Database Management System

CHAPTER-17 (Transactions)

- A transaction is a unit of program execution that accesses and possibly updates various data items.

Schedules

Schedule – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

Schedule 1

Let T1 transfer $50 from A to B, and T2 transfer 10% of the balance from A to B.

A serial schedule in which T1 is followed by T2:

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

A serial schedule where T2 is followed by T1:

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Schedule 3

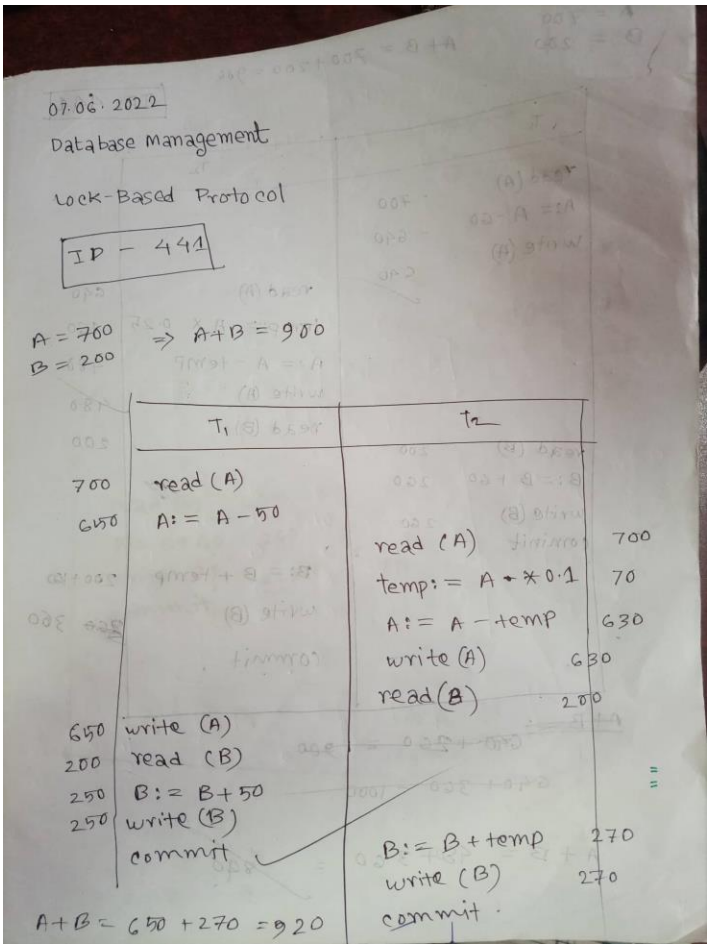
Let T1 and T2 be the transactions defined previously. The following schedule is not a serial schedule, but it is equivalent to Schedule 1:

T_1	T_2
read (A) $A := A - 50$ write (A)	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	
	read (B) $B := B + temp$ write (B) commit

Schedule 3

The following concurrent schedule does not preserve the value of (A + B):

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit



07.06.2022

Database Management

Lock-Based Protocol

ID - 441

A = 700
B = 200 $\Rightarrow$ A+B = 900

	T_1	T_2	
700	read (A)		
650	A := A - 50		
		read (A)	700
		temp := A * 0.1	70
		A := A - temp	630
		write (A)	630
		read (B)	200
650	write (A)		
200	read (B)		
250	B := B + 50		
250	write (B)		
	commit		
		B := B + temp	270
		write (B)	270
		commit.	

A+B = 650 + 270 = 920

	T_1	T_2	
	read (A)		700
	A := A - 60		640
		read (A)	700
		temp := A * 0.25	175
		A := A - temp	525
		write (A)	525
	write (A)		640
	read (B)		200
	B := B + 60		260
	write (B)		260
	commit		
		read (B)	260
		B := B + temp	435
		write (B)	435
		commit.	

A+B = 640 + 435 = 1075

Serve the Sum of

CHAPTER-03

(Modeling Data in the Organization)

E-R Model Constructs

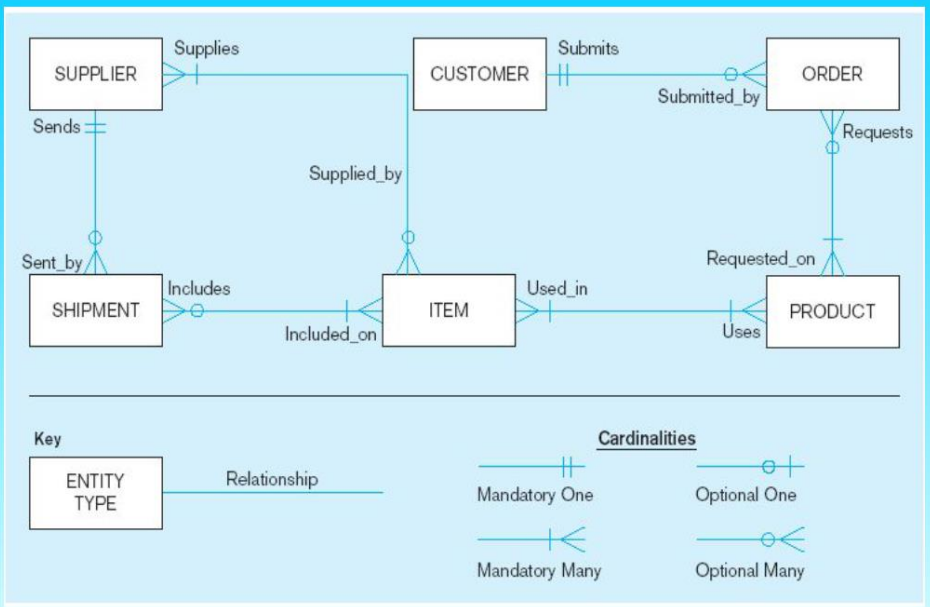
❖ Entities:

