Unit I

Introduction to DBMS

1. Database Systems
2. Data modeling
3. Relational Model

Database Systems

Objectives:

- What is a database and a database system?
- Explain the characteristics of a database system
- Data Abstraction
- Database languages
- Explain the concept of data independence and its importance
- Explain the architecture of a database system

- Database is a collection of coherent, meaningful , interrelated data.
- A database management system is a system that allows inserting, updating, deleting & processing of a data.

Database system Applications

- Banking
- Airlines
- Universities
- Credit cards transactions
- Telecommunications
- Finance
- Sales
- Human resources

Purpose of database systems

- In 1960's, one way to keep the information on a computer is to store it in operating system files.
- The system stores permanent record in various files and it needs different application programs to extract records from and add records to the appropriate files

Disadvantages of a file processing systems

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

Characteristics of Database Systems (Benefits of DBMS)

- Performance (No data inconsistencies)
- Minimal Redundancy
- Data Integrity
- Privacy and Security
- Migration of Data
- Data Independence

Data abstraction

- A major purpose of database system is to provide users with an abstract view of the data
- The system hides certain details of how the data are stored and maintained
- Since many database users are not computer trained, developers hide the complexity from users through several levels of abstraction

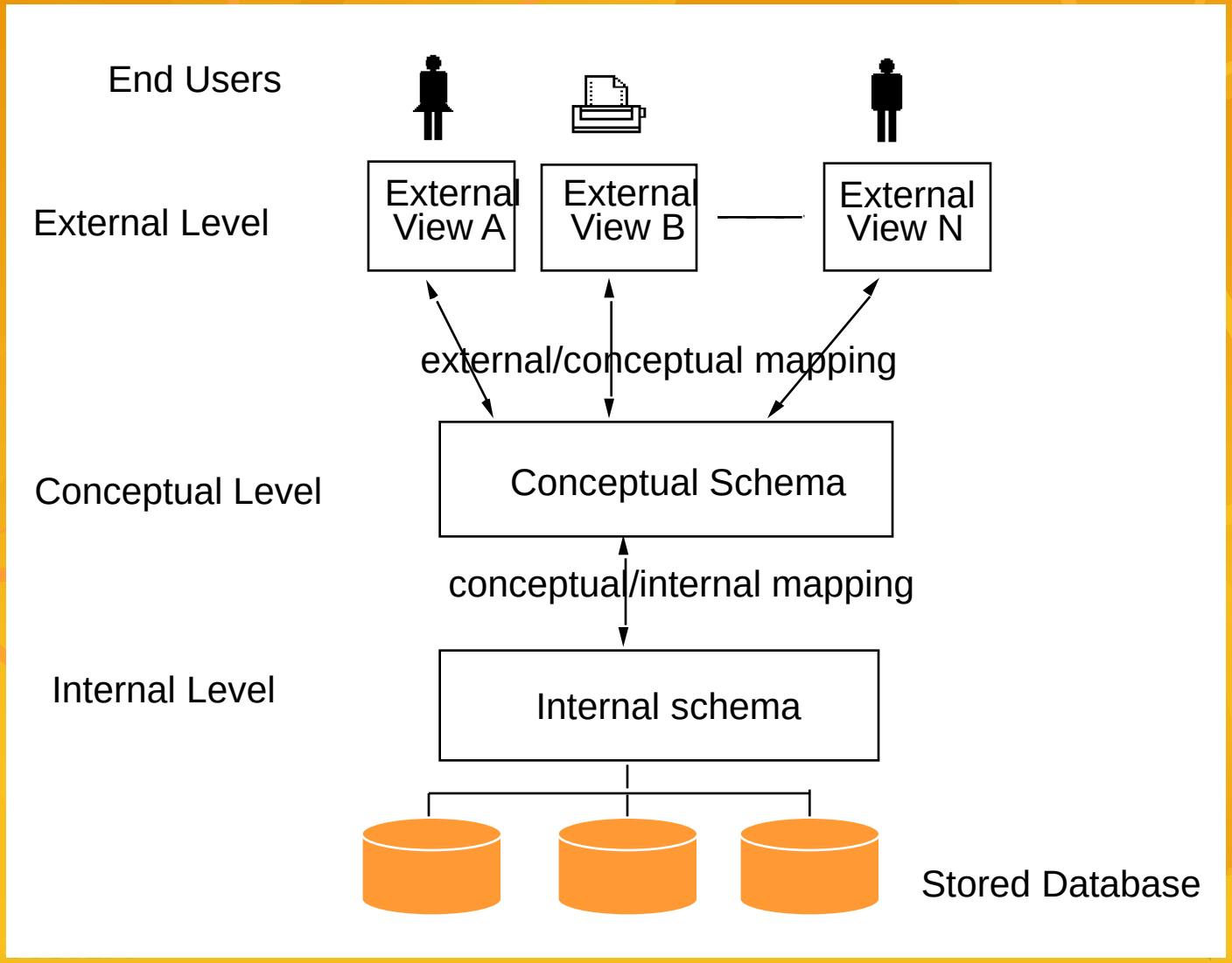
Data abstraction

- Physical level: describes how the data are actually stored (internal schema)
- Logical level: describes what data are stored in the database and what relationship exist among those data (conceptual schema)
- View level: describes only part of the entire database (external schema)

Schemas and instances

- Databases change over time as information is inserted and deleted
- The collection of information stored in the database at a particular moment is called an instance of the database
- The overall design of the database is called the database schema
- Database schema is not expected to change frequently
- A displayed schema is called schema diagram

The three schema architecture



Data independence

- Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level
- Two types of data independence:
- Logical data independence
- Physical data independence

Data independence

- Logical data independence
- It is the capacity to change the conceptual schema without having to change external schemas or application programs
- Physical data independence
- It is the capacity to change the internal schema without having to change the conceptual schema. Hence the external schemas need not be changed as well

Data models

- Data model is a set of data structures and conceptual tools used to describe the structure of a database

  (data types, relationships, and constraints)
- A data model provides a way to describe the design of a database at the physical, logical and view level
- Most data models also include a set of basic operations for specifying retrievals and updates on the database

Data models

- Different data models are
  - Relational model:
  - The Entity- Relationship model:
  - Object based data model:
  - Semistructured data model:

Data models

- Relational model:
- It uses a collection of tables to represent both data and the relationship among those data.
- Each table has multiple columns and each column has a unique name
- Each table contains records of a particular type
- Most widely used data model

Data models

- The Entity- Relationship model:
- The E-R model is a collection of basic objects called entities and of a relationship among these objects
- An entity is a real world thing or object
- It is widely used in database design

Data models

- Object oriented data model:
- Extending the E-R model with concept of encapsulation, methods(functions) and object identity
- The object relational data model combines features of the object oriented data model and relational data model

Data models

- Semistructured data model:
- It permits the specification of data where individual data items of the same type may have different set of attributes
- **The extensible Markup language (XML) is widely used to represent semistructured data**

Database languages

- Data definition language (DDL)
- Specify database schema
- Also used to specify additional properties of the data
- The data values stored in the database must satisfy certain constraints
- The DDL provides facility to specify such constraints

Database languages

- Data manipulation language (DML)
- Enables user to access or manipulate data
  - Insertion of new information into the database
  - Deletion of information from the database
  - Modification of information stored in the database
- Data query language (DQL)
  - Retrieval of information stored in the database

Components and over all structure of a DBMS

- The functional components of a database system can be divided into
- Disk storage:
- Databases require a large amount of storage space. Since main memory of computers can not store this much information, the information is stored on disk. Data are moved between disk storage and main memory as needed.

