies used for each type of relationship.

1. Conversion of many to many relationship to relational model:
   - A many to many relationship set is represented as a table with columns for the primary keys of the two participating entity sets, and the descriptive attributes of the relationship set.

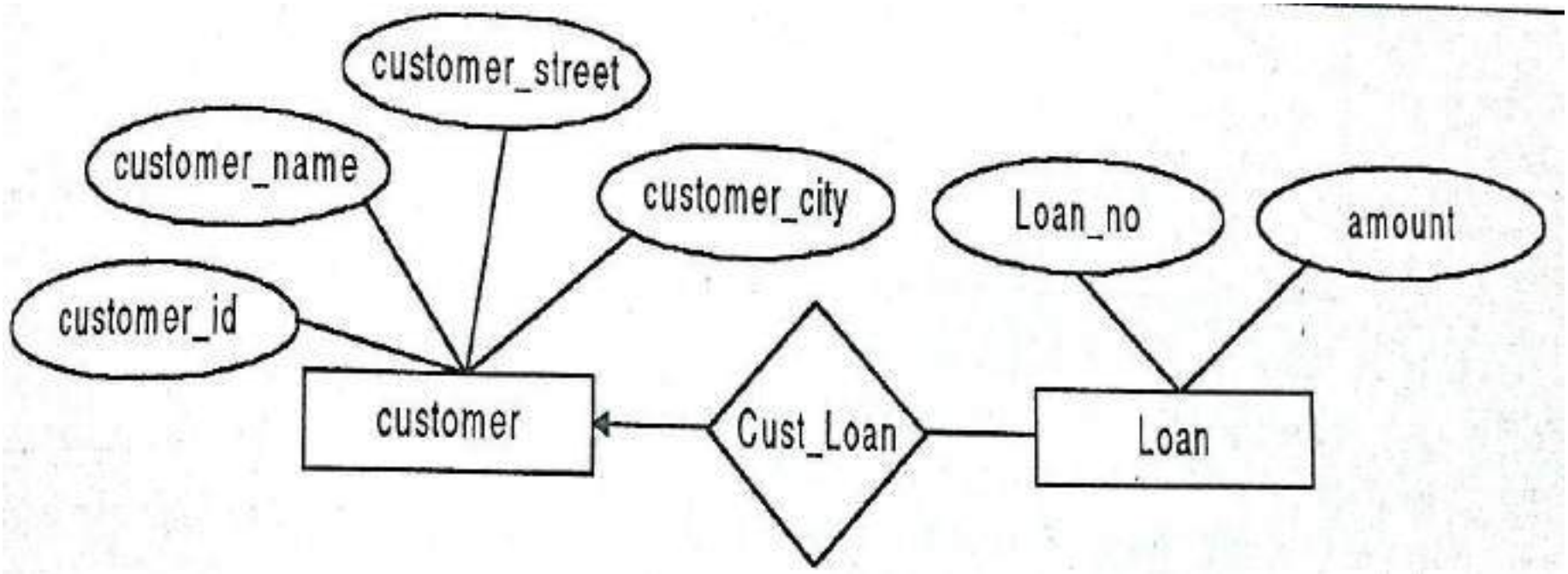
customer_id	loan-number
C1	L1
C1	L2
C2	L2
C2	L3
C3	L3



2. Conversion of one to many relationship to relational model:

For one to many or many to one relationship, there is no need to create the separate table for the relationship. Copy the primary key of entity set on one side of relationship into the entity set on “many” side of relationship.

customer_id	loan-number
C1	L1
C1	L2
C2	L3
C2	L4
C3	L5



Customer and Loan relationship with 1:M type

customer_id	customer_name	cutomer_street	customer_city	loan_nubmer
C1	Ajay	Model Town	Patiala	L1
C1	Ajay	Model Town	Patiala	L2
C2	Ram	Dharampura	Patiala	L3
C2	Ram	Dharampura	Patiala	L4

customer_id	customer_name	cutomer_street	customer_city	loan_nubmer
C1	Ajay	Model Town	Patiala	L1, L2
C2	Ram	Dharampura	Patiala	L3, L4

This is not a valid database, so this solution is rejected.

loan_number	amount	Customer_id
L1	10000	C1
L2	20000	C1
L3	30000	C2
L4	40000	C2

3. Conversion of one to one relationship to relational model:

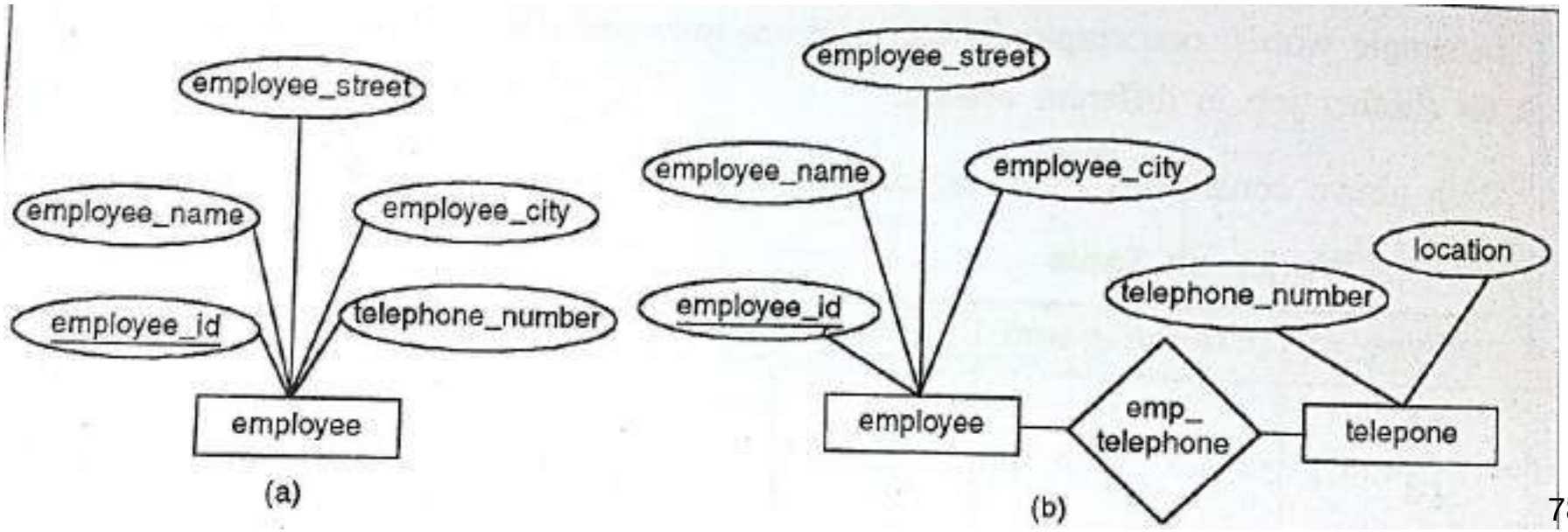
- In case of 1:1 relationship, there is no need to create a separate table for relationship and the primary key of any entity set can be moved to other side depending upon the requirement. It is preferable to copy the primary key of non-totally participated entity set towards totally participated entity set.

Design Issues

- There are number of different ways to define set of entities and relationship among them. We examine basic issues in the design of an E-R model. These are:
  - Use of entity sets vs. attributes
  - Use of entity sets vs. relationship sets
  - Binary versus n-ary relationship sets
  - Placement of relationship attributes

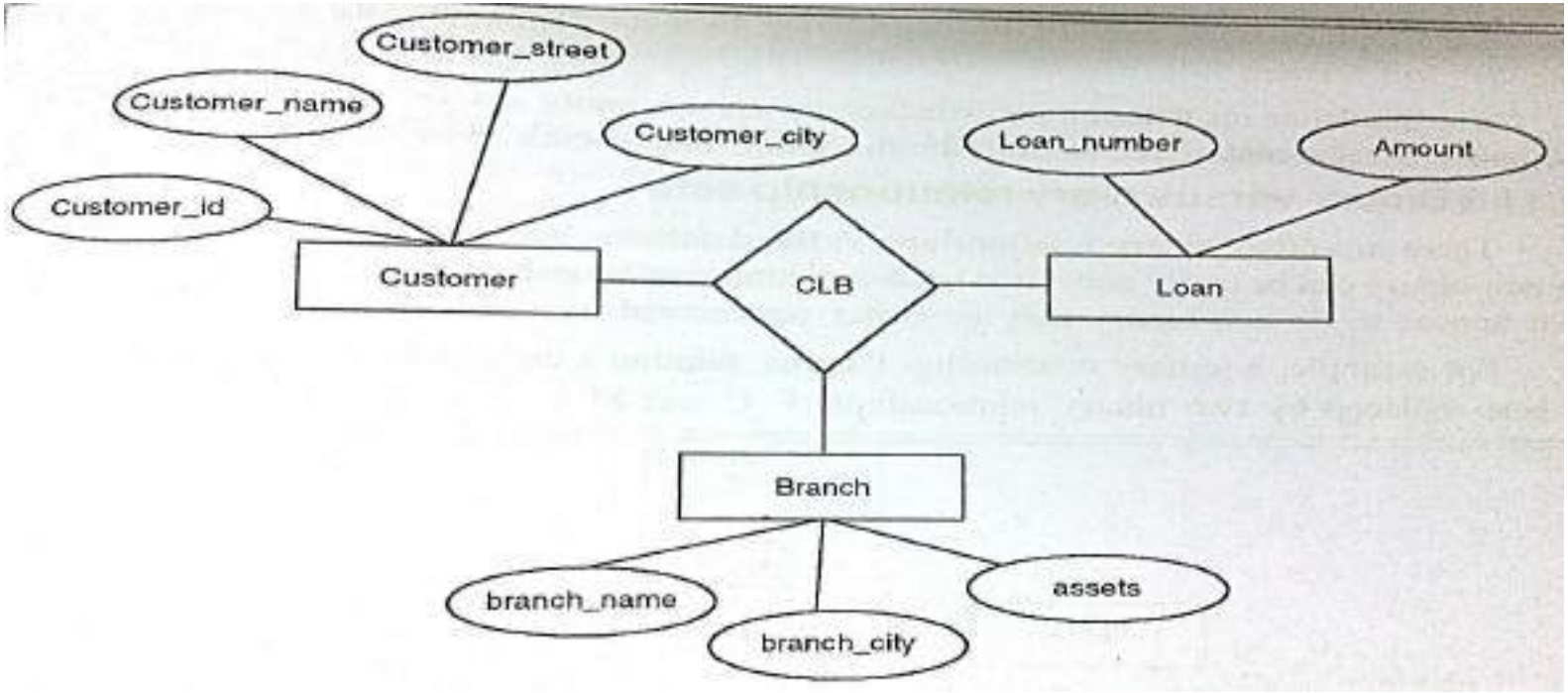
Design Issues: Use of entity sets vs. attributes