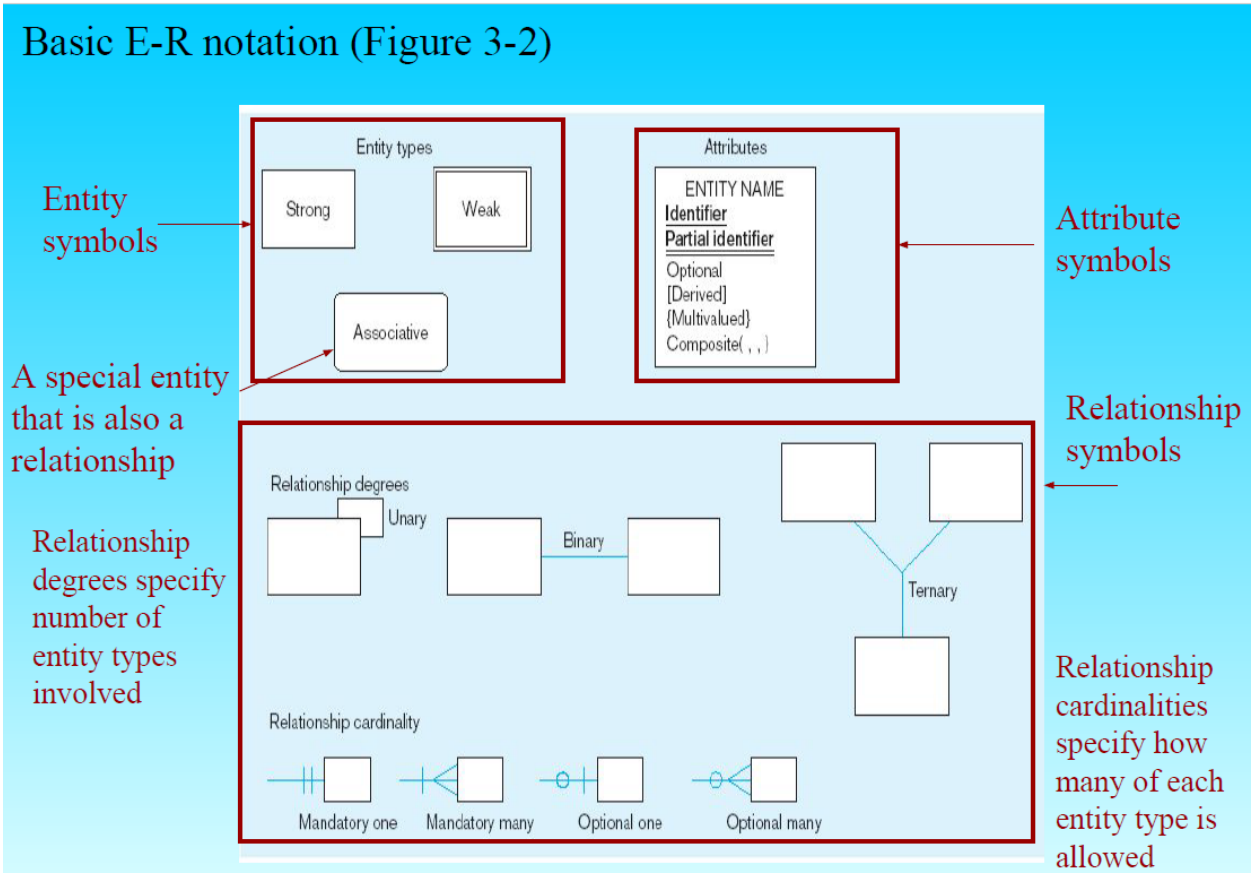
- Entity instance–person, place, object, event, concept (often corresponds to a row in a table)
- Entity Type–collection of entities (often corresponds to a table)

❖ Relationships:

- Relationship instance–link between entities (corresponds to primary key-foreign key equivalencies in related tables)
- Relationship type–category of relationship…link between entity Types