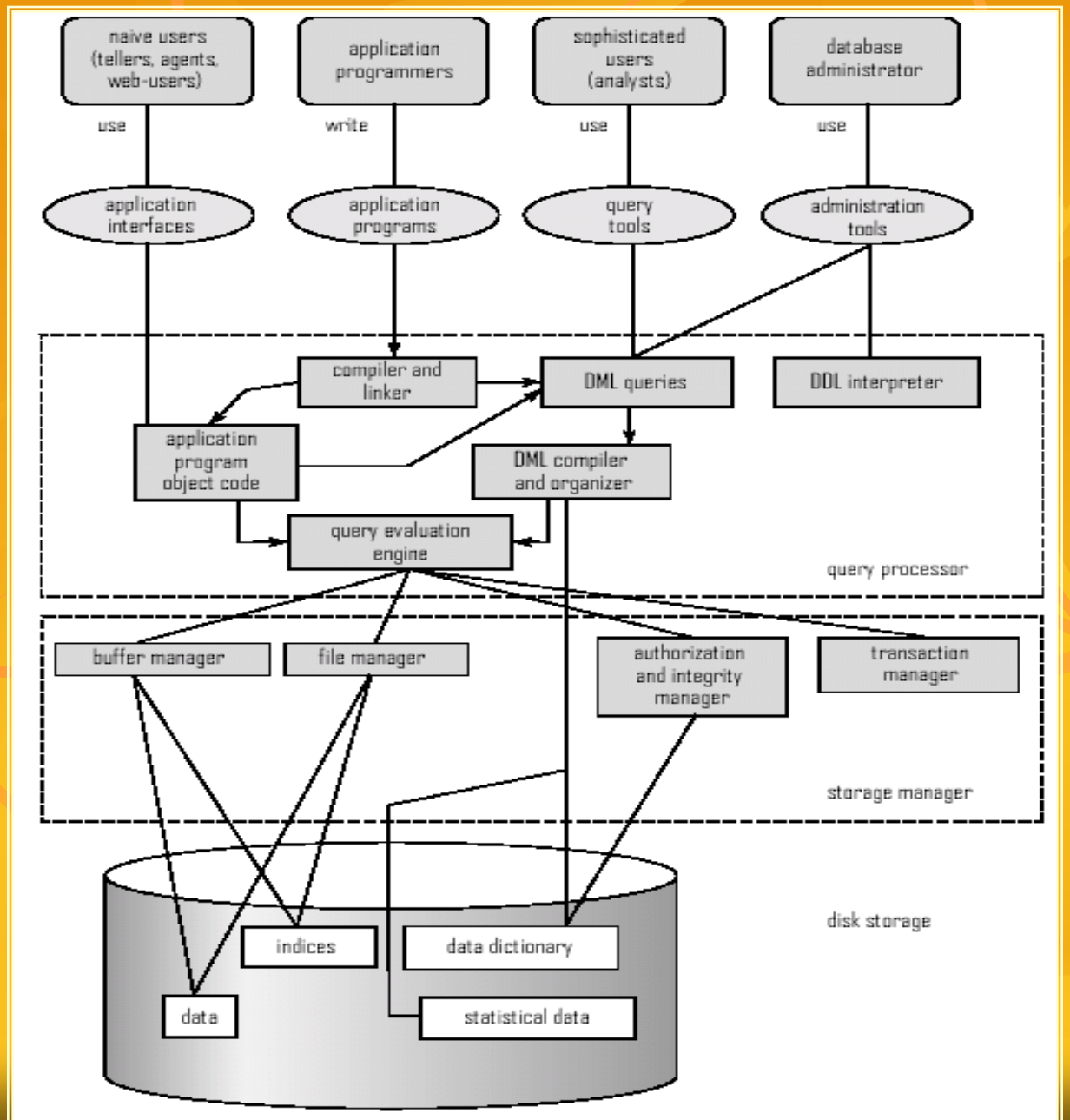
- Disc storage has several data structure
- Data files: which store the database itself
- Data dictionary: which stores metadata about the structure (schema)of the database
- Indices: which can provide fast access to data items. Hashing is an alternative to indexing that is faster in some but not all cases

- The storage manager:
- It is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system
- It is responsible for storing, retrieving and updating data in the database

- The storage manager components include:
- Authorization and integrity manager: tests for satisfaction of integrity constraints and checks the authority of users to access data
- Transaction manager: ensures that the database remains in a consistent state despite system failures
- File manager: manages the allocation of space on disk storage
- Buffer manager: responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory

- The query processor components:
- The query processor components include
- DDL interpreter: interprets DDL statements and records the definitions in the data dictionary
- DML compiler: translates DML statements in query language into an evaluation plan that the query evaluation engine understands.
- Query evaluation engine: executes low level instructions generated by the DML compiler

Database Architecture



Database Architecture

- The architecture of a database system is greatly influenced by the computer system on which the database system runs
- Database can be centralized or client-server
- In client server architecture, one server machine executes work on be half of multiple client machines
- Database applications are usually partitioned into two tier architecture and three tier architecture