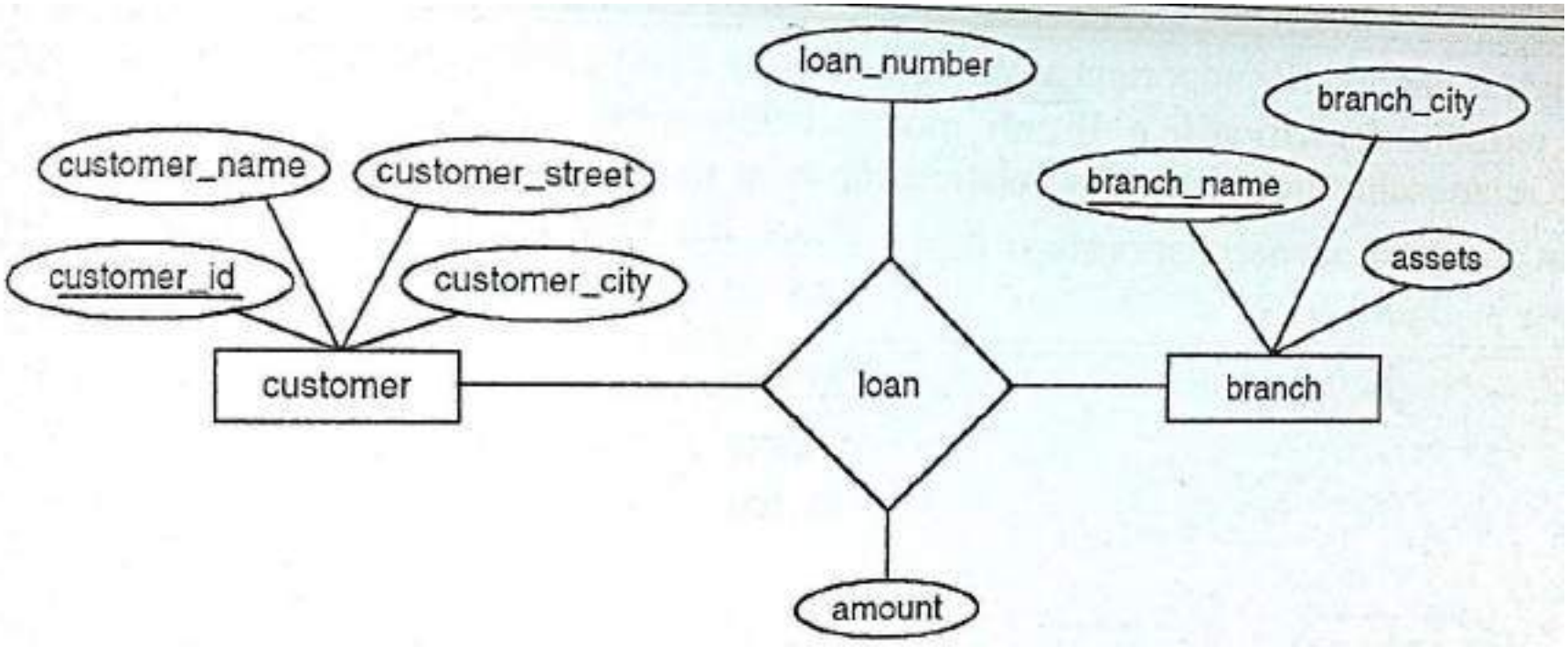
- Treating telephone as an attribute telephone-number implies that employees have exactly one telephone number each
- Treating telephone as an entity permits employees to have several telephone numbers associated with them.
- The distinctions to consider an object as attribute or entity depends on the structure of real-world enterprise being modeled.



Design Issues: Use of entity sets vs. relationship sets

- Each loan is represented by a relationship between a customer and a branch, if every loan is associated with exactly one customer and is associated with exactly one branch
- If several customers hold a joint loan then, loan should be considered as an entity

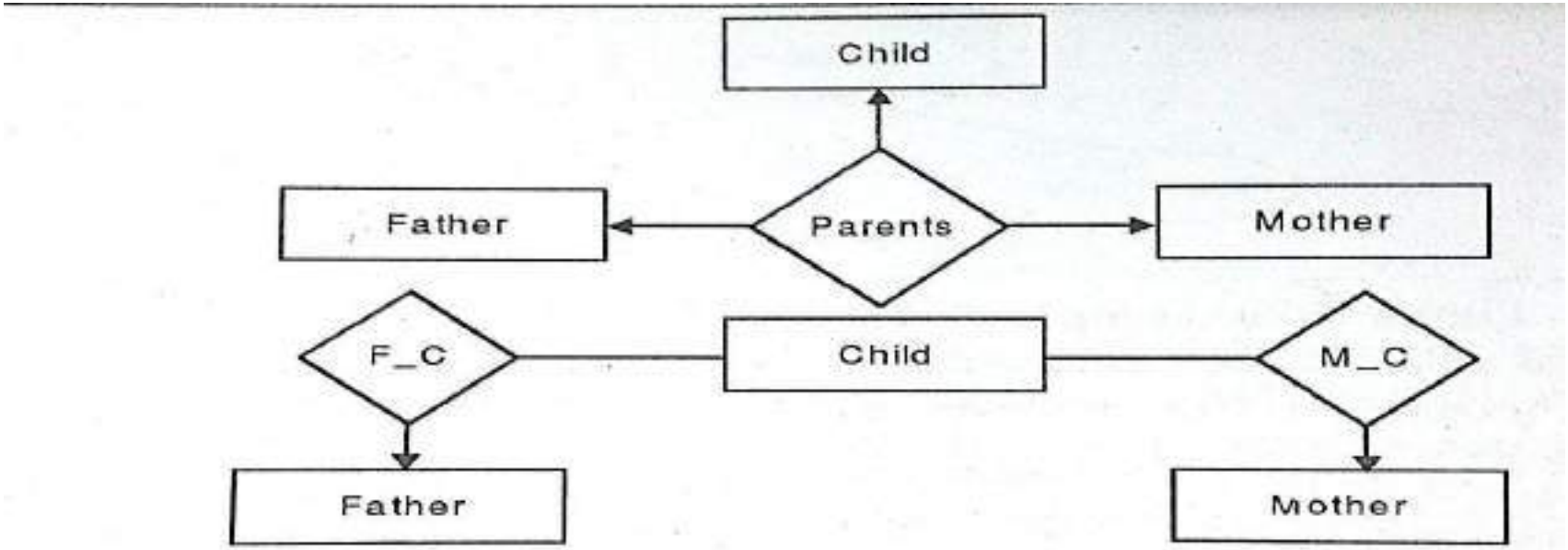




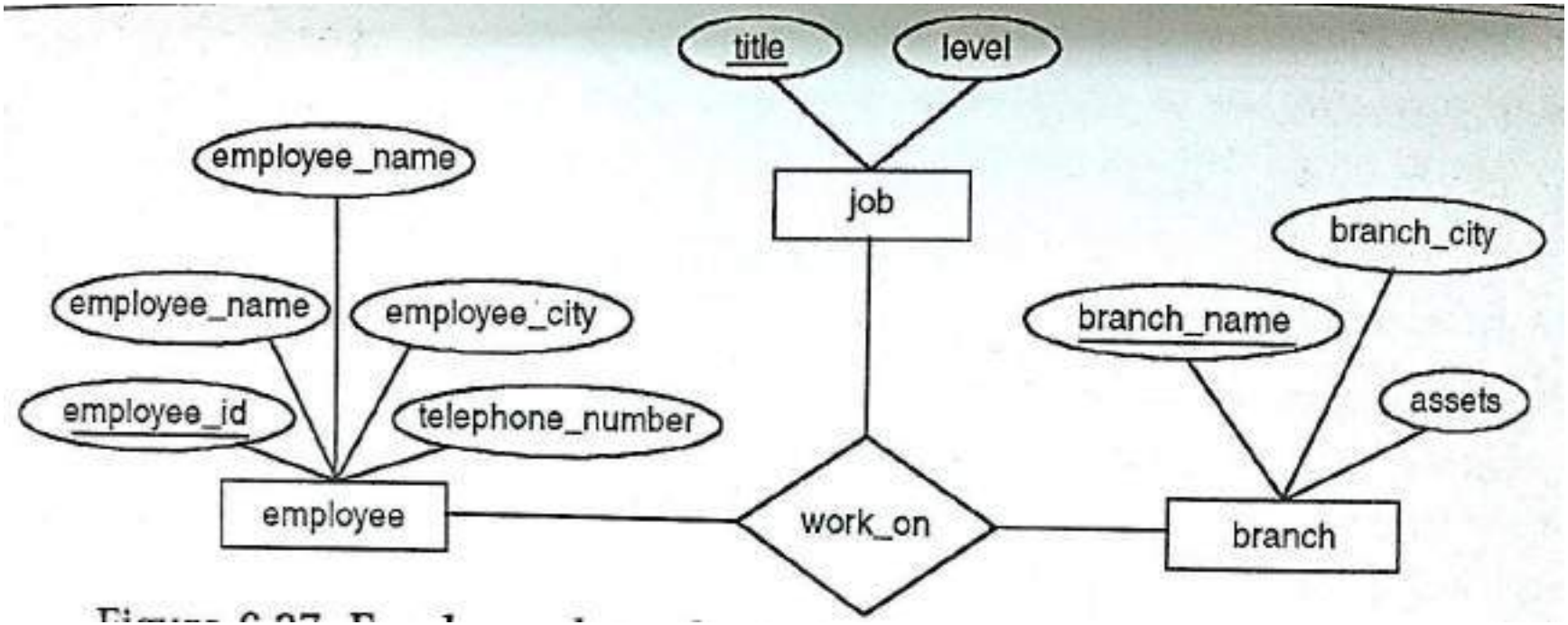
The guideline in determining whether to use an entity set or a relationship set is to designate a relationship set to describe an action that occurs between entities

Design Issues: Binary versus n-ary relationship sets

- Although it is possible to replace any nonbinary (*n*-ary, for *n* > 2) relationship set by a number of distinct binary relationship sets, a *n*-ary relationship set shows more clearly that several entities participate in a single relationship.



- There is one advantage of using binary relationship, with this it is possible to store partial information but in ternary relationship it is not possible to represent it.
- But there are some relationships that are naturally non-binary.



Design Issues: Placement of relationship attributes

- The cardinality ratio of a relationship can affect the placement of relationship attributes
- Example: Let us take two entities instructor and student with relationship advisor, if we want to add attribute date (which specifies when the instructor became advisor)
- If relationship is one to many then the date must be placed to the entity set on the “many” side of relationship
- For one to one relationship, it could be placed on either side
- For many to many relationship, the date must be placed as the relationship attribute

Formal Relational Query Languages

BY: Richa Jain

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.
- Relation is a table with columns and rows.

- Select , project and rename operations are called unary operations because they operate on one relation
- Other three operations are called Binary operations because they operate on pair of relations

Select Operation

- Select operation selects tuples that satisfies a given predicate
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“Physics”}}(instructor)$$

Select Operation – Example

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation

- Notation: $\Pi_{A_1, A_2, \ldots, A_k}(r)$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Composition of Relational Operations

- Find those customers who live in Harrison?
- $\Pi_{customer\text{-}name}\ (\sigma_{customer\text{-}city} = \text{“Harrison”}\ (customer))$

loan-number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Loan number and the amount of the loan.

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.

  1. *r, s* must have the *same* **arity** (same number of attributes)

  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

Union Operation – Example

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Example

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Example

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Fall”}\ \wedge\ year=2009}\,(section))\ \cup$$

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Spring”}\ \wedge\ year=2010}\,(section))$$

- Since relations are sets duplicate values such as CS101, which is offered in both semesters are replaced by a single occurrence.

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) -$$
$$\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* – *s*:

A	B
α	1
β	1

Cartesian-Product Operation

- Notation $r \times s$
- Defined as:

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Cartesian-Product Operation—Example

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Composition of Operations

- Can build expressions using multiple operations
- Example:
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_x(E)$$

returns the expression *E* under the name *X*

- If a relational-algebra expression *E* has arity *n*, then

$$\rho_{x(A_1,A_2,\ldots,A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Example Query

- Find the largest salary in the university
  - Step 1: find instructor salaries that are less than some other instructor salary (i.e. not maximum)
    - using a copy of *instructor* under a new name *d*
      - $\prod_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_{d}(instructor)))$
  - Step 2: Find the largest salary
    - $\prod_{salary} (instructor) - \prod_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_{d}(instructor)))$

Example Queries

- Find the names of all instructors in the Physics department, along with the *course_id* of all courses they have taught

- Query 1

$$\Pi_{instructor.ID,course\_id} (\sigma_{dept\_name=\text{“Physics”}} (\sigma_{instructor.ID=teaches.ID} (instructor \times teaches)))$$

- Query 2

$$\Pi_{instructor.ID,course\_id} (\sigma_{instructor.ID=teaches.ID} (\sigma_{dept\_name=\text{“Physics”}} (instructor) \times teaches))$$

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s (E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

We define additional operations that do not add any power to the
relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$
- Suppose that we wish to find all customers who have both a loan and an account. Using set intersection, we can write

$$\Pi_{customer\text{-}name} (borrower) \cap \Pi_{customer\text{-}name} (depositor)$$

Set-Intersection Operation – Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural-Join Operation

- Notation: $r \bowtie s$

- A natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- The natural join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relational schemas, and finally removes duplicate attributes.
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  – Result schema = (A, B, C, D, E)

  – $r \bowtie s$ is defined as:

$$\prod_{r.A,\, r.B,\, r.C,\, r.D,\, s.E} (\sigma_{r.B = s.B \,\wedge\, r.D = s.D} (r \times s))$$

Natural Join Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural Join and Theta Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Comp. Sci.”}}(instructor \bowtie teaches \bowtie course))$

- The theta join operation is a variant of the natural join operation that allows us to combine a selection and a Cartesian product into a single operation.
- θ in Theta join is the join condition. Theta joins combines tuples from different relations provided they satisfy the theta condition.
- The **theta join** operation $r \bowtie_\theta s$ is defined as

  - $r \bowtie_\theta s = \sigma_\theta(r \times s)$

Assignment Operation

- The assignment operation (←) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.