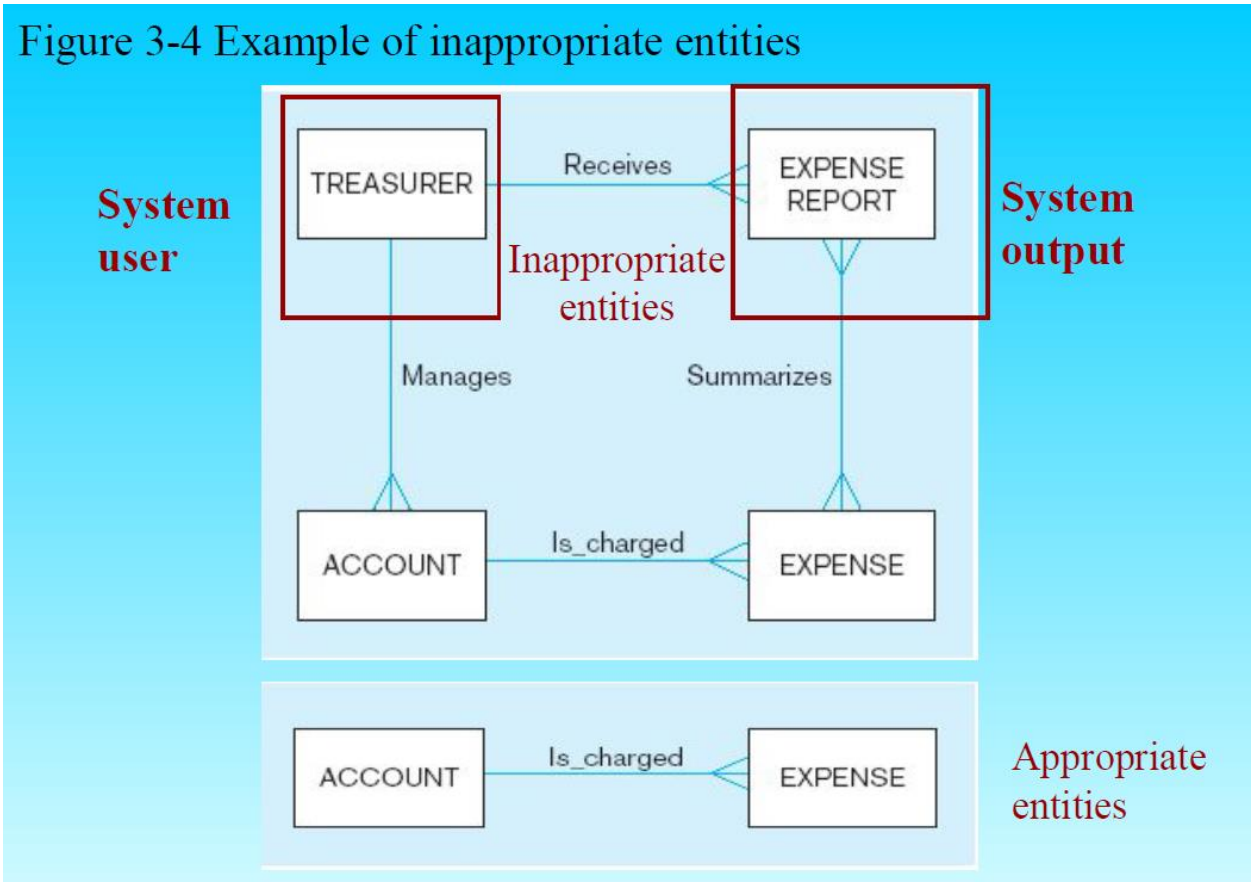
Sample E-R Diagram (Figure 3-1)





What Should an Entity Be?

- **SHOULD BE:**
  - An object that will have many instances in the database
  - An object that will be composed of multiple attributes
  - An object that we are trying to model
- **SHOULD NOT BE:**
  - A user of the database system
  - An output of the database system (e.g., a report)



Attributes

Attribute–property or characteristic of an entity or relationship type

Classifications of attributes:

■ Required versus Optional Attributes

■ Simple versus Composite Attribute

■ Single-Valued versus Multivalued Attribute

■ Stored versus Derived Attributes

■ Identifier Attributes

Identifiers (Keys)

❖ Identifier (Key)–An attribute (or combination of attributes) that uniquely identifies individual instances of an entity type.

■ Simple versus Composite Identifier

■ Candidate Identifier–an attribute that could be a key…satisfies the requirements for being an identifier

Characteristics of Identifiers

■ Will not change in value

■ Will not be null

■ No intelligent identifiers (e.g., containing\ locations or people that might change)

■ Substitute new, simple keys for long, composite keys

Figure 3-7 A **composite** attribute

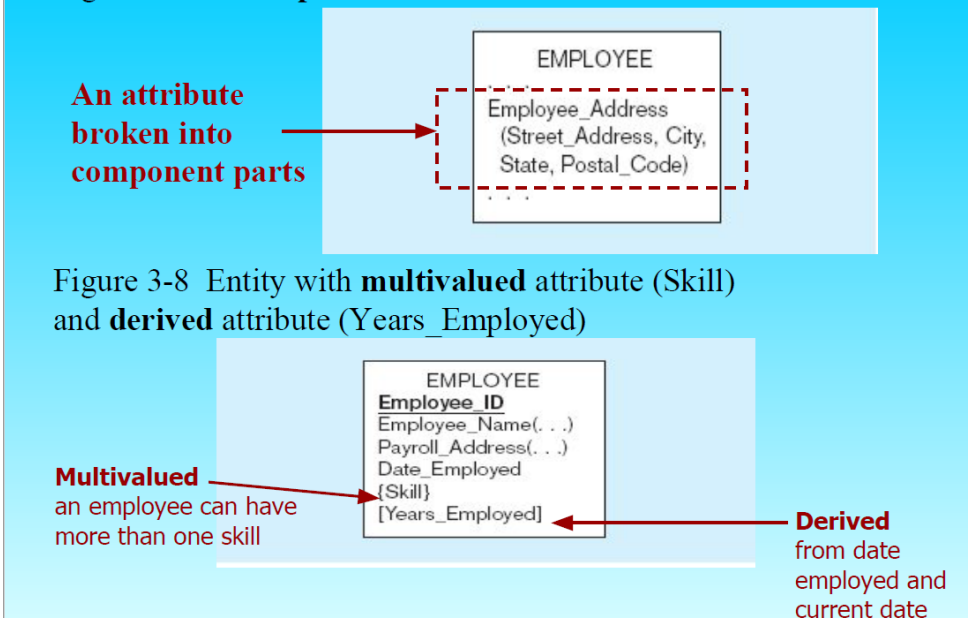
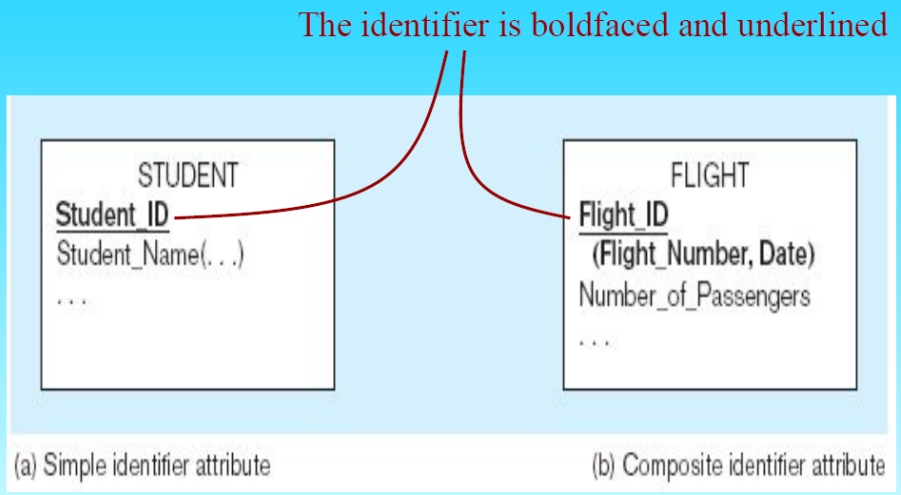