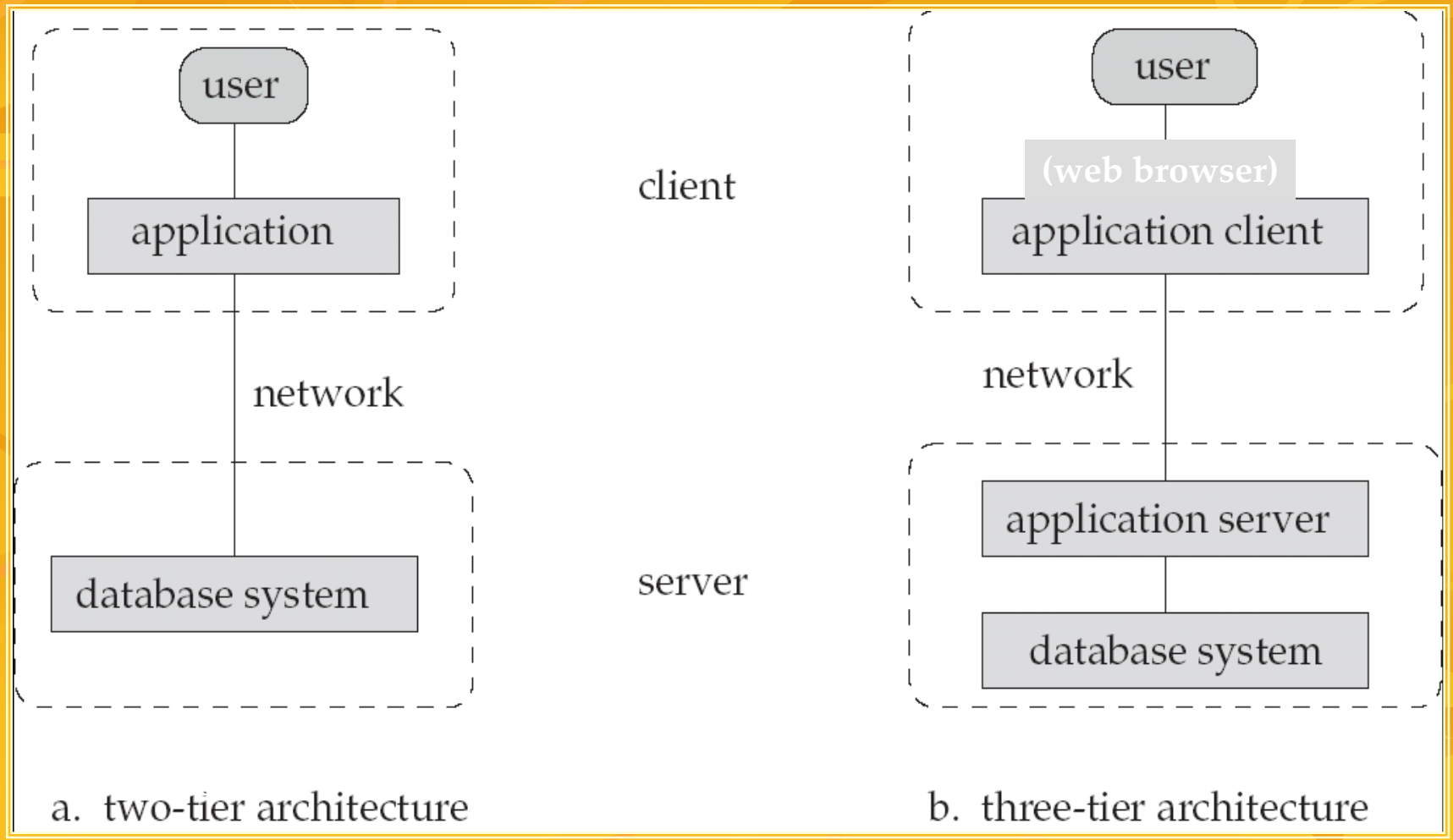
Database Architecture

- In a two tier architecture, the application is partitioned into a component that resides at the client machine which invokes database system functionality at the server machine through query language statements
- Application program interface standards like ODBC and JDBC are used for interaction between the client and the server

Database Architecture

- In three tier architecture, the client machine acts as a front end and does not contain any direct database call
- The client end communicates with an application server.
- The application server in turn communicates with a database system to access data.

Database Application Architectures



Database users and database administrator

- People who work with a database can be categorized as database users or database administrators
- Different types of database system users are
- Naïve users: ATM card holder
- Application programmers: computer professional who write application programs
- Sophisticated users: form their requests in a database query language
- Specialized users: who write specialized database applications e.g. expert system, CAD systems

Database administrator

- DBMS have central control of both the data and the programs that access those data
- A person who has such central control over the system is called a database administrator (DBA)
- The function of DBA include:
- Schema definition
- Granting of authorization for data access
- Routine maintenance

Data modeling

Objective:

- What is data modeling?
- Basic concepts in E-R model like entity, attributes , relationship, constraints & identifiers (keys)
- E-R diagrams
- EER diagrams

Data modeling

- A data model is a plan for building a database.
- Data modeling is the process of creating a data model by applying formal data model descriptions using data modeling techniques.
- It is sometimes called *database modeling* because a data model is eventually implemented in a database.
- The Entity-Relation Model (ER) is the most common method used to build data models

Entity-Relationship (E-R) Modeling

Key Terms

- ***Entity:***
- An entity can be a person, place, object, event or concept in the user environment about which the organization wishes to maintain data
- Real-world object distinguishable from other objects
- *e.g a student, car, job, subject, building ...*

EMPLOYEE

Entity Sets

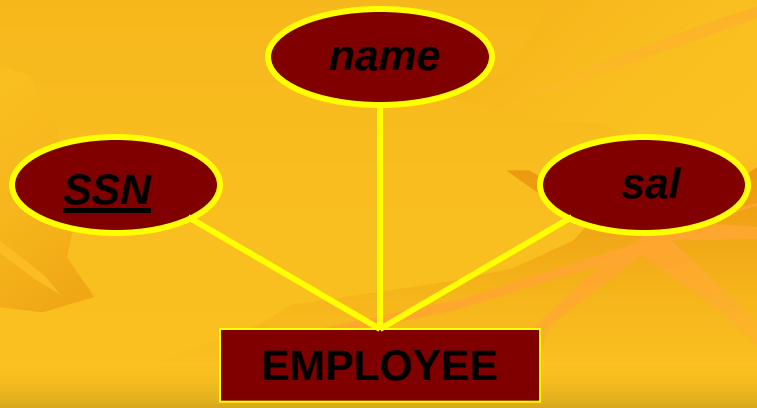
A collection of similar entities (e.g. all employees)

- All entities in an entity set have the same set of attributes
- Each entity set has a *key*
- Each attribute has a *domain*

EMPLOYEES

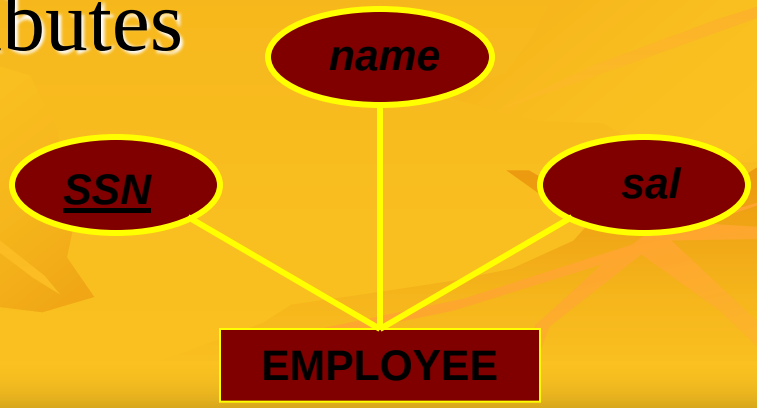
SSN	NAME	SAL
321-23-3241	Kim	23,000
645-56-7895	Jones	45,000

- ***Entity Type:***
- A collection of entities that share common properties or characteristics
- Each Entity Type is described by its **NAME** and attributes
- The Entity Type describes the “Schema” or “Intension” for a set of entities
- Collection of all entities of a particular entity type at a given point in time is called the “Entity Set” or “Extension” of an Entity Type