Division Operator

- Not supported as a primitive operator, but useful for expressing queries like:

  Find sailors who have reserved ***all*** *boats*.

- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:

$$A/B = \{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B\}$$

i.e., *A/B* contains all *x* tuples (sailors) such that for *every y* tuple (boat) in *B*, there is an *xy* tuple in *A*.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.

Outer Join – Example

- Relation *instructor1*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation teaches1

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example

- Join

  instructor ⋈ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- *Left Outer Join*
- *instructor* ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join

instructor ⟖ *teaches*

ID	name	dept_name	course_id
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

instructor ⟗ *teaches*

ID	name	dept_name	course_id
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples t such that predicate P is true for t
- t is a *tuple variable*, $t[A]$ denotes the value of tuple t on attribute A
- $t \in r$ denotes that tuple t is in relation r
- P is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\,(Q\,(t\,)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q\,(t\,)$ is true
   - $\forall t \in r\,(Q\,(t\,)) \equiv Q$ is true "for all" tuples t in relation r

Record Based Data Models

BY: RICHA JAIN

Data Models

- A data model comprises of three components:
  - ***A structural part*, consisting of a set of rules according to which databases can be constructed.**
  - ***A manipulative part*, defining the types of operation that are allowed on the data.**
  - ***Possibly a set of integrity rules*, which ensures that the data is accurate.**

Data Models

- The purpose of a data model is to represent data and to make the data understandable.
- There have been many data models proposed in the literature. They fall into three broad categories:
  - **Object Based Data Models**
  - **Record Based Data Models**
  - **Physical Data Models**

Object Based Data Models

- Object based data models use concepts such as entities, attributes, and relationships.
- The types of object based data model are:
  - ***Entity-Relationship Model*: It has emerged as one of the main techniques for modeling database design and forms the basis for the database design methodology.**
  - ***Object Oriented Model*: It extends the definition of an entity to include, not only the attributes that describe the state of the object but also the actions that are associated with the object, that is, its behavior.**

Record Based Data Models

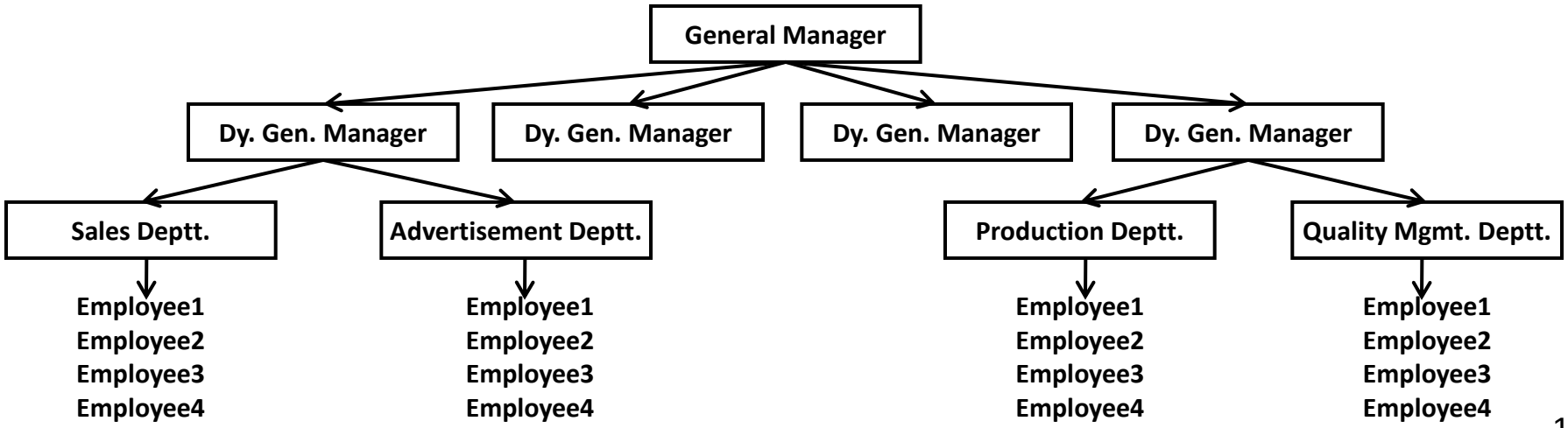
- In contrast to object based data models, they are used to specify the overall logical structure of the database and to provide a higher-level description of the implementation. The three most widely accepted record based data models are:
  - ***Hierarchical Model***
  - ***Network Model***
  - ***Relational Model***

Physical Data Models

- Physical data models describe how data is stored in the computer, representing information such as record structures, record ordering, and access paths.
- There are not as many physical data models as logical data models, the most common one being the *Unifying Model*.

Hierarchical Model

- Hierarchical Database model is one of the oldest database models, dating from late 1950s. This model is like a structure of a tree with the records forming the nodes and fields forming the branches of the tree.



Hierarchical Model

- Supplier Records

S_No.	Name	Status	City
S1	Suneet	20	Qadian
S2	Ankit	10	Amritsar
S3	Amit	30	Amritsar

- Part Records

P_No.	Name	Color	Weight	City
P1	Nut	Red	12	Qadian
P2	Bolt	Green	17	Amritsar
P3	Screw	Blue	17	Jalandhar
P4	Screw	Red	14	Qadian

- Shipment Records

S_No.	P_No.	Qty.
S1	P1	250
S1	P2	300
S1	P3	500
S2	P1	250
S2	P2	500
S3	P2	300

Hierarchical Model

P1	Nut	Red	12	Qadian

S1	Suneet	20	Qadian	250
S2	Ankit	10	Amritsar	250

P2	Bolt	Green	17	Amritsar

S1	Suneet	20	Qadian	300
S2	Ankit	10	Amritsar	500
S3	Amit	30	Amritsar	300

P3	Screw	Blue	17	Jalandhar

S1	Suneet	20	Qadian	500

P4	Screw	Red	14	Qadian

Operations on Hierarchical Model

1. ***Insert Operation:*** It is not possible to insert the information of the supplier e.g. S4 who does not supply any part. This is because a node cannot exist without a root. Since, a part P5 that is not supplied by any supplier can be inserted without any problem, because a parent can exist without any child. So, we can say that insert anomaly exists only for those children, which has no corresponding parents.

Operations on Hierarchical Model

2. *Update Operation:* Suppose we wish to change the city of supplier S1 from Qadian to Jalandhar, then we will have to carry out two operations such as searching S1 for each part and then multiple updations for different occurrences of S1. But if we wish to change the city of part P1 from Qadian to Jalandhar, then these problems will not occur because there is only a single entry for part P1.

The problem of inconsistency will not arise. So, we can say that update anomalies only exist for children not for parent because children may have multiple entries in the database.

Operations on Hierarchical Model

3. Delete Operation: In hierarchical model, quantity information is incorporated into supplier record. Hence the only way to delete a shipment (or supplied quantity) is to delete the corresponding supplier record. But such an action will lead to loss of information of the supplier, which is not desired.

For example: Supplier S2 stops supplying 250 quantity of part P1, then the whole record of S2 has to be deleted under part P1 which may lead to loss the information of supplier. Another problem will arise if we wish to delete a part information and that part happens to be only part supplied by some supplier.

Operations on Hierarchical Model

- The deletion of parent causes the deletion of child records also and if the child occurrence is the only occurrence in the whole database, then the information of child records will also be lost with the deletion of parent. E.g., if we wish to delete the information of part P2 then we also lost the information of S3, S2 and S1 supplier. The information of S2 and S1 can be obtained from P1, but the information about supplier S3 is lost with the deletion of record for P2.

Operations on Hierarchical Model

4. *Record Retrieval:* Record retrieval methods for hierarchical model are complex and asymmetric which can be clarified with the following queries:

Query1: Find the supplier number for suppliers who supply part P2.

- Solution: First we search the information of parent P2 from database, since parent occurs only once in the whole database, so we obtain only a single record for P2. Then, a loop is constructed to search all suppliers under this part and supplier numbers are printed for all suppliers.

Operations on Hierarchical Model

Query2: Find part numbers for parts supplied by supplier S2.

- Solution: In order to get required part number we have to search S2 under each part. If supplier S2 is found under a part then the corresponding part number is printed, otherwise we go to next part until all the parts are searched for supplier S2.

Advantages of Hierarchical Model

- ***Simplicity:*** Since the database is based on the hierarchical structure, the relationship between the various layers is logically (conceptually) simple. Thus the design of a hierarchical database is simple.
- ***Data Security:*** Hierarchical model was the first database model that offered the data security that is provided and enforced by the DBMS.
- ***Data Integrity:*** The child segments are always automatically referred by its parent, so this model promotes data integrity.

Advantages of Hierarchical Model

- ***Efficiency:*** This model is a very efficient, one when the database contains a large number of 1: N relationships (one-to-many relationships) and when the users require large number of transactions, using data whose relationships are fixed.

Disadvantages of Hierarchical Model

- ***Implementation Complexity:*** The database designers should have very good knowledge of the physical data storage characteristics.
- ***Database Management Problems:*** If you make any changes in the database structure of a hierarchical database, then you need to make the necessary changes in all the application programs that access the database.
- ***Lack of Structural Independence:*** Hierarchical database systems use physical storage paths to navigate to the different data segments. So the application programs should have a good knowledge of the relevant access paths to access the data.

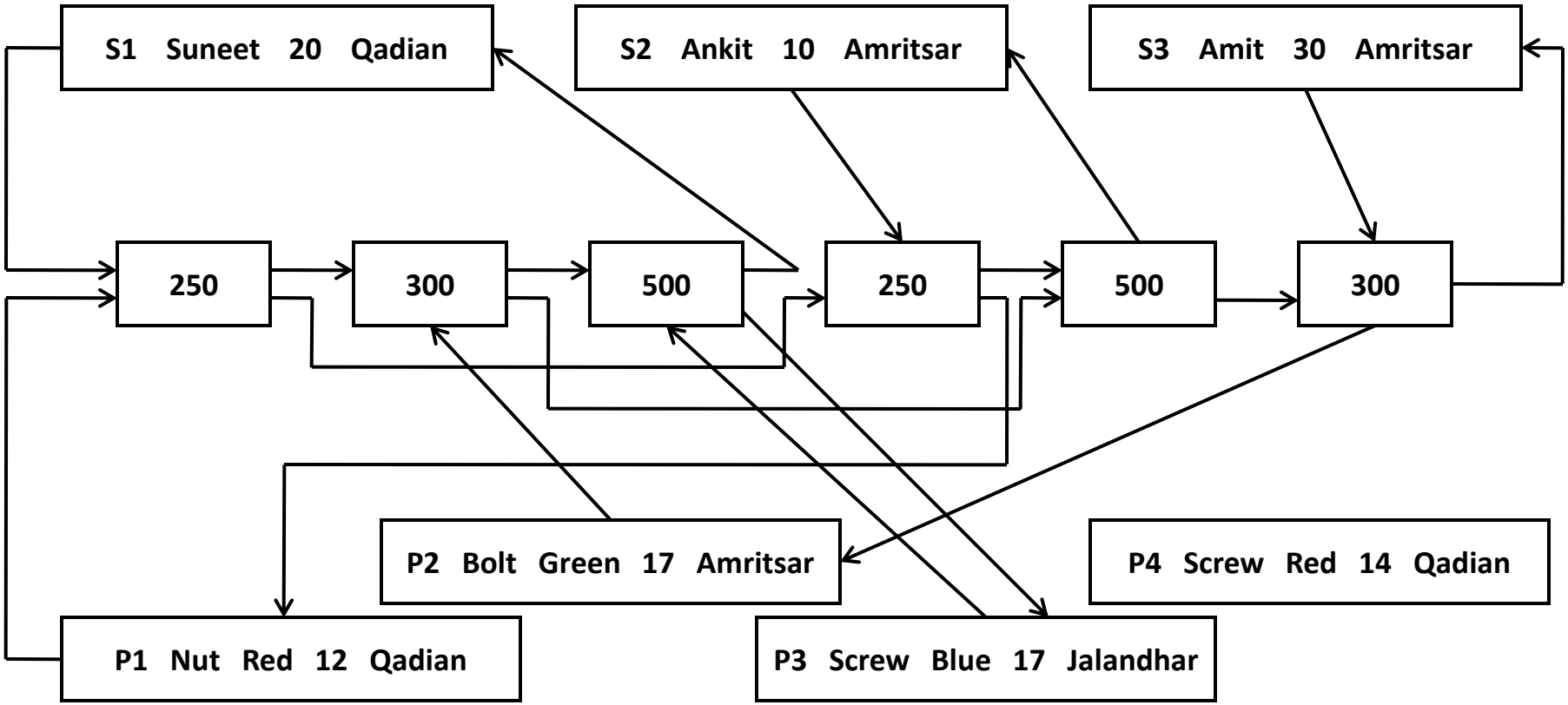
Disadvantages of Hierarchical Model

- ***Programs Complexity:*** *It* requires know-ledge of complex pointer systems, which is often beyond the grasp of ordinary users.
- ***Operational Anomalies:*** This model suffers from the Insert anomalies, Update anomalies and Deletion anomalies, retrieval operation is complex and asymmetric, thus it is not suitable for all the cases.
- ***Implementation Limitation:*** The many-to-many (N:N) relationships, which are more common in real life are very difficult to implement in a hierarchical model.