Figure 3-8 Entity with **multivalued** attribute (Skill) and **derived** attribute (Years_Employed)

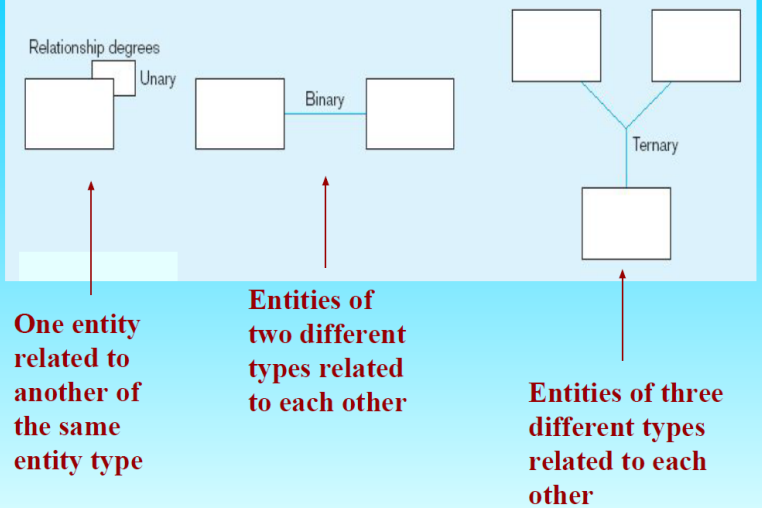
Figure 3-9 Simple and composite identifier attributes



Degree of Relationships

- Degree of a relationship is the number of entity types that participate in it
  - Unary Relationship
  - Binary Relationship
  - Ternary Relationship

Degree of relationships – from Figure 3-2



- Strong entities
  - exist independently of other types of entities

- has its own unique identifier
- identifier underlined with single-line

❖ Weak entity

- dependent on a strong entity (identifying owner)…cannot exist on its own
- does not have a unique identifier (only a partial identifier)
- Partial identifier underlined with double-line
- Entity box has double line

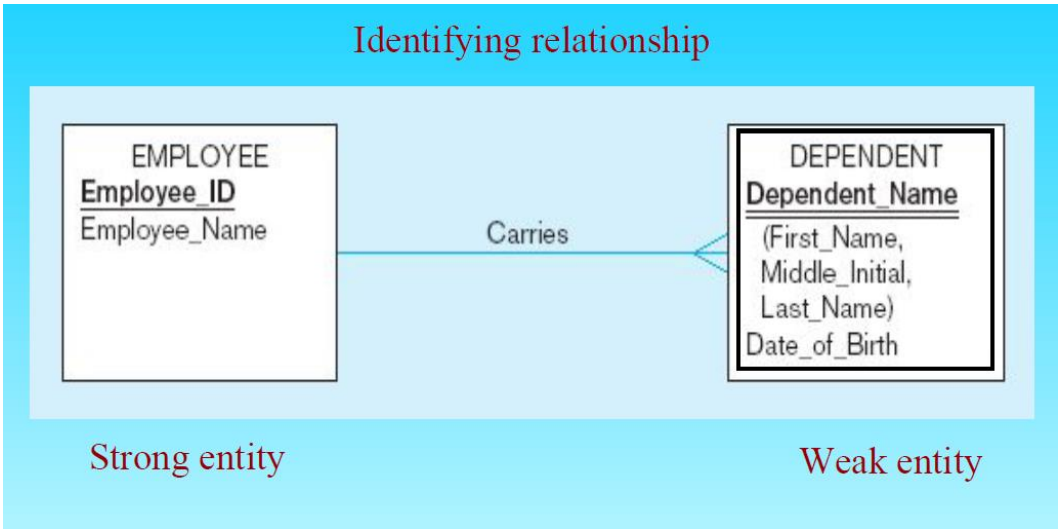


Figure 3-11b An associative entity (CERTIFICATE)

Associative entity is like a relationship with an attribute, but it is also considered to be an entity in its own right.