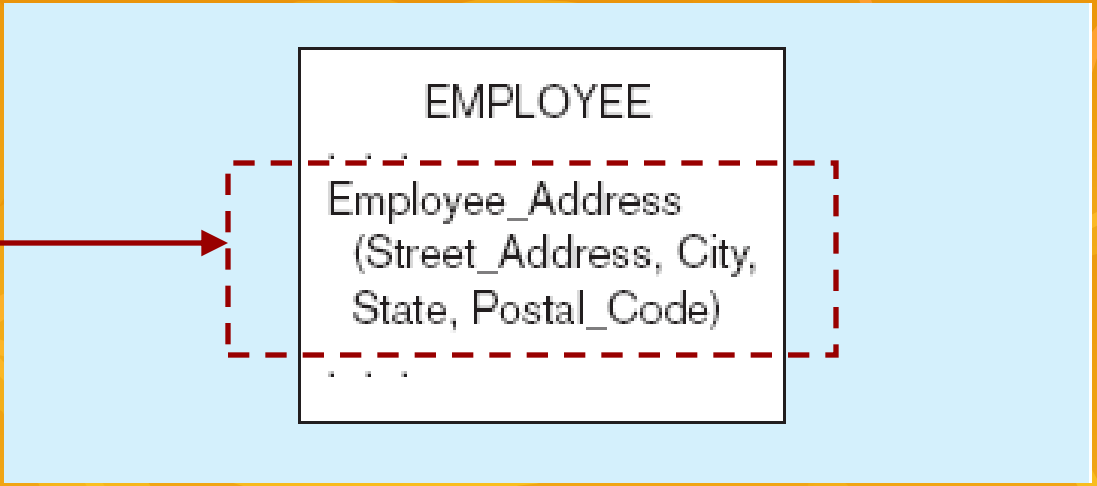
Attributes

- A named property or characteristic of an entity that is of interest to an organization
- Attribute Types
  - Composite Vs. Simple Attributes
  - Single-valued Vs. Multi-valued Attributes
  - Derived attributes
  - Key attributes

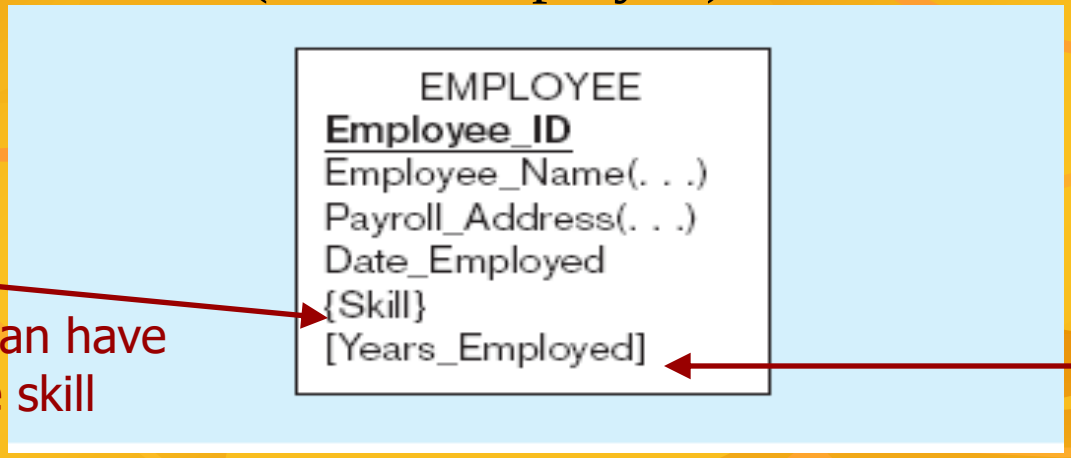


A **composite** attribute

An attribute broken into component parts



Entity with **multivalued** attribute (Skill) and **derived** attribute (Years_Employed)



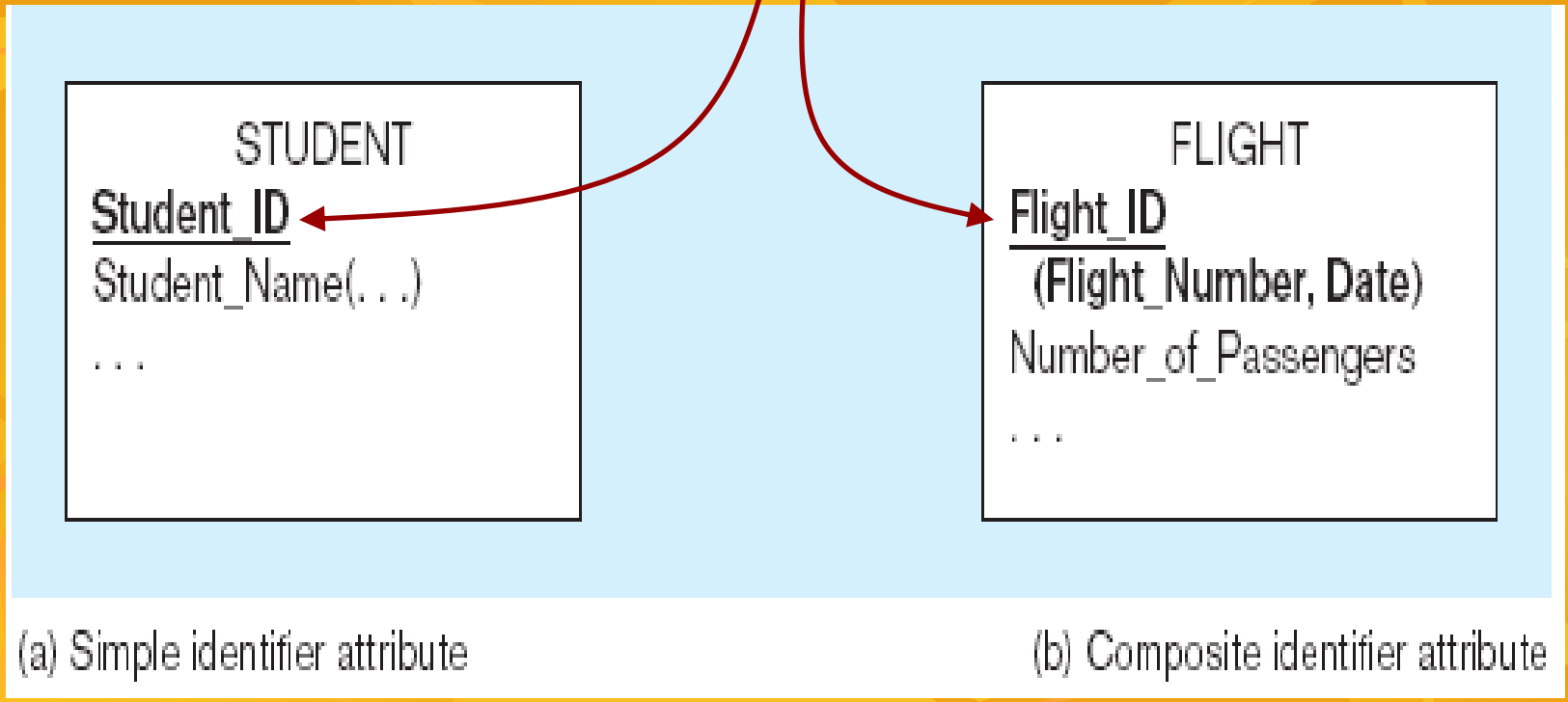
Multivalued
an employee can have more than one skill