Network Model

- The Network model replaces the hierarchical tree with a graph thus allowing more general connections among the nodes.
- A network structure allows 1:1, 1:M, M:M relationships among entities.

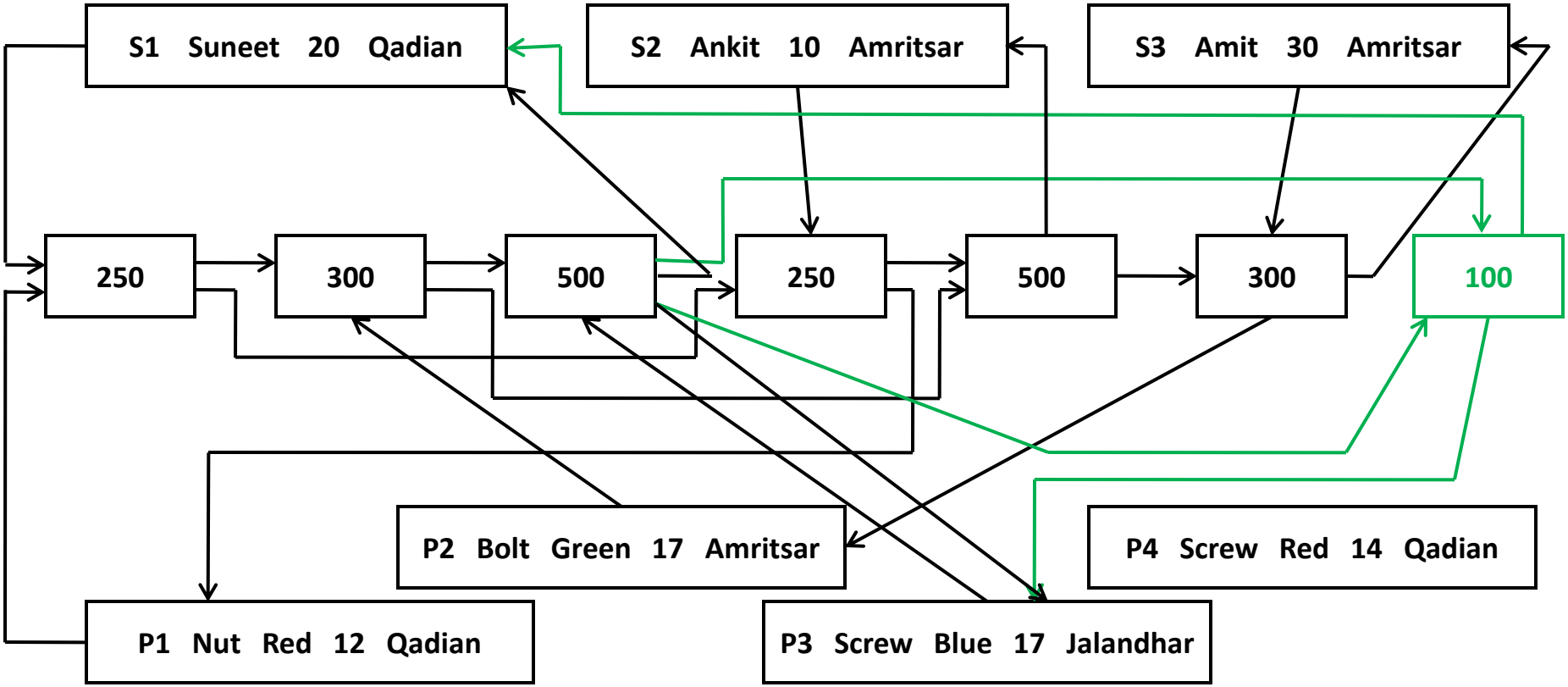
Network View of Sample Database



Operations on Network Model

- ***Insert Operation:*** To insert a new record containing the details of a new supplier, we simply create a new record occurrence. Initially, there will be no connector. The new supplier's chain will simply consist of a single pointer starting from the supplier to itself. e.g., supplier S4 can be inserted in network model that does not supply any part as a new record occurrence with a single pointer from S4 to itself.
- Similarly a new part can be inserted who does not supplied by any supplier.
- Consider another case if supplier S1 now starts supplying P3 part with quantity 100, then a new connector containing the 100 as supplied quantity is added in to the model and the pointer of S1 and P3 are modified.

Operations on Network Model



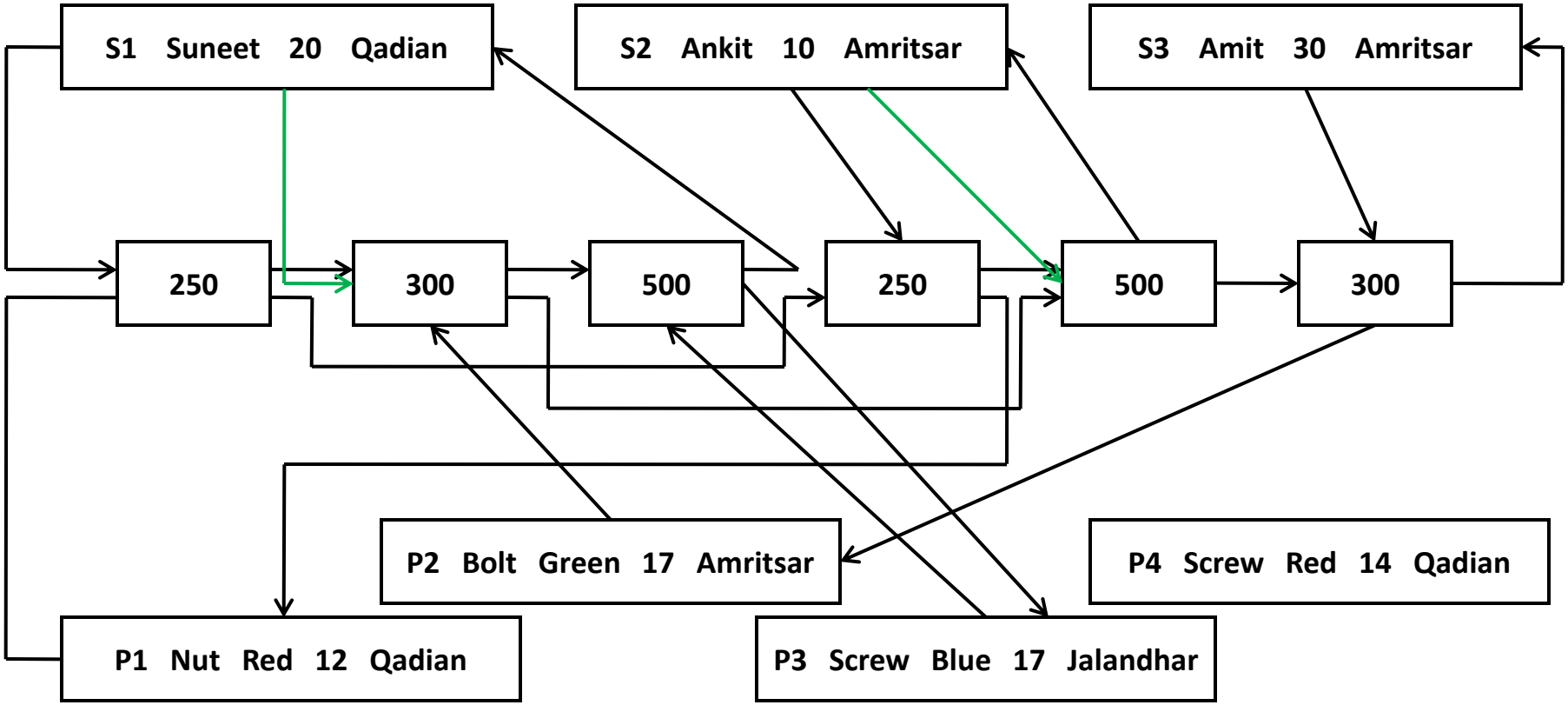
Operations on Network Model

- ***Update Operation:*** Unlike hierarchical model, in a network model updating a record is a much easier process. We can change the city of S1 from Qadian to Jalandhar without search or inconsistency problems because the city for S1 appears at just one place in the network model. Similarly same operation is performed to change the any attribute of part.

Operations on Network Model

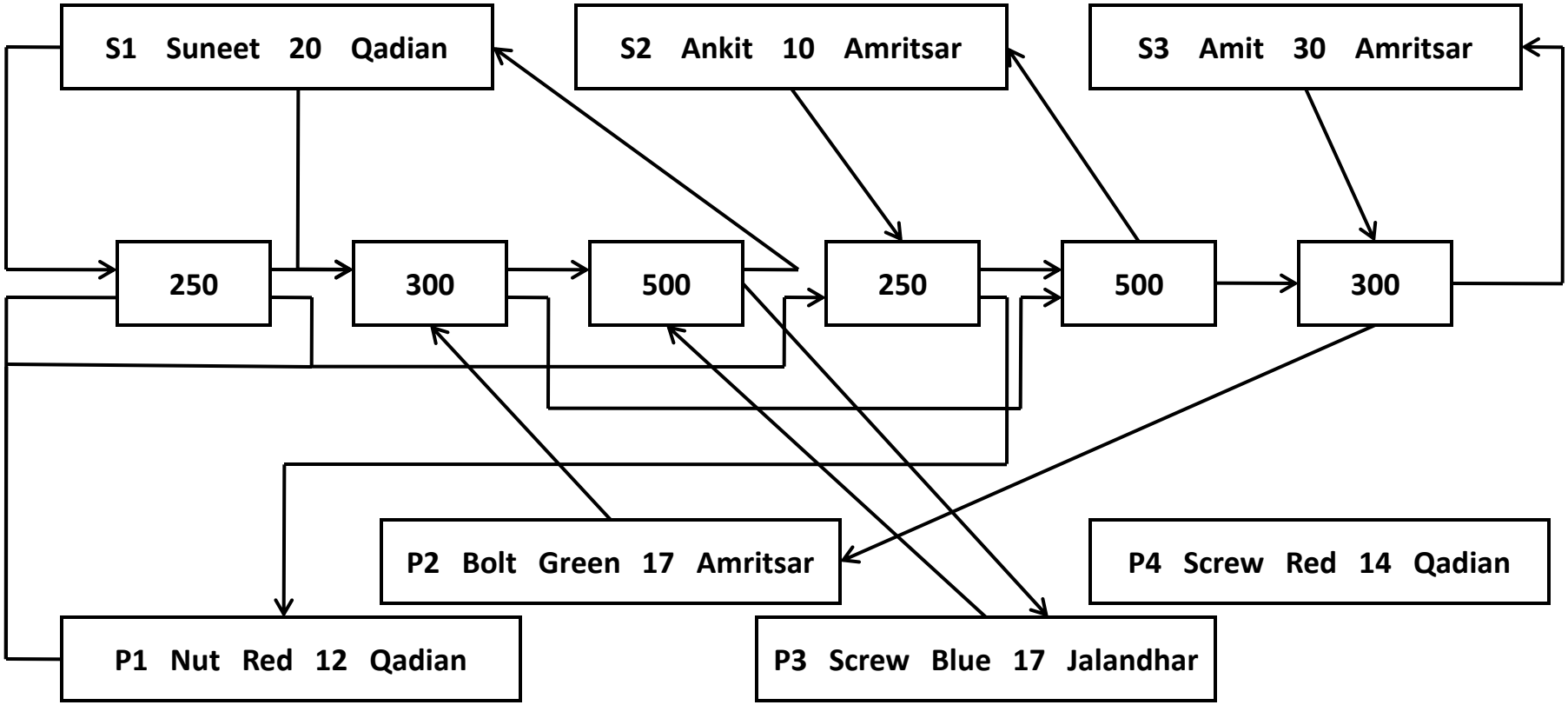
- ***Delete Operation:*** If we wish to delete the information of any part say P1, then that record occurrence can be deleted by removing the corresponding pointers and connectors, without affecting the supplier who supply that part i.e. P1, the model is modified. Similarly same operation is performed to delete the information of supplier.