Chapter-04

Supertypes and Subtypes

■ Subtype: A subgrouping of the entities in an entity type that has attributes distinct from those in other subgroupings

■ Supertype: A generic entity type that has a relationship with one or more subtypes

Figure 4-1 Basic notation for supertype/subtype notation

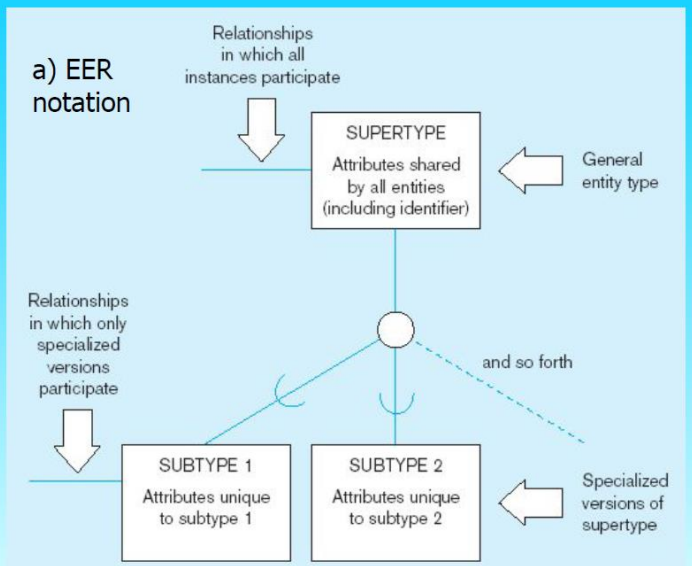
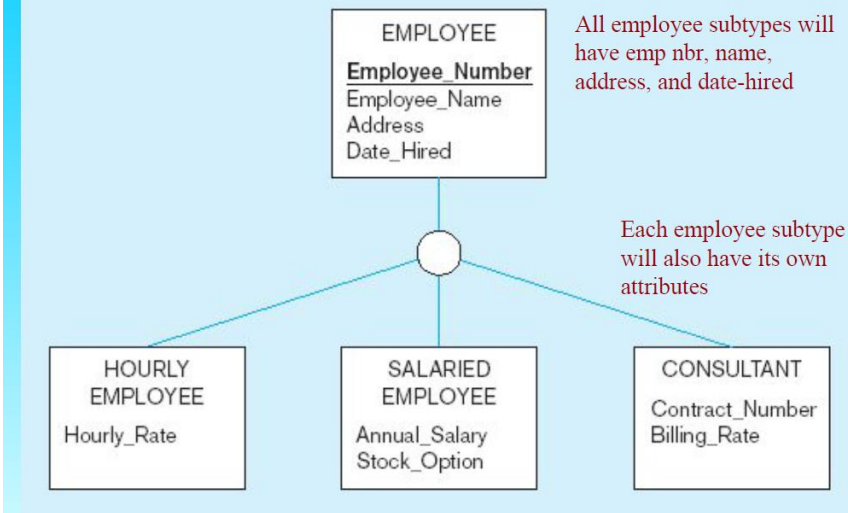
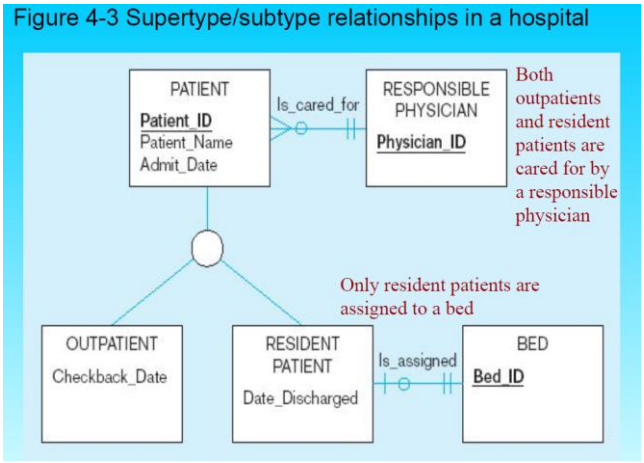


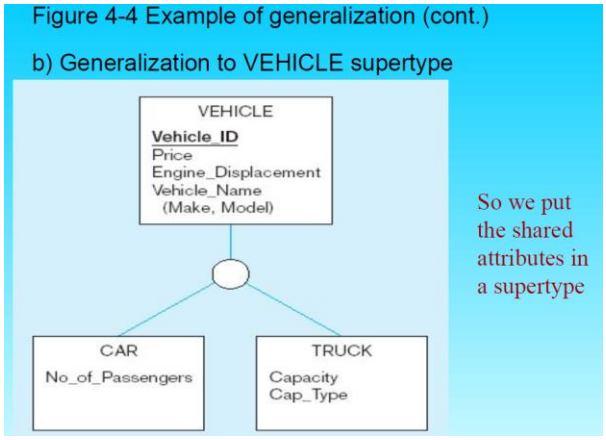
Figure 4-2 Employee supertype with three subtypes



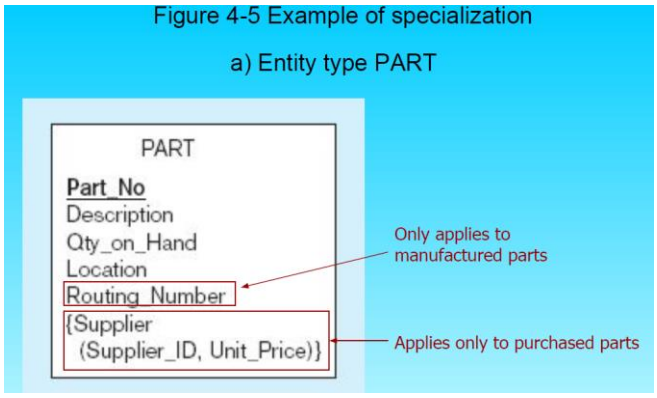


Generalization and Specialization

■ Generalization: The process of defining a more general entity type from a set of more specialized entity types. BOTTOM-UP



■ Specialization: The process of defining one or more subtypes of the supertype and forming supertype/subtype relationships. TOP-DOWN