Derived
from date employed and current date

40

Simple and composite identifier attributes

The identifier is boldfaced and underlined



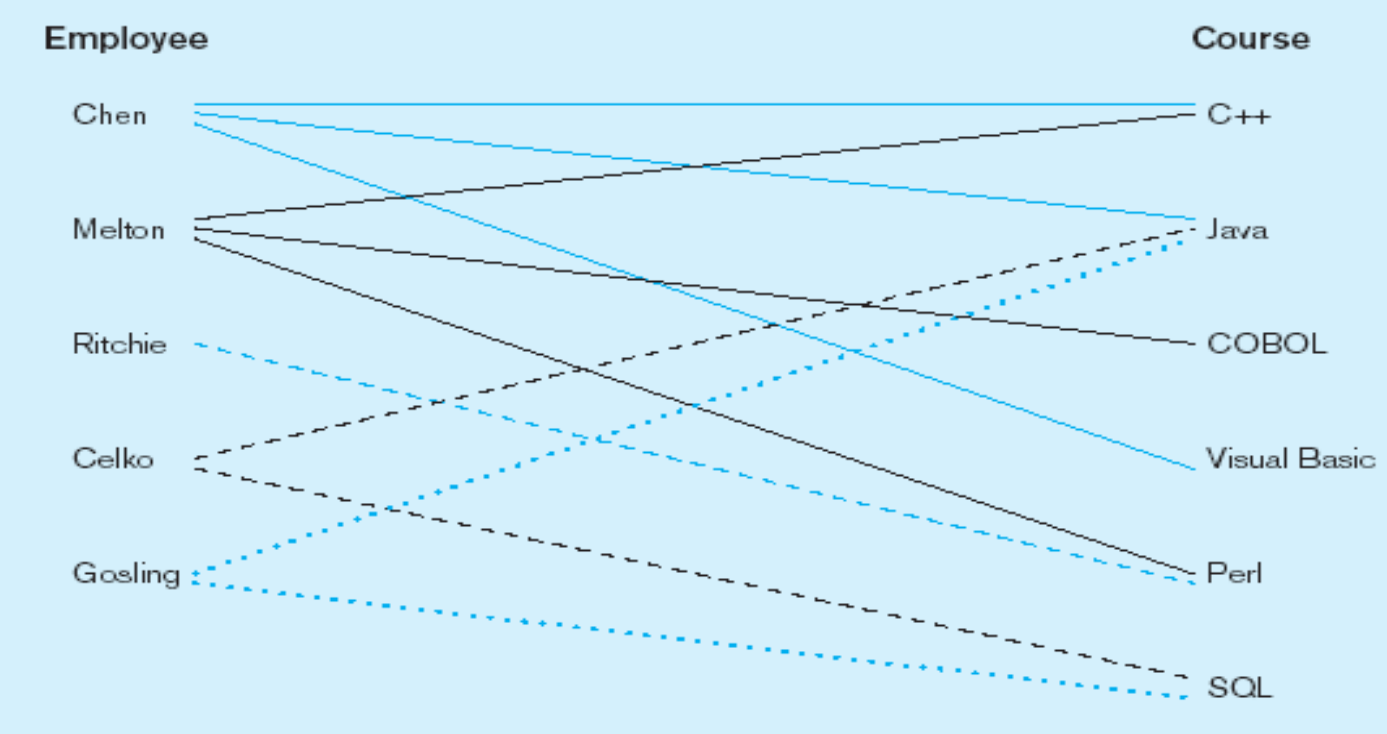
Relationship:

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

Relationship types and instances

a) Relationship type

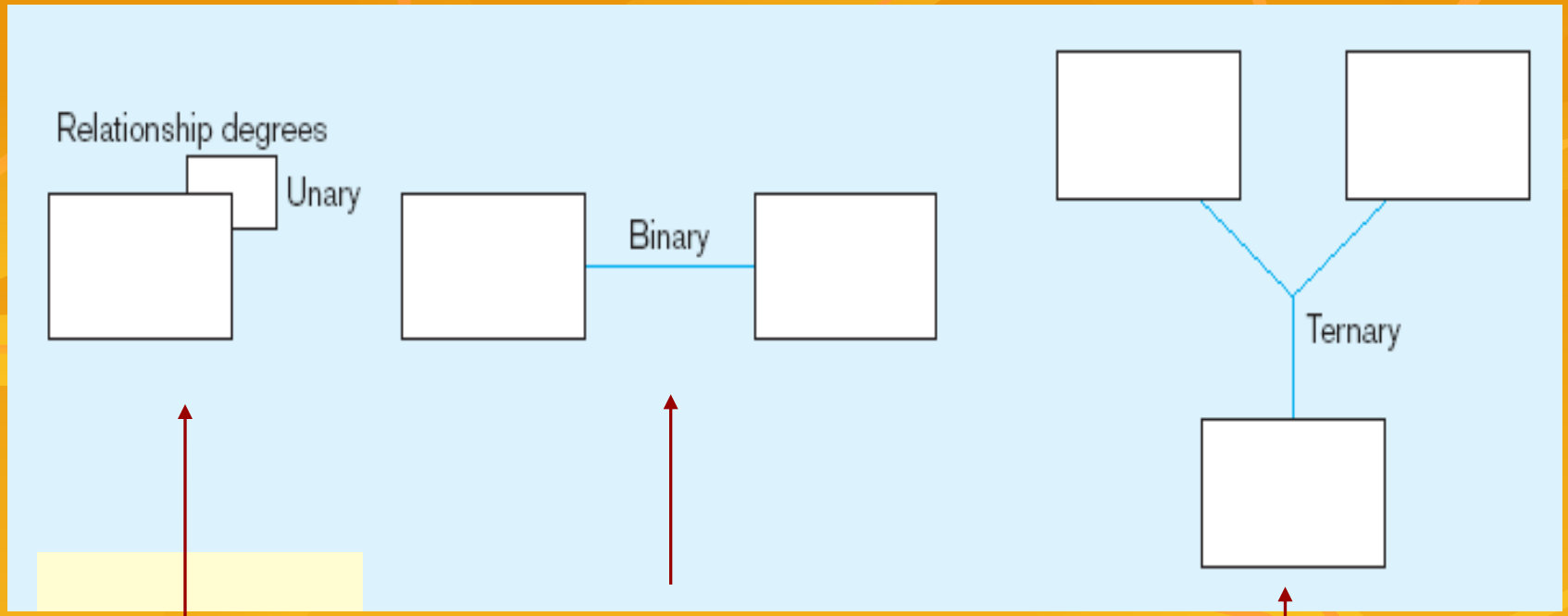
b) Relationship instances



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



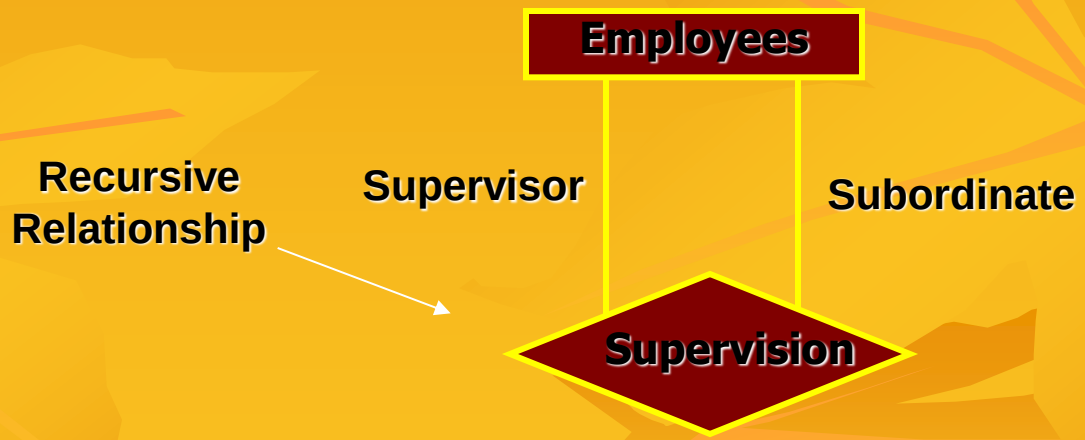
One entity related to another of the same entity type

Entities of two different types related to each other

Entities of three different types related to each other

Recursive Relationships

- Same entity type can participate more than once in the same relationship type under different "roles"
- Such relationships are called "**Recursive Relationships**"

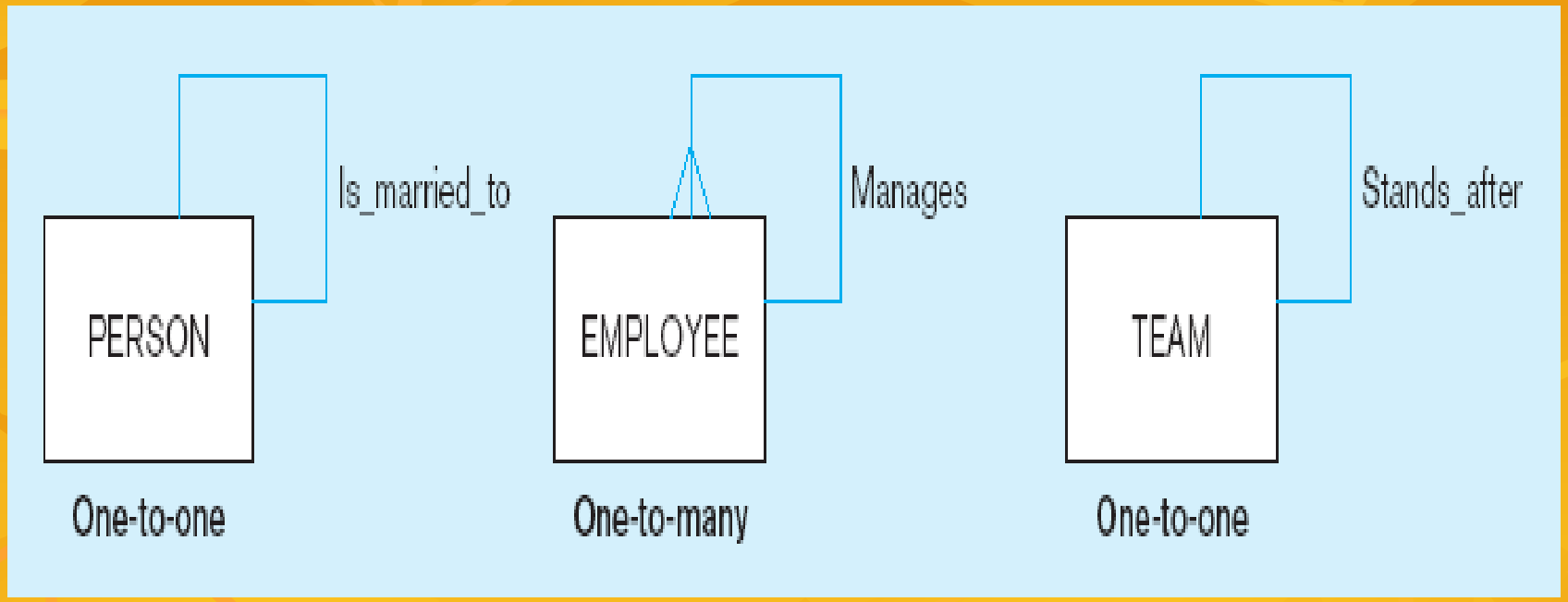


Cardinality of Relationships

- One-to-One
  - Each entity in the relationship will have exactly one related entity
- One-to-Many
  - An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
- Many-to-Many
  - Entities on both sides of the relationship can have many related entities on the other side

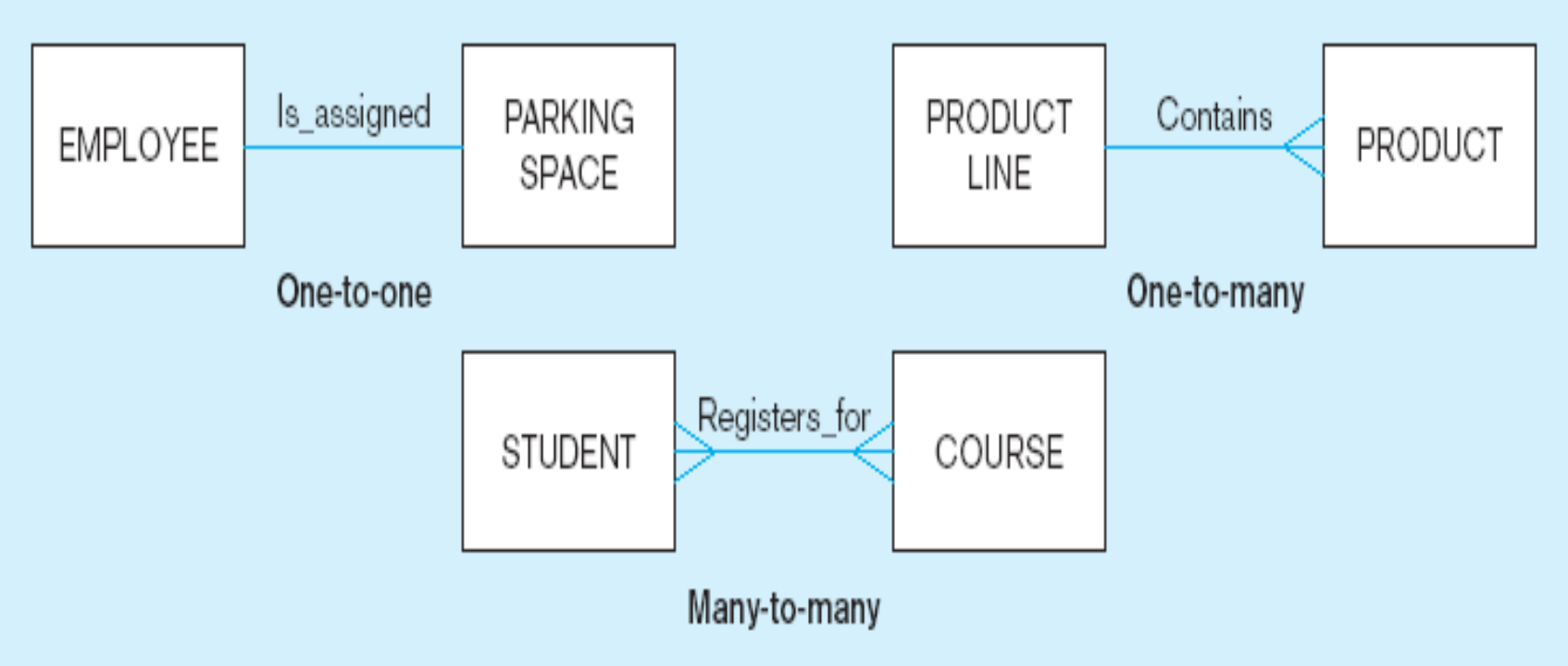
Examples of relationships of different degrees