Operations on Network Model



Operations on Network Model

- **E.g., if supplier S1 stops the supply of part P1 with 250 quantity the model is modified without affecting P1 and S1 information.**



Operations on Network Model

- ***Retrieval Operation:*** Record retrieval method for network model are symmetric but complex. In order to understand this consider the following example queries:

Query 1. Find supplier number for suppliers who supply part P2

Solution: In order to retrieve the required information, first we search for the required part i.e. P2 we will get only one occurrence of P2 from the entire database. Then a loop is constructed to visit each connector under this part i.e. P2. Then for each connector we check the supplier over that connector and supplier number for the concerned supplier record occurrence is printed as shown in below algorithm.

Operations on Network Model

Query 2. Find part number for parts supplied by supplier S2.

- First we search for the required supplier i.e. S2 and we will get only one occurrence of S2 from the entire database. Then a loop is constructed to visit each connector under this supplier i.e. S2. Then for each connector we check the part over that connector and part number is printed for that part record.

Advantages of Network Model

- ***Conceptual Simplicity:*** Just like the hierarchical model, the network model is also conceptually simple and easy to design.
- ***More Relationship Types:*** The network model can handle the 1:N and N:N relationships, which is a real help in modeling the real life situations.
- ***Ease of Data Access:*** Data access is easier and flexible than the hierarchical model.

Advantages of Network Model

- ***Data Integrity:*** The network model does not allow a member to exist without an owner. Thus a user must first define the owner record and then the member record. This ensures the data integrity.
- ***Data Independence:*** The network model is better than the hierarchical model in isolating the programs from the complex physical storage details.

Disadvantages of Network Model

- ***System Complexity:*** All the records are maintained using pointers and hence the whole database structure becomes very complex.
- ***Operational Anomalies:*** The network model's insertion, deletion and updating operations of any record require large number of pointer adjustments, which makes it very complex and complicated.
- ***Absence of Structural Independence:*** Since the data access method in the network database model is a navigational system, making structural changes to the database is very difficult in most cases and impossible in some cases. If changes are made to the database structure then all the application programs need to be modified before they can access data.

Relational Model

- Relational model stores data in the form of tables. The relational model consists of three major components:

1. The set of relations and set of domains that defines the way data can be represented
2. Integrity rules that defines the procedure to protect the data
3. The operations that can be performed on data

Operations in Relational Model

- *Insert Operation:* Suppose we wish to insert the information of supplier who does not supply any part, can be inserted in S table without any anomaly.
- Similarly if we wish to insert information of a new part that is not supplied by any supplier can be inserted into a P table. So, we can say that insert operations can be performed in all the cases without any anomaly.

Operations in Relational Model

- ***Update Operation:*** Suppose supplier S1 has moved from Qadian to Jalandhar. In this case we need to make changes in the record so that the supplier table is up-to-date. Since supplier number is the primary key in the S (supplier) table, so there is only a single entry of S1, which needs a single update and problem of data inconsistencies would not arise. Update operation in relational model is very simple and without any anomaly.
- ***Delete Operation:*** Suppose if supplier S3 stops the supply of part P2, then we have to delete the shipment connecting part P2 and supplier S3 from shipment table SP. This information can be deleted from SP table without affecting the details of supplier of S3 in supplier table and part P2 information in part table. We can delete the information of parts in P table and the information of suppliers in S table and in SP table similarly.

Operations in Relational Model

- ***Record Retrieval:*** Record retrieval is easy:

Query1: Find the supplier numbers for suppliers who supply part P2.

- Solution: In order to get this information we have to search the information of part P2 in the SP table (shipment table). For this a loop is constructed to find the records of P2 and on getting the records, corresponding supplier numbers are printed.

Query2: Find part numbers for parts supplied by supplier S2.

- Solution: In order to get this information we have to search the information of supplier S2 in the SP table (shipment table). For this a loop is constructed to find the records of S2 and on getting the records corresponding part numbers are printed.

Advantages of Relational Model

- ***Structural Independence:*** This model has structural independence. Since it is possible to make changes to the database structure without affecting the DBMS's capability to access data.
- ***Conceptual Simplicity:*** Since records are maintained in the tables and tables are used to create the database, it is conceptually simple and easy to use.

Advantages of Relational Model

- ***Design, Implementation, Maintenance and Usage Ease:*** Design, implementation and maintenance is easy when we are using the tables to hold data.
- ***Ad hoc Query Capability:*** The query language of the relational database models is Structured Query Language (SQL) which makes ad hoc queries a reality. It is very powerful, flexible and easy-to-use.

Disadvantages of Relational Model

- The drawbacks of the relational database systems could be avoided if proper corrective measures are taken. The drawbacks are not because of the shortcomings in the database model, but the way it is being implemented.
  - **Hardware Overheads**
  - **Ease of design can lead to Bad Design**
  - **'Information Island' Phenomenon**

Comparison of data models

Hierarchical Model	Network Model	Relational Model
1. We cannot insert the information of a child who does not have any parent	It does not suffer from any insert anomaly	It does not suffer from any insert anomaly
2. The deletion of parent results in deletion of child record	It does not suffer from delete anomaly	Since, the information is stored in different tables, this model is free from delete anomaly
3. There may be problems of inconsistency during the update operation	It does not suffer from update anomaly	It does not suffer from update anomaly because it removes the redundancy of data through normalization
4. Retrieve algorithms used for hierarchical models are complex and asymmetric	It uses the retrieve algorithms that are complex and symmetric	Retrieve algorithms of relational model are simple and symmetric

Database Constraints

BY: Richa Jain

Database Integrity Constraints

- A *constraint* is a rule that the database manager enforces.
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number

Database Integrity Constraints

- Integrity rules may be divided into three broad categories:
  - Domain integrity rules
  - Entity integrity rules
  - Referential integrity rules

Domain Integrity

- Domain integrity rules are concerned with maintaining the correctness of attribute values within relations.
- Domain integrity ensures the data values inside a database follow defined rules for values, range, and format.
- A database can enforce these rules using a variety of techniques, including CHECK constraints, UNIQUE constraints, and DEFAULT constraints.

Entity Integrity

- Entity integrity ensures each row in a table is a uniquely identifiable entity. We can apply entity integrity to a table by specifying a PRIMARY KEY constraint.
- Entity Integrity can be enforced through indexes, UNIQUE constraints and PRIMARY KEY constraints.

Entity Integrity

- Entity Integrity ensures two properties for primary keys:
  - The primary key for a row is unique; it does not match the primary key of any other row in the table.
  - The primary key is not *null*, no component of the primary key may be set to *null*.
- The uniqueness property ensures that the primary key of each row uniquely identifies it; there are no duplicates.
- The second property ensures that the primary key has meaning, has a value; no component of the key is missing.
- The system enforces Entity Integrity by not allowing operations (INSERT, UPDATE) to produce an invalid primary key. Any operation that creates a duplicate primary key or one containing *nulls* is rejected.

Referential integrity

- Referential integrity rules are concerned with maintaining the correctness and consistency of relationships between relations.
- *Referential integrity* is the state of a database in which all values of all foreign keys are valid.
- We can apply referential integrity using a FOREIGN KEY constraint and CHECK constraints.

Referential integrity

- A table in which a referential constraint and a foreign key are defined is called a *referencing table*, while a table that is referenced from a referencing table with a foreign key is called a *referenced table*.
- In a referenced table, a primary key that is referenced by the foreign key must be pre-defined.

Referential integrity

- The rules are:
  1. We can't delete a record from a primary table if matching records exist in a related table.
  2. We can't change a primary key value in the primary table if that record has related records.
  3. We can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
  4. However, we can enter a Null value in the foreign key, specifying that the records are unrelated.

Functional Dependency & Normalization

BY: Richa Jain

Dependency

- A dependency occurs in a database when information stored in the same database table uniquely determines other information stored in the same table.

Functional Dependency

- A functional dependency is defined as a constraint between two sets of attributes in a relation from a database.

- Given a relation *R*, a set of attributes *X* in *R* is said to **functionally determine** another attribute *Y*, also in *R*, (written $X \rightarrow Y$) if and only if each *X* value is associated with at most one *Y* value.

- A *Functional Dependency* describes a relationship between *attributes* within a single relation.

- An attribute is *functionally dependent* on another if we can use the value of one attribute to determine the value of another.

- We use the arrow symbol → to indicate a functional dependency. X → Y is read *X functionally determines Y*

In other words….

X is the *determinant set* and *Y* is the *dependent attribute*. Thus, given a tuple and the values of the attributes in *X*, one can determine the corresponding value of the *Y* attribute.

Example

Employee

SSN	Name	JobType	DeptName
557-78-6587	Lance Smith	Accountant	Salary
214-45-2398	Lance Smith	Engineer	Product

Note: Name is functionally dependent on SSN because an employee's name can be uniquely determined from their SSN. Name does not determine SSN, because more than one employee can have the same name..

Functional Dependence (FD)

The following table illustrates A → B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Functional Dependence (FD)

- The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

Since for A = 3 there is associated more than one value of B.

Functional dependency can also be defined as follows:

An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Fully Functional Dependency

Fully Functional Dependence(FFD) is defined as Attribute Y is FFD on attribute X , if it is FD on X and not FD on any proper subset of X. For example, in relation Supplier, different cities may have the same status. It may be possible that cities like Amritsar, Jalandhar may have the same status 10.

So, the City is not FD on Status

But, the combination of Sno,Status can give only one corresponding City because Sno is unique. Thus,

(Sno, Status) → City

It means city is FD on composite attribute (Sno,Status) however City is not fully functional dependent on this composite attribute,

Example

- **Supplier Records**

S_No.	Name	Status	City
S1	Suneet	20	Qadian
S2	Ankit	30	Amritsar
S3	Amit	30	Amritsar

- **Part Records**

P_No.	Name	Color	Weight	City
P1	Nut	Red	12	Qadian
P2	Bolt	Green	17	Amritsar
P3	Screw	Blue	17	Jalandhar
P4	Screw	Red	14	Qadian

- **Shipment Records**

S_No.	P_No.	Qty.
S1	P1	250
S1	P2	300
S1	P3	500
S2	P1	250
S2	P2	500
S3	P2	300

Candidate Functional Dependency (CFD)

- A candidate functional dependency is functional dependence that includes all attributes of the table. It should also be noted that a well-formed dependency diagram must have at least one candidate functional dependency and that there can be more than one candidate functional dependence for a given dependency diagram.

Primary Functional Dependency (PFD)

- PFD is a candidate functional dependency that is selected to determine the primary key. The determinant of PFD is the primary key of the relational database table. Each dependency diagram must have one and only one primary functional dependency. If a relational database table has only one candidate functional dependency, then it automatically becomes the primary functional dependency.

Primary Functional Dependency (PFD)

- Once the primary key has been determined, there will be three possible types of functional dependencies:
  - **A → B, A key attribute functionally determines a non-key attribute.**
  - **A → B, A non-key attribute functionally determines a non-key attribute.**
  - **A → B, A non-key attribute functionally determines a key attribute.**

Primary Functional Dependency (PFD)

- A *Partial Functional Dependency* is a functional dependency where the determinant consists of key attributes, but not the entire primary key, and the determined consists of non-key attributes.
- A *Transitive Functional Dependency* is a functional dependency where the determinant and the determined both consists of non-key attributes.