Entity Clusters

❖ Set of one or more entity types and associated relationships grouped into a single abstract entity type

CHAPTER-05

(Logical Database Design and the Relational Model)

Relation

Definition: A relation is a named, two-dimensional table of data

■ Table consists of rows (records) and columns (attribute or field)

Requirements for a table to qualify as a relation:

■ It must have a unique name

■ Every attribute value must be atomic (not multivalued, not composite)

■ Every row must be unique (can't have two rows with exactly the same values for all their fields)

■ Attributes (columns) in tables must have unique names

NOTE: all relations are in 1st Normal form

Key Fields

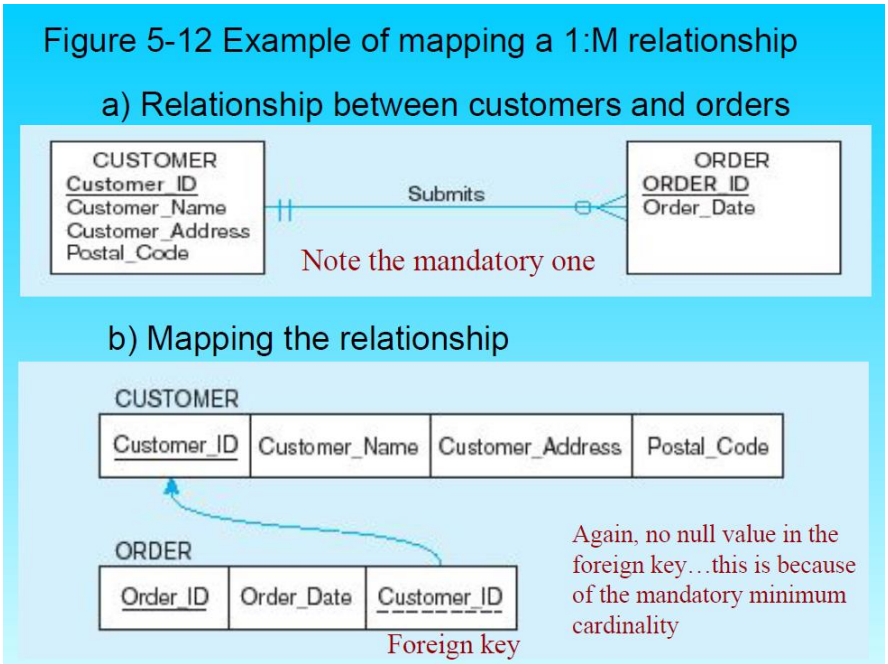
Keys are special fields that serve two main purposes:

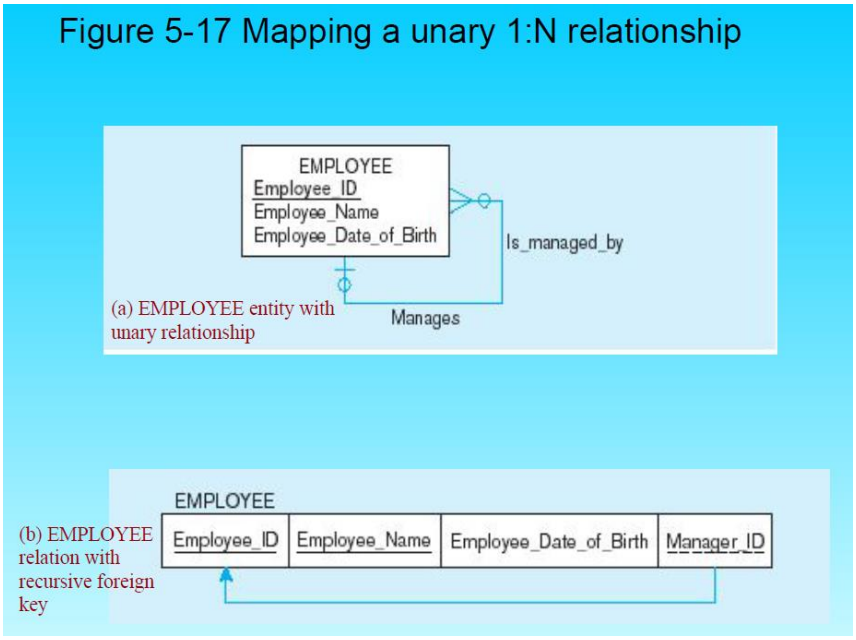
■ **Primary keys** are unique identifiers of the relation in question.

Examples: include employee numbers, social security numbers, etc.

■ **Foreign keys** are identifiers that enable a dependent relation (on the many sides of a relationship) to refer to its parent relation (on the one side of the relationship)

Transforming EER Diagrams into Relations (cont.)





Data Normalization

The process of decomposing relations with anomalies to produce smaller, well-structured relations expression.

Well-Structured Relations

■ A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies

❖ Goal is to avoid anomalies

■ Insertion Anomaly–adding new rows forces user to create duplicate data

■ Deletion Anomaly–deleting rows may cause a loss of data that would be needed for other future rows

■ Modification Anomaly–changing data in a row force changes to other rows because of duplication

Example–Figure 5-2b

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Question–Is this a relation?

Answer–Yes: Unique rows and no multivalued attributes

Question–What's the primary key?