a) Unary relationships



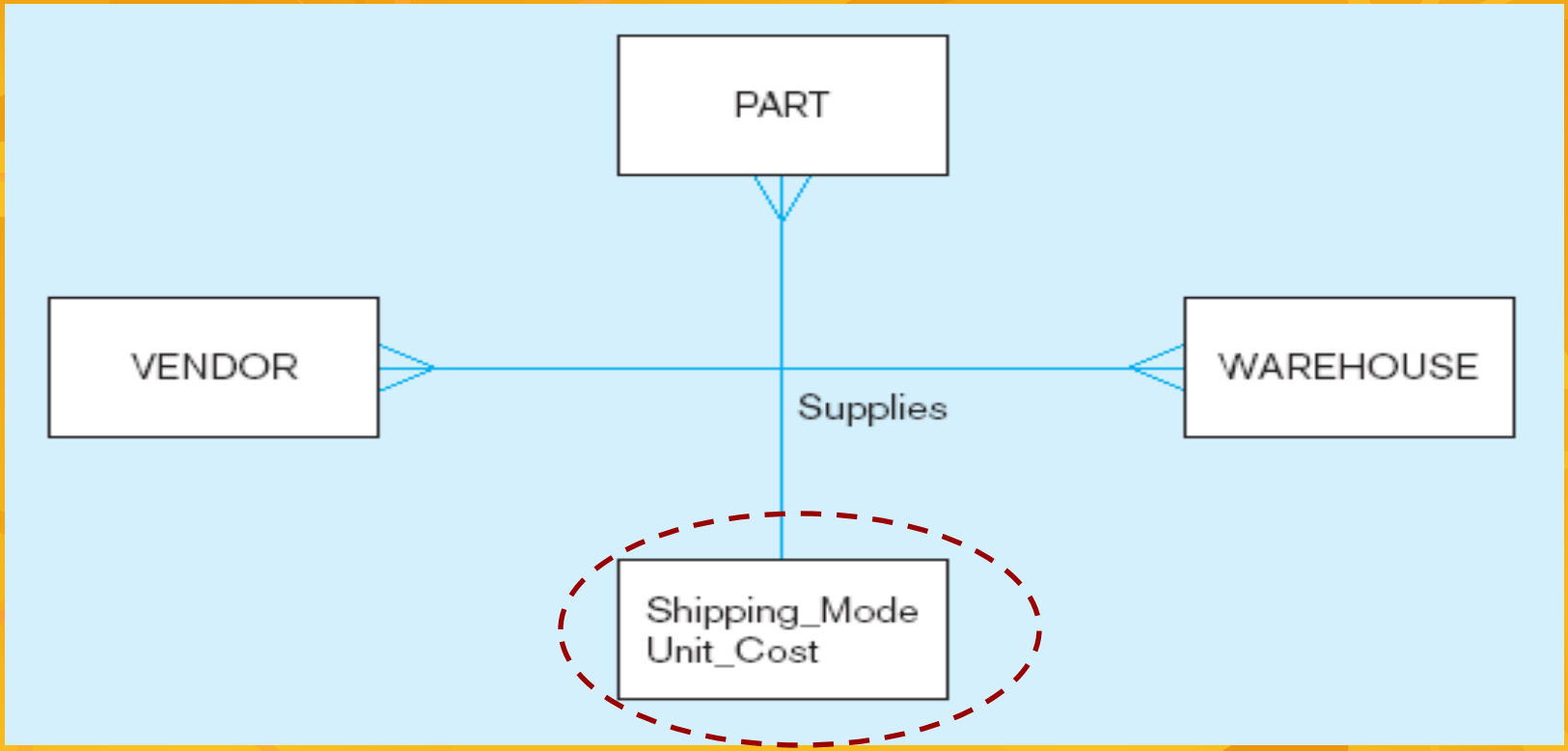
Examples of relationships of different degrees (cont.)

b) Binary relationships



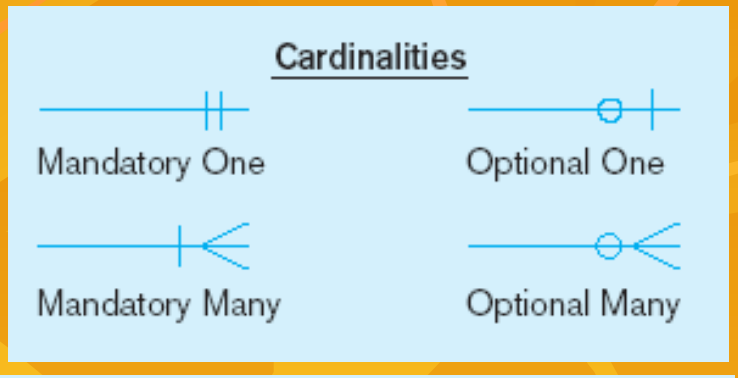
Examples of relationships of different degrees (cont.)

c) Ternary relationship

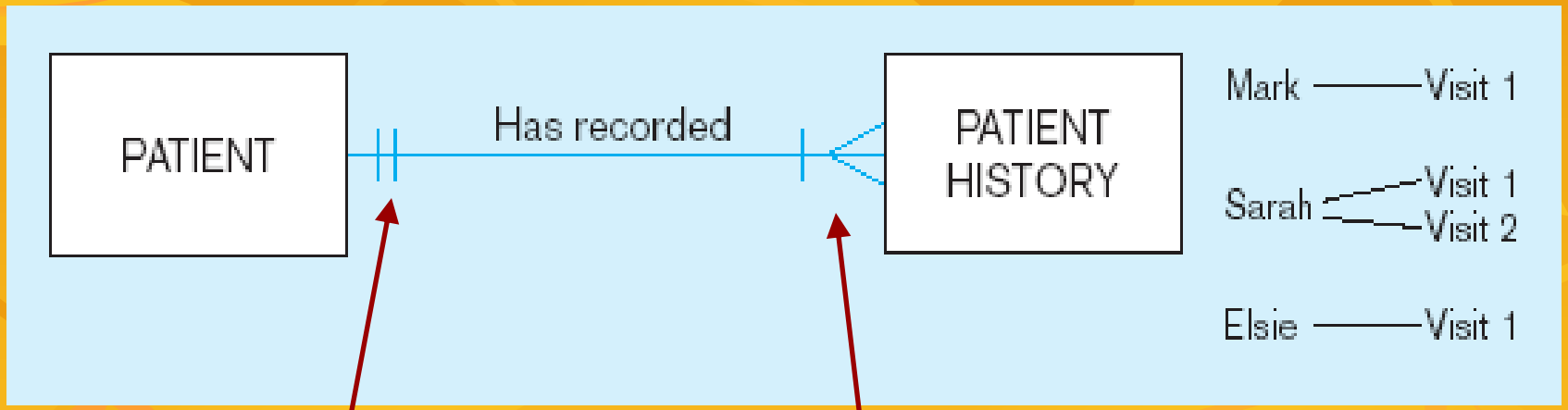


Note: a relationship can have attributes of its own

Examples of cardinality constraints



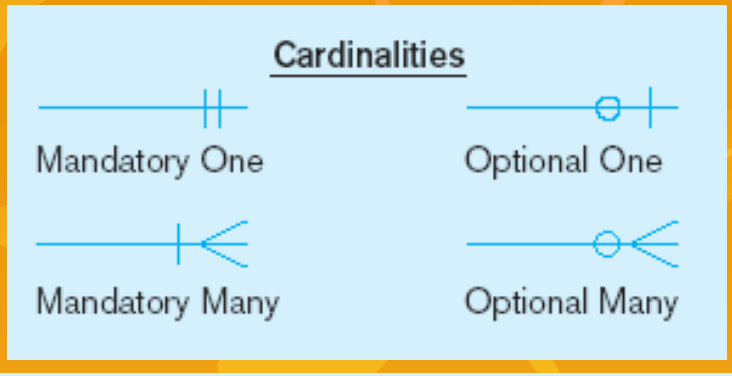
a) Mandatory cardinalities



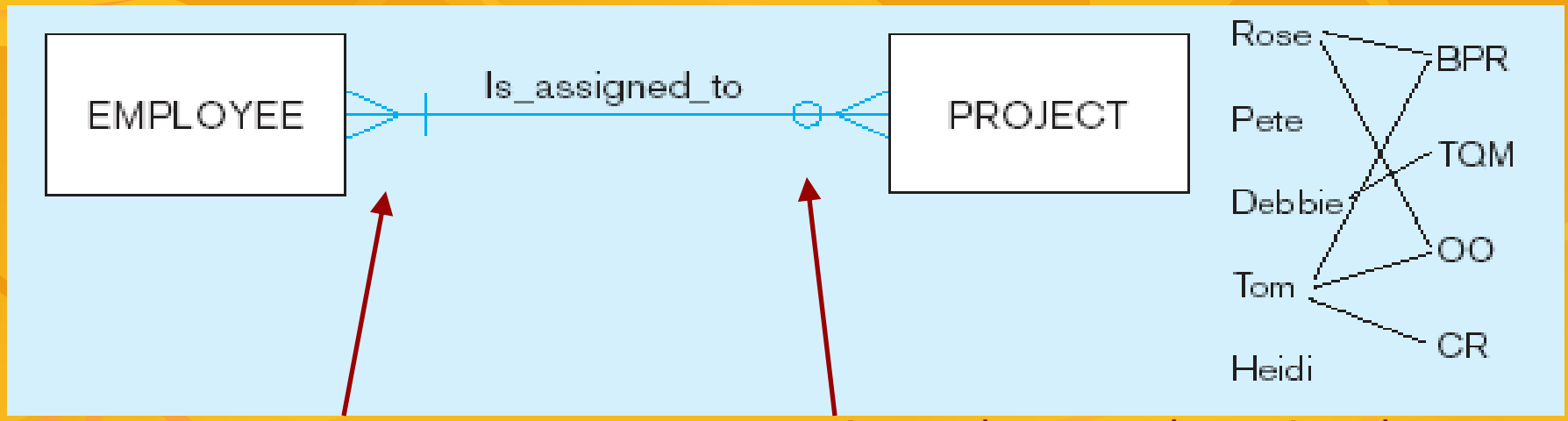
A patient history is recorded for one and only one patient

A patient must have recorded at least one history, and can have many

Examples of cardinality constraints (cont.)



b) One optional, one mandatory

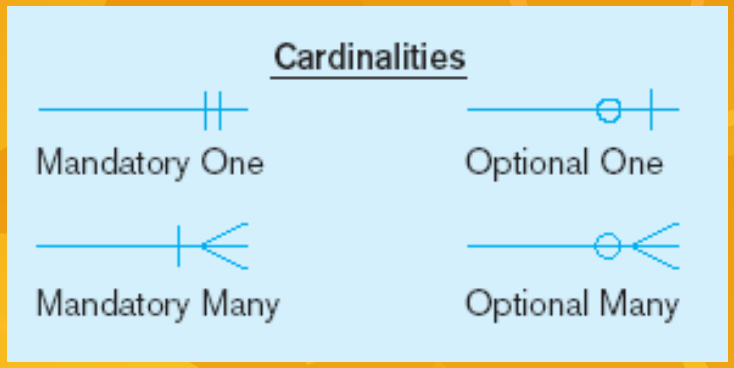


A project must be assigned to at least one employee, and may be assigned to many

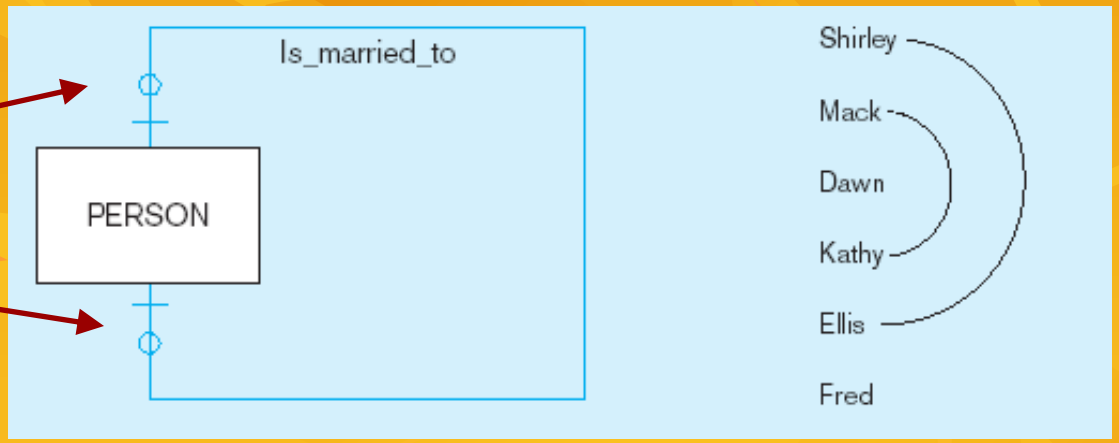
An employee can be assigned to any number of projects, or may not be assigned to any at all

Examples of cardinality constraints (cont.)

c) Optional cardinalities