Primary Functional Dependency (PFD)

- A *Boyce-Codd Functional Dependency* is a functional dependency where the determinant consists of non-key attributes and the determined consists of key attributes.
- A *Join Dependency* exist if a relation R is equal to the join of the projections X, Y, …., Z. where X, Y, … Z are projections of R.

Primary Functional Dependency (PFD)

- A *Multi-Value Dependency (MVD)* occurs when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and C are multi-value facts about A, which is represented as A →→ B and A →→ C, then multi value dependency exit only if B and C are independent on each other.

Trivial Functional Dependency

- **Trivial:** If an FD X → Y holds where Y subset of X, then it is called a trivial FD. Trivial FDs always hold.

- **Non-trivial:** If an FD X → Y holds where Y is not subset of X, then it is called non-trivial FD.

- **Completely non-trivial:** If an FD X → Y holds where x intersect Y = Φ, is said to be completely non-trivial FD.

Keys

- key is a set of attributes that uniquely identifies an entire tuple, a functional dependency allows us to express constraints that uniquely identify the values of certain attributes.

- However, a candidate key is always a determinant, but a determinant doesn't need to be a key.

Closure

- Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as ***F+***) is the set of all functional dependencies that may be logically derived from *F*.

- ***F+***, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F*.

- The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

Axioms

Developed by Armstrong in 1974, there are six rules (axioms) that all possible functional dependencies may be derived from them.

Axioms Cont…

1. *Reflexivity Rule* --- If *X* is a set of attributes and *Y* is a subset of *X*, then $X \rightarrow Y$ holds.

   each subset of *X* is functionally dependent on *X*.

2. *Augmentation Rule* --- If $X \rightarrow Y$ holds and *W* is a set of attributes, then $WX \rightarrow WY$ holds.

3. *Transitivity Rule* --- If $X \rightarrow Y$ and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ holds.

Derived Theorems from Axioms

4. *Union Rule* --- If $X \rightarrow Y$ and $X \rightarrow Z$ holds, then $X \rightarrow YZ$ holds.
5. *Decomposition Rule* --- If $X \rightarrow YZ$ holds, then so do $X \rightarrow Y$ and $X \rightarrow Z$.
6. *Pseudotransitivity Rule* --- If $X \rightarrow Y$ and $WY \rightarrow Z$ hold then so does $WX \rightarrow Z$.

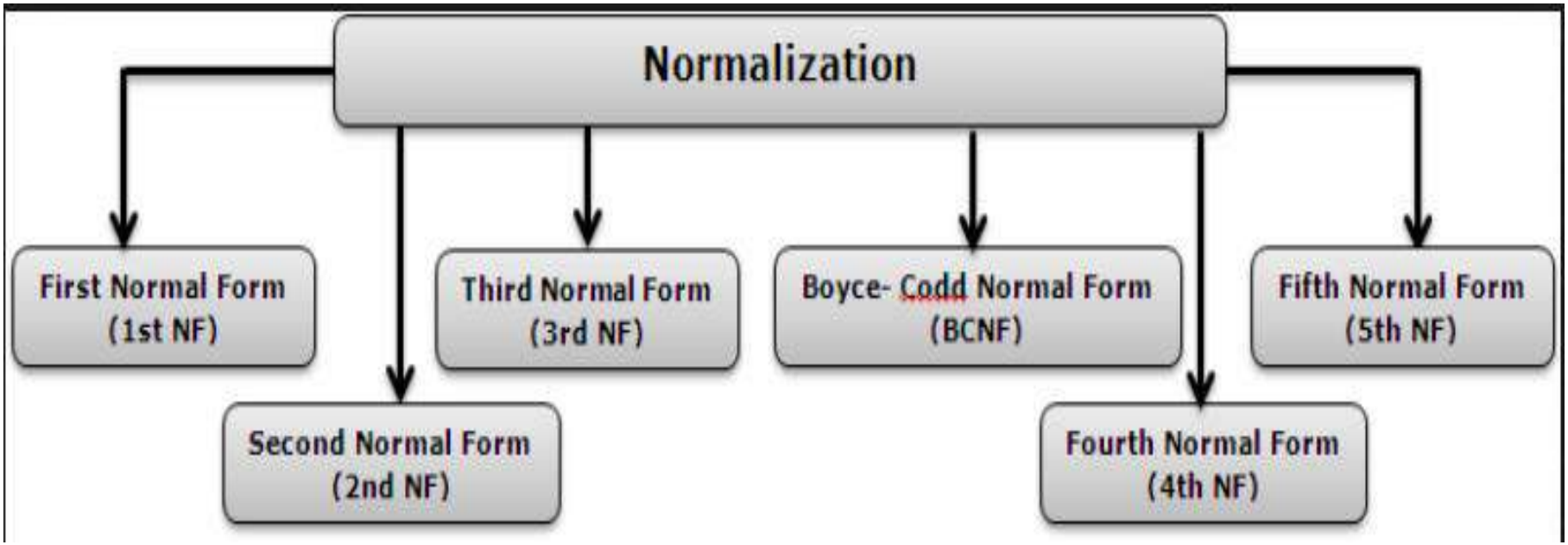
Normalization

- Data normalization is a technique of organizing the data in database.
- Normalization of data can be defined as a process during which redundant relation schemas are decomposed by breaking up their attributes into smaller relation schemas that process desirable properties.

Objectives of Normalization

- To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- To obtain powerful relational retrieval algorithms based on a collection of primitive relational operators.
- To free relations from undesirable insertion, update and deletion anomalies.
- To reduce the need for restructuring the relations as new data types are introduced.
- To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

Normal Forms



First Normal Form

- A relation is said to be in 1NF if and only if every cell/entry of the relation has at most a single value. In other words “a relation is in 1NF if and only if all underlying domains contain atomic values or single value only.”
- The objective of normalizing a table is to remove its repeating groups and ensure that all entries of the resulting table have at most a single value.

Example: Unnormalized table

Course Code	Course Name	Teacher Name	Roll No	Name	System Used	Hourly Rate	Total Hours
C1	Visual Basic	ABC	100	A1	P – I	20	7
			101	A2	P – II	30	3
			102	A3	Celeron	10	6
			103	A4	P – IV	40	1
C2	Oracle & Dev	DEF	100	A1	P – I	20	7
			104	A5	P – III	35	3
			105	A6	P – II	30	1
			101	A2	P – II	30	2
C3	C++	KJP	106	A7	P – IV	40	3
			107	A8	P – IV	40	2
			108	A9	P – I	20	1
C4	Java	Kumar	109	A10	Cyrix	20	2

Approaches to normalize table

- In general, there are two basic approaches to normalize tables.

First Approach: Flattening the table

The first approach known as "flattening the table" removes repeating groups by filling in the "missing" entries of each "incomplete row" of the table with copies of their corresponding non-repeating attributes. The following example illustrates this.

STUDENT (*Flattening*) (Normalized Table)

Course Code	Course Name	Teacher Name	Roll No	Name	System Used	Hourly Rate	Total Hours
C1	Visual Basic	ABC	100	A1	P – I	20	7
C1	Visual Basic	ABC	101	A2	P – II	30	3
C1	Visual Basic	ABC	102	A3	Celeron	10	6
C1	Visual Basic	ABC	103	A4	P – IV	40	1
C2	Oracle & Dev	DEF	100	A1	P – I	20	7
C2	Oracle & Dev	DEF	104	A5	P – III	35	3
C2	Oracle & Dev	DEF	105	A6	P – II	30	1
C2	Oracle & Dev	DEF	101	A2	P – II	30	2
C3	C++	KJP	106	A7	P – IV	40	3
C3	C++	KJP	107	A8	P – IV	40	2
C3	C++	KJP	108	A9	P – I	20	1
C4	Java	Kumar	109	A10	Cyrix	20	2

Approaches to normalize table

Second Approach: Decomposition of the table

The second approach for normalizing a table requires that the table be decomposed into two new tables that will replace the original table.

However, before decomposing the original table it is necessary to identify an attribute or a set of its attributes that can be used as table identifiers.

Rule of decomposition

- One of the two tables contains the table identifier of the original table and all the non-repeating attributes.

- The other table contains a copy of the table identifier and all the repeating attributes.

To normalize the STUDENT table we need to replace it by two new tables. The first table COURSE contains the table identifier and the non-repeating groups.These attributes are Course_Code (the table identifier), Course_Name, and Teacher_Name.

The second table contains the table identifier and all the repeating groups. Therefore, the attributes of COURSE_STUDENT table are Course_Code, Rollno, Name, System_Used, Hourly_Rate and Total_Hrs.

STUDENT (Decomposition) (Normalized Table)

COURSE

Course Code	Course Name	Teacher Name
C1	Visual Basic	ABC
C2	Oracle & Dev	DEF
C3	C++	KJP
C4	Java	Kumar

COURSE_STUDENT

PK = (Course_code, RollNo)

Course Code	Roll No	Name	System Used	Hourly Rate	Total Hours
C1	100	A1	P – I	20	7
C1	101	A2	P – II	30	3
C1	102	A3	Celeron	10	6
C1	103	A4	P – IV	40	1
C2	100	A1	P – I	20	7
C2	104	A5	P – III	35	3
C2	105	A6	P – II	30	1
C2	101	A2	P – II	30	2
C3	106	A7	P – IV	40	3
C3	107	A8	P – IV	40	2
C3	108	A9	P – I	20	1
C4	109	A10	Cyrix	20	2

Anomalies in 1NF Relations

- Redundancies in 1NF relations lead to a variety of data anomalies.

1. Insert anomalies:

We cannot insert the information about the student until he joins any course e.g. we cannot store information about the roll no 110 until he join any course, similarly we are unable to store the information about the course until there is a student who enroll into that course.

These anomalies occur because course_code, rollno is the composite key and we cannot insert null in any of these two attributes.

Anomalies in 1NF Relations

2. Update anomalies:

This relation is also susceptible to update anomalies because the course in which a student studies may appear many times in the table. If a teacher moves to another course, we are now faced with two problems: we either search the entire table looking for that teacher and update his or her course_code value or we miss one or more tuples of that STUDENT and end up with an inconsistent database. For small tables, this type of anomaly may not seem to be much of a problem.

But for larger tables this may cause the problem of inconsistency.

Anomalies in 1NF Relations

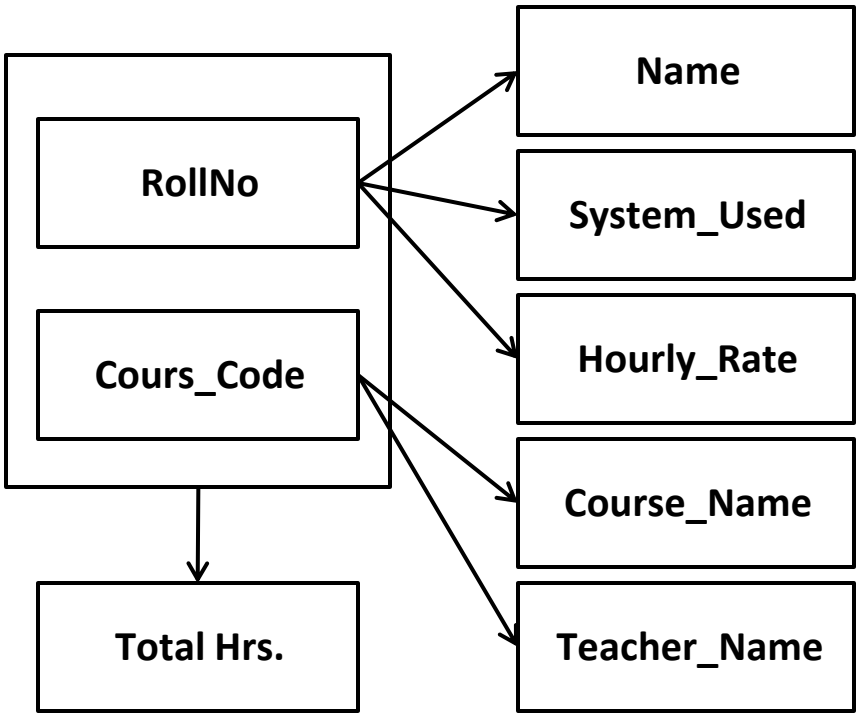
3. Delete anomalies:

This relation experiences deletion anomalies whenever we delete the last tuple of a particular student. In this case, we not only delete the course information that connects that student to a particular course, but also lose other information about the system on which this student works.