Answer–Composite: Emp_ID, Course_Title

Anomalies in this Table

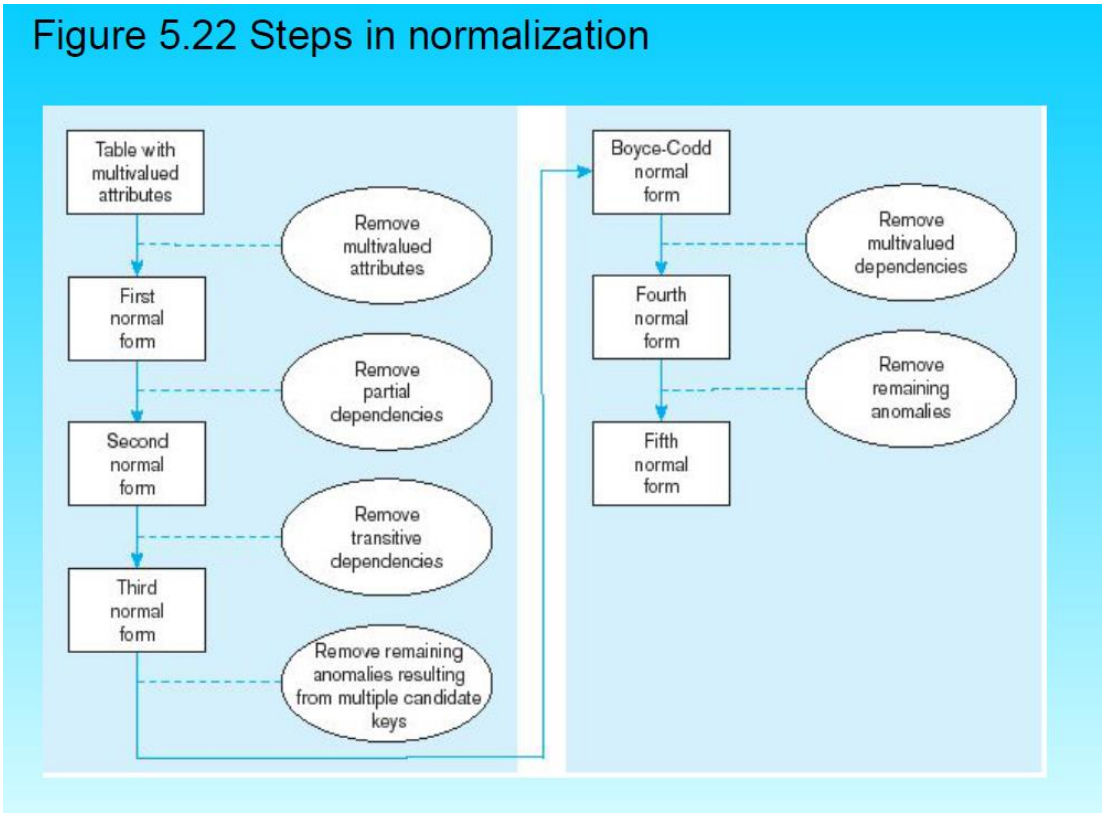
- **Insertion**–can't enter a new employee without having the employee take a class
- **Deletion**–if we remove employee 140, we lose information about the existence of a Tax Acc class
- **Modification**–giving a salary increase to employee 100 forces us to update multiple records

Why do these anomalies exist?

Because there are two themes (entity types) in this one relation. This results in data duplication and an unnecessary dependency between the entities

Candidate Key:

■ A unique identifier. One of the candidate keys will become the primary key



First Normal Form

■ No multivalued attributes

■ Every attribute value is atomic

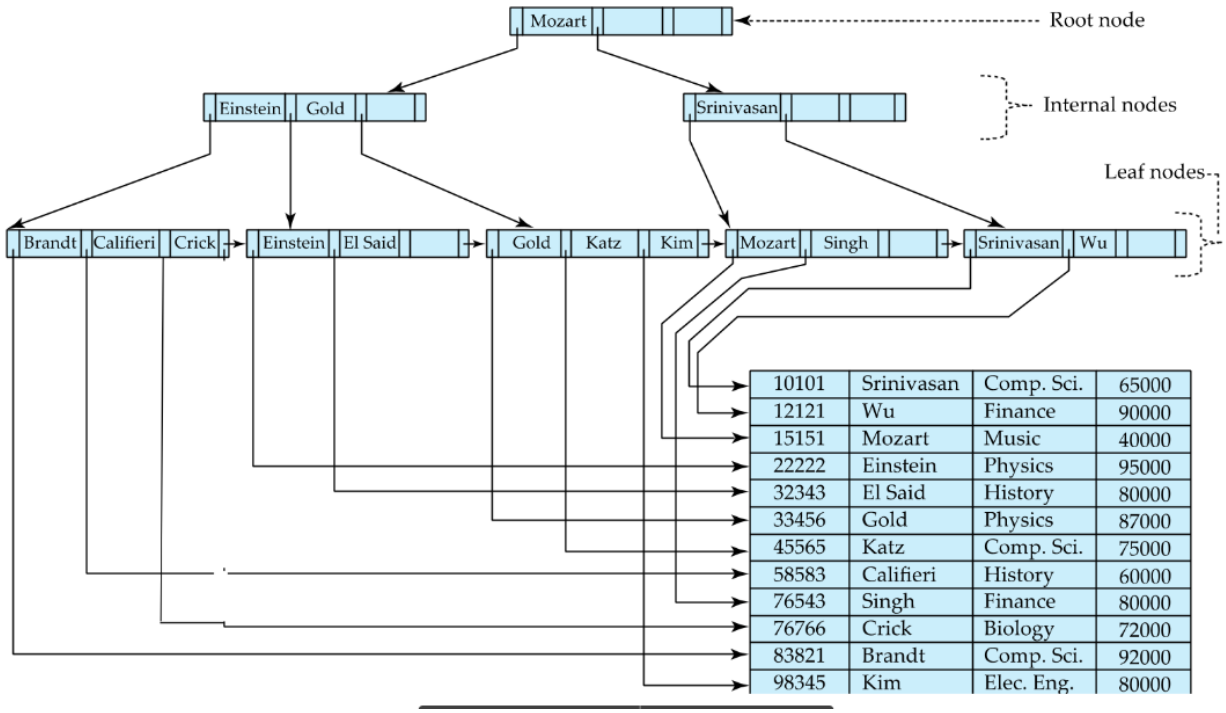
Chapter 14 (Indexing)

Dense index — Index record appears for every search-key value in the file

Sparse Index: contains index records for only some search-key values.