Let us consider, the case where we have to delete the information of student having rollno 109, then we also lose the information about course_code C4 . Also if we have to delete the information of java course we lose the information about the student Kumar.

Second Normal Form (2NF)

- A relation *R* is in 2NF if and only if it is in 1NF and every non-key attribute is fully functional dependent on the primary key.



Functional Dependence Diagram

Second Normal Form (2NF)

- A resultant database of 1NF *Course_Code* does not satisfy above rule, because non-key attributes Name, System_Used and Hourly_Rate are not fully dependent on the primary key (*Course_Code, Rollno*) because Name, *System_Used* and *Hourly_Rate* are functional dependent on *Rollno* and *Rollno* is a subset of the primary key so it does not hold the law of fully functional dependence.

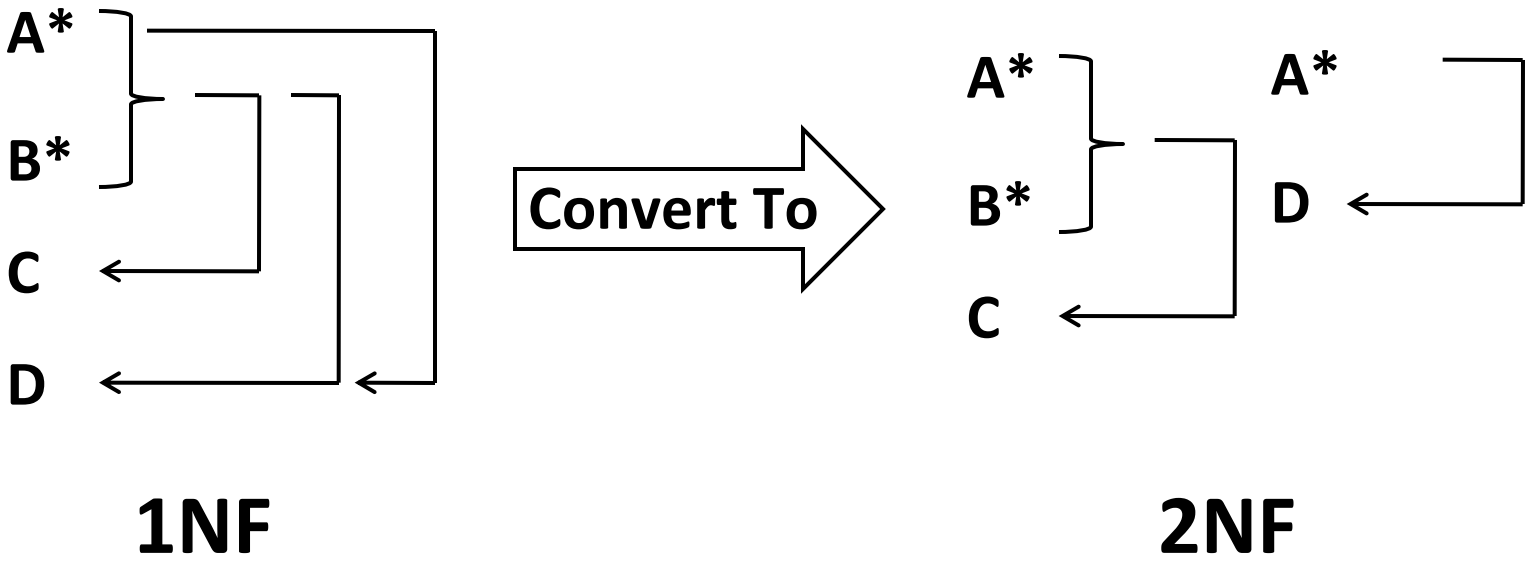
Rule to convert 1NF to 2NF

- Consider a relation where a primary key consists of attributes A and B. These two attributes determine all other attributes.
- Attribute C is fully dependent on the key.
- Attribute D is partially dependent on the key because we only need attribute A to functionally determine it.
- Attributes C and D are non-key attributes.

Rule to convert 1NF to 2NF

- The rule is to replace the original relation by two new relations. The first new relation has three attributes: A, B and C. The primary key of this relation is (A, B) i.e. the primary key of the original relation. The second relation has A and D as its only two attributes. Observe that attribute A has been designated, as the primary key of the second relation and that attribute D is now fully dependent on the key.

Rule to transform 1NF to 2NF



Second Normal Form (2NF)

HOURS_ASSIGNED

Course Code	Roll No	Total Hours
C1	100	7
C1	101	3
C1	102	6
C1	103	1
C2	100	7
C2	104	3
C2	105	1
C2	101	2
C3	106	3
C3	107	2
C3	108	1
C4	109	2

STUDENT_SYSTEM_CHARGE

Roll No	Name	System Used	Hourly Rate
100	A1	P – I	20
101	A2	P – II	30
102	A3	Celeron	10
103	A4	P – IV	40
100	A1	P – I	20
104	A5	P – III	35
105	A6	P – II	30
101	A2	P – II	30
106	A7	P – IV	40
107	A8	P – IV	40
108	A9	P – I	20
109	A10	Cyrix	20

COURSE

Course Code	Course Name	Teacher Name
C1	Visual Basic	ABC
C2	Oracle & Dev	DEF
C3	C++	KJP
C4	Java	Kumar

Removal of Anomalies of 1NF Relations

- ***Insert Anomalies***: It is now possible to insert the information about the student who does not join any course e.g. we can store the information about the Rollno 110 who does not join any course in STUDENT_SYSTEM_CHARGE database. Similarly now we are able to store the information about the course which has no enrolled student e.g. we can store that C1 course is of Visual Basic in COURSE table.

Removal of Anomalies of 1NF Relations

- ***Update Anomalies***: Now, it is possible to change the teacher for a particular course in the COURSE table through a single modification. So, no data inconsistency will arise.
- ***Delete Anomalies***: in the revised structure, we can delete the information of student having Rollno 109 without losing the information about his course i.e. C4.

Anomalies in 2NF

- Relations in 2NF are still subject to data anomalies. Let us assume that the system on which a student works functionally determines the hourly rate charged from the student i.e.

$$System\_Used \rightarrow Hourly\_Rate$$

- Due to this fact the anomalies will occur in case of 2NF.

Anomalies in 2NF Relations

- ***Insert Anomalies***: Insertion anomalies occur in the *Student_System_Charge* relation. For example, consider a situation where we would like to set in advance the rate to be charged from the students for a particular system. We can not insert this info. Until there is a student assigned to that type of system because Rollno is the primary key for this relation and we can not insert the null value into it.

Anomalies in 2NF

- ***Update Anomalies***: Update anomalies will also occur in the *Student_System_Charge* relation because there may be several students which are working on the same type of the system. If the Hourly_Rate for a particular system changes, we need to make sure that the corresponding rate is changed for all the students that work on that type of system. Otherwise, the database may end up in an inconsistent state.

Anomalies in 2NF

- ***Delete Anomalies***: Delete anomalies will also occur in the *Student_System_Charge* relation. This type of anomaly occurs whenever we delete the tuple of a student who happens to be the only student left which is working on a particular system. In this case, we will also lose the information about the rate that we charge for that particular system.

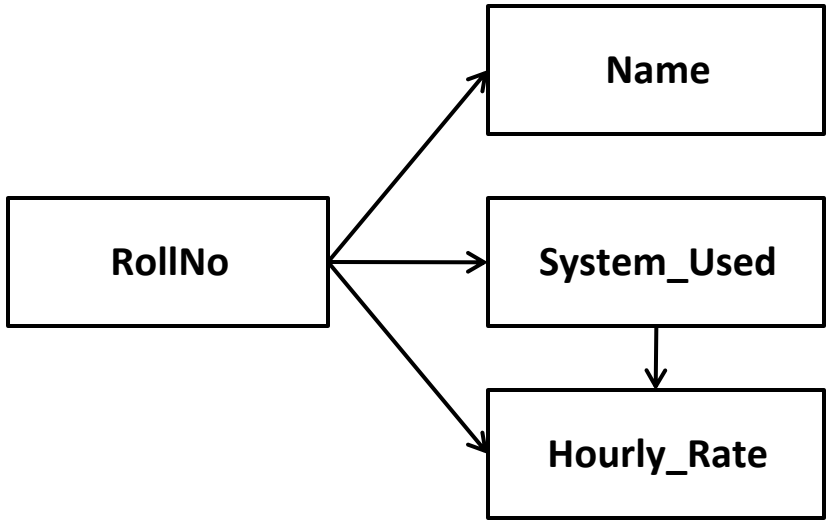
Prime and nonprime attributes

- An attribute of a relation schema R is called Prime attribute if it is a member of some candidate key of R.
- An attribute is called non prime if it is not a prime attribute i.e. it is not the member of any candidate key.

Third Normal Form (3NF)

- A relation *R* is in 3NF if and only if the following conditions are satisfied simultaneously:
  - ***R* is already in 2NF**
  - **No nonprime attribute functionally determines any other nonprime attribute Or No nonprime attribute is transitively dependent on the key**
- The objective of transforming relations into 3NF is to remove all transitive dependencies.

Third Normal Form (3NF)



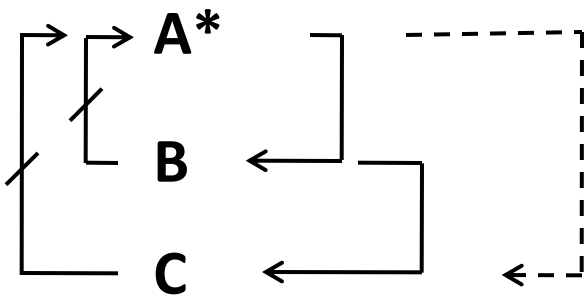
Functional Dependence Diagram

RollNo → System_Used

System_Used → Hourly_Rate

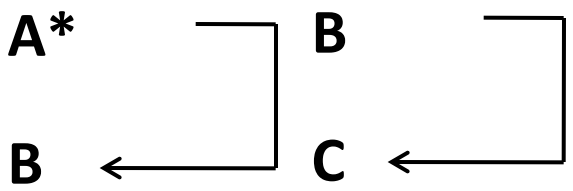
It Means RollNo → Hourly_Rate

Rule to Resolve Transitivity Dependence



2NF

Convert To



3NF

Third Normal Form (3NF)

STUDENT_SYSTEM_CHARGE

Roll No	Name	System Used	Hourly Rate
100	A1	P – I	20
101	A2	P – II	30
102	A3	Celeron	10
103	A4	P – IV	40
100	A1	P – I	20
104	A5	P – III	35
105	A6	P – II	30
101	A2	P – II	30
106	A7	P – IV	40
107	A8	P – IV	40
108	A9	P – I	20
109	A10	Cyrix	20

2NF

Convert To

STUDENT_SYSTEM

Roll No	Name	System Used
100	A1	P – I
101	A2	P – II
102	A3	Celeron
103	A4	P – IV
100	A1	P – I
104	A5	P – III
105	A6	P – II
101	A2	P – II
106	A7	P – IV
107	A8	P – IV
108	A9	P – I
109	A10	Cyrix

3NF

CHARGES

System Used	Hourly Rate
Celeron	10
Cyrix	20
P – I	20
P – II	30
P – III	35
P – IV	40

Removal of Anomalies of 2NF Relations

- **Insert Anomalies:** In the revised structure of STUDENT_SYSTEM and CHARGES, it is possible to insert in advance the rate to be charged from the students for a particular system.
- **Update Anomalies:** If the Hourly_Rate for a particular system changes, we need only to change a single record in CHARGES database for that particular system.

Removal of Anomalies of 2NF Relations

- **Delete Anomalies:** We delete the tuple of a student who happens to be the only student left which is working on a particular system without losing the information about the rate that we charge for that particular system.

Anomalies in 3NF

- The relations in 3NF are suceptible to data anomalies particularly when the relations have two overlapping candidate keys or when a non-prime attribute functionally determines a prime attribute. Let us consider the example which illustrate these anomalies:

Anomalies in 3NF

- We can take a case of Supplier_Part Table having following attributes:

  Supplier_Part(Sno, Sname, Pno, Qty)

  Lets suppose that Sname is unique for each Sno as shown below:

Sno	Sname	Pno	Qty
S1	Rahat	P1	300
S2	Raju	P2	200
S1	Rahat	P3	100
S2	Raju	P1	200

Anomalies in 3NF

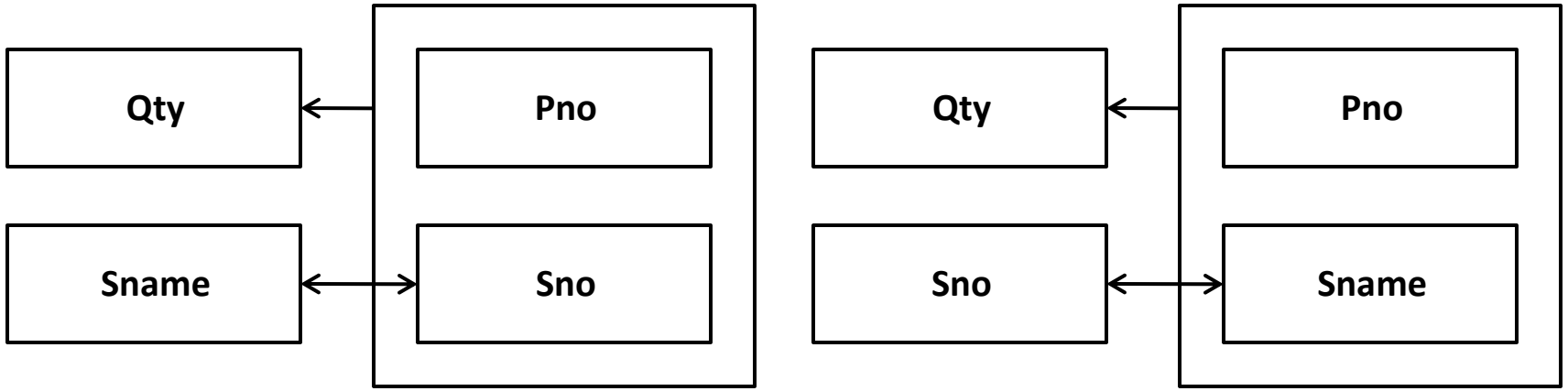
- This relation has two candidate keys: (Sno, Pno) and (Sname, Pno) that overlap on the attribute Pno. The relation is in 3NF because there is single nonprime attribute.
- The relation is susceptible to **update anomalies** e.g. if one of the supplier changes its name, then we have to make multiple changes which is equal to the number part supplied by that particular supplier.

Boyce-Codd Normal Form

- Boyce-Codd Normal Form (BCNF) is used to eliminate the anomalies of 3NF.
- BCNF states that a relation R is in BCNF if and only if every determinant is a candidate key.
- Here determinant is a simple attribute or composite attribute on which some other attribute is fully functionally dependent
- E.g. (Sno, Pno) → Qty, here (Sno, Pno) is a composite determinant. Sno → Sname, here Sno is simple attribute determinant.

Boyce-Codd Normal Form

Functional Dependency Diagram of Supplier_Part Relation



FD of the above relations are:

(Sno, Pno) → Qty

(Sname, Pno) → Qty

Sno → Sname

Sname → Sno

Boyce-Codd Normal Form

- Both the relations are in 3NF, because there is only one non-key attribute i.e. Qty and it is FFD and non-transitively dependent on the primary key.
- But Supplier_Part relation is not in BCNF because this relation has four determinants:
  - **(Sno, Pno), (Sname, Pno), (Sno), (Sname)**
- Out of these four determinants (Sno, Pno) and (Sname, Pno) are unique but Sno and Sname determinants are not candidate keys.

Boyce-Codd Normal Form

- In order to make this relation in BCNF, we non-loss decompose this relation in two projections SN (Sno, Sname) and SP (Sno, Pno, Qty).
- SN relation has two determinants Sno, Sname and both are unique.
- SP has one determinant (Sno, Pno) and is also unique.

Decomposition of tables

- Decomposition means dividing a table into more than one table. The main purpose of decomposition is to eliminate redundancy by decomposing a relation into several relations in a higher normal form.
- Types of decomposition:
  - Lossy decomposition
  - Lossless decomposition

Lossy Decomposition

- Lossy decomposition results in the loss of the information. Let R be a relation , decomposition of R is a set of relation schemas (R_1, R_2, R_3....) such that R = R_1 U R_2 U .....U R_n such that each R_i is a subset of R (for i = 1,2…,n)
- For example, For relation R(x,y,z) there can be 2 subsets:

  R1(x,z) and R2(y,z)

If we union R1 and R2, we get R ,i.e, R = R1 U R2

Lossy Decomposition

- The major problem with decomposition is that we may not be able to get the original relation after performing the union of instances of the original relation- results in information loss.

Example : Problem with Decomposition

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

R1

Model Name	Category
a11	Canon
s20	Nikon
a70	Canon

R2

Price	Category
100	Canon
200	Nikon
150	Canon

Example : Problem with Decomposition

R1 U R2

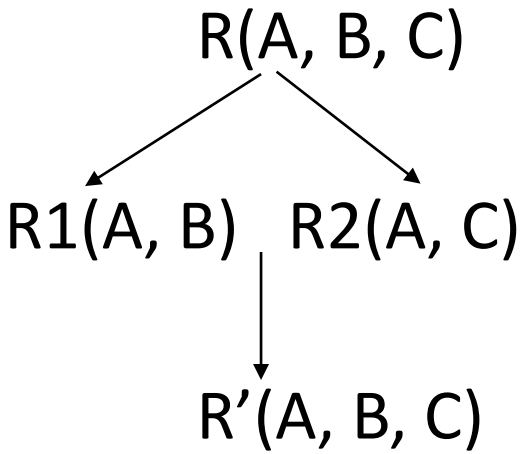
Model Name	Price	Category
a11	100	Canon
a11	150	Canon
s20	200	Nikon
a70	100	Canon
a70	150	Canon

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

Loss-less decomposition

- A decomposition {R1, R2,…, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2,…, Rn produces exactly the relation R.
- A decomposition is lossless if we can recover:



Decompose

Recover

Thus, R' = R

Forth Normal Form (4NF)

- A relation R is in 4NF if and only if the following conditions are satisfied simultaneously:
  - **R is already in 3NF or BCNF.**
  - **If it contains no multi-valued dependencies.**
- Multi-Valued Dependency (MVD)
  - **MVD is the dependency where one attribute value is potentially a 'multi-valued fact' about another.**

Forth Normal Form (4NF)

- MVD can be defined informally as follows:
  - **MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and C are muti-value facts about A, which is represented as A→→B and A→→C ,then muti value dependency exist only if B and C are independent of each other.**

Forth Normal Form (4NF)

- Two things to note about this definition:
  - **Firstly, For a table to contain MVD, it must have three or more attributes.**
  - **Secondly, it is possible to have a table containing two or more attributes which are inter-dependent multi valued facts about another attribute.**
  - **The attributes giving rise to the multi-valued facts must be independent of each other.**

Forth Normal Form (4NF)

Course_Student_Book

Course	S_Name	Text_Book
Physics	Ankit	Mechanics
Physics	Ankit	Optics
Physics	Rahat	Mechanics
Physics	Rahat	Optics
Chemistry	Ankit	Org. Chemistry
Chemistry	Ankit	Inorg. Chemistry
English	Raj	Eng. Literature
English	Raj	Eng. Grammer

MVD exists :

Course →→ S_Name

Course →→ Text_Book

Forth Normal Form (4NF)

- Anomalies of database with MVDs:
  - **If a new student joins the physics course then we have to make two insertions for that student in the database, which is equal to no. of physics text books.**
  - **If the name of the physics textbook is required to change we have the update the no. of records equal to no. of students in physics course.**
  - **If a physics textbook is required to be deleted then we have to delete no. of records.**

Forth Normal Form (4NF)

- To put *Course_Student_Book* relation into 4NF, two separate tables are formed as shown below:

Course_Student

Course	S_Name
Physics	Ankit
Physics	Rahat
Chemistry	Ankit
English	Raj

Course_Student

Course	Text_Book
Physics	Mechanics
Physics	Optics
Chemistry	Org. Chemistry
Chemistry	Inorg. Chemistry
English	Eng. Literature
English	Eng. Grammer

Fifth Normal Form (5NF)

- A relation R is in 5NF if and only if the following conditions are satisfied simultaneously:
  - **R is already in 4NF.**
  - **It cannot be further non-loss decomposed.**
- 5NF is of little practical use to the database designer, but it is of interest from a theoretical point of view.

Fifth Normal Form (5NF)

- In all of the normal forms discussed so far, no loss decomposition was achieved by the decomposing of a single table into two separate tables. No loss decomposition is possible because of the availability of the join operator as part of the relational model. In considering 5NF, consideration must be given to table where non-loss decomposition can only be achieved by decomposition into three or more separate tables.

Fifth Normal Form (5NF)

- Consider the table: *Agent_Company_Product* below, table lists agents, the companies they work for and the products they sell for those companies. The *agents do not necessarily sell all the products supplied by the companies they do business with*.

Agent_Company_Product

Agent	Company	P_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	CDF	Bolt
Raj	ABC	Bolt

Fifth Normal Form (5NF)

- Suppose the table is decomposed into three projections say P1, P2, P3:

P1

Agent	Company
Suneet	ABC
Suneet	CDF
Raj	ABC

P2

Agent	P_Name
Suneet	Nut
Suneet	Screw
Suneet	Bolt
Raj	Bolt

P3

Company	P_Name
ABC	Nut
ABC	Screw
ABC	Bolt
CDE	Bolt

Fifth Normal Form (5NF)

- Apply Natural Join to Projection P1 and P2 over the *Agent* column:

Natural Join of P1 & P2

Agent	Company	P_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	ABC	Bolt*
Suneet	CDE	Nut*
Suneet	CDE	Screw*
Suneet	CDE	Bolt
Raj	ABC	Bolt

- **The resulting table is spurious, since the asterisked row of the table contains incorrect information.**

Fifth Normal Form (5NF)

- Apply Natural Join to Projection P1, P2 and P3 over the *Company* and *P_Name* columns:

Natural Join of (P1 & P2) & P3

Agent	Company	P_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	ABC	Bolt*
Suneet	CDE	Bolt
Raj	ABC	Bolt

- **It is still containing spurious row. It is not simply possible to decompose *Agent_Company_Product* table without losing information.**

Fifth Normal Form (5NF)

- Now Consider the different case where, if an agent for a company and the company makes a product, then *he always sells that product for the company*. Under these circumstances, the *Agent_Company_Product* table is shown below:

Agent_Company_Product

Agent	Company	P_Name
Suneet	ABC	Nut
Raj	ABC	Bolt
Raj	ABC	Nut
Suneet	CDF	Bolt
Suneet	ABC	Bolt

Fifth Normal Form (5NF)

- The assumption being that ABC makes both Nuts and Bolts and that CDF makes Bolts only. This table can be decomposed into its three projections without loss of information as shown below:

P1

Agent	Company
Suneet	ABC
Suneet	CDF
Raj	ABC

P2

Agent	P_Name
Suneet	Nut
Suneet	Bolt
Raj	Bolt
Raj	Nut

P3

Company	P_Name
ABC	Nut
ABC	Bolt
CDE	Bolt

Fifth Normal Form (5NF)

- All redundancy is removed, if the natural join of P1 and P2 is taken, the result is:

Natural Join of P1 & P2

Agent	Company	P_Name
Suneet	ABC	Nut
Suneet	ABC	Bolt
Suneet	CDE	Nut*
Suneet	CDE	Bolt
Raj	ABC	Bolt
Raj	ABC	Nut

- **The resulting table is spurious, since the asterisked row of the table contains incorrect information.**

Fifth Normal Form (5NF)

- Now, if this result is joined with P3 over the column '*Company*' and '*P_Name*' the following table is obtained.

Natural Join of (P1 & P2) & P3

Agent	Company	P_Name
Suneet	ABC	Nut
Suneet	ABC	Bolt
Suneet	CDE	Bolt
Raj	ABC	Bolt
Raj	ABC	Nut

- **This is a correct recomposition of the original table and no loss decomposition into the three projections is achieved.**

Fifth Normal Form (5NF)

- If the original table and the table formed, after decomposing the original table into no. of tables and then joining those table together, are identical then the original table violates the 5NF.
- Detecting that a table violates 5NF is very difficult in the practice and for this reason this normal form has little in any practical application.

Steps of Normalization