- Applicable when records are sequentially ordered on search-key

Example of B$^+$-Tree



B$^+$-Tree Index Files (Cont.)

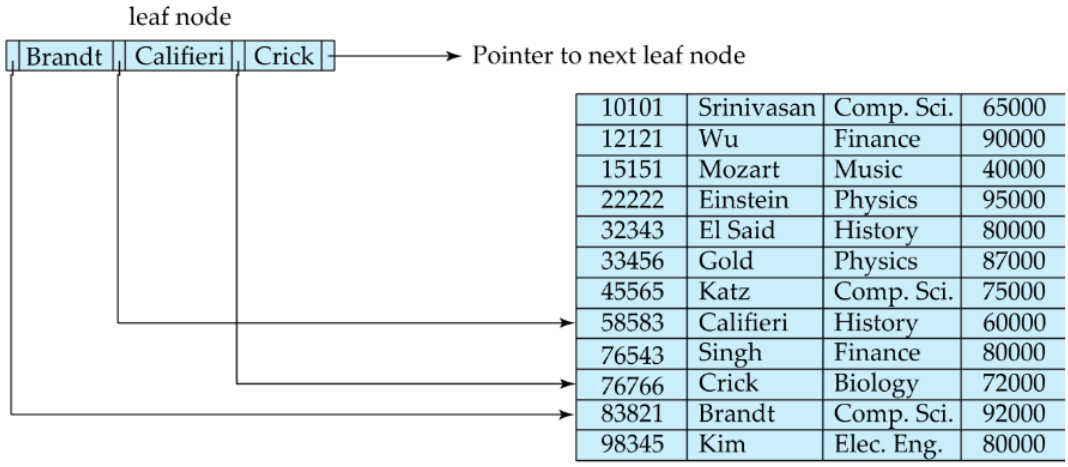
A B$^+$-tree is a rooted tree satisfying the following properties:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
- A leaf node has between $\lceil (n–1)/2 \rceil$ and $n–1$ values
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n–1)$ values.

Leaf Nodes in B$^+$-Trees

Properties of a leaf node:

- For $i = 1, 2, \ldots, n-1$, pointer P_i points to a file record with search-key value K_i,
- If L_i, L_j are leaf nodes and $i < j$, L_i's search-key values are less than or equal to L_j's search-key values
- P_n points to next leaf node in search-key order



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

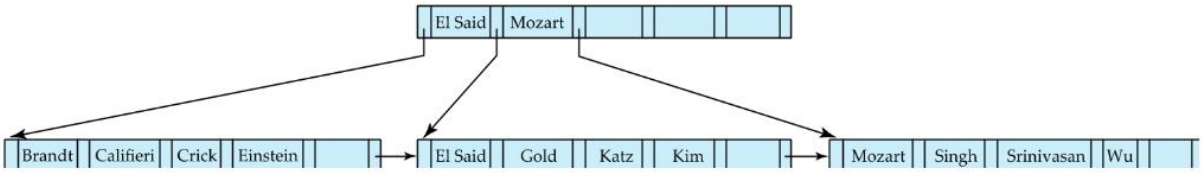
Non-Leaf Nodes in B$^+$-Trees

- Non leaf nodes form a multi-level sparse index on the leaf nodes. For a non-leaf node with m pointers:
  - All the search-keys in the subtree to which P_1 points are less than K_1
  - For $2 \le i \le n-1$, all the search-keys in the subtree to which P_i points have values greater than or equal to K_{i-1} and less than K_i
  - All the search-keys in the subtree to which P_n points have values greater than or equal to K_{n-1}
  - General structure

P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

Example of B$^+$-tree

- B$^+$-tree for *instructor* file (n = 6)



- Leaf nodes must have between 3 and 5 values ($\lceil (n–1)/2 \rceil$ and n –1, with n = 6).
- Non-leaf nodes other than root must have between 3 and 6 children ($\lceil (n/2 \rceil$ and n with n =6).
- Root must have at least 2 children.

B$^+$-Tree Node Structure

- Typical node

P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

  - K_i are the search-key values
  - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

  $K_1 < K_2 < K_3 < \ldots < K_{n-1}$

  (Initially assume no duplicate keys, address duplicates later)

Chapter 18: Concurrency Control

Deadlock (Cont.)

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- **Starvation** is also possible if concurrency control manager is badly designed. For

Deadlock Handling

- System is **deadlocked** if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Deadlock Handling

- ***Deadlock prevention*** protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies:
  - Require that each transaction locks all its data items before it begins execution (pre-declaration).
  - Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

More Deadlock Prevention Strategies

- **wait-die** scheme — non-preemptive
  - Older transaction may wait for younger one to release data item.
  - Younger transactions never wait for older ones; they are rolled back instead.
  - A transaction may die several times before acquiring a lock
- **wound-wait** scheme — preemptive
  - Older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it.
  - Younger transactions may wait for older ones.
  - Fewer rollbacks than *wait-die* scheme.
- In both schemes, a rolled back transactions is restarted with its original timestamp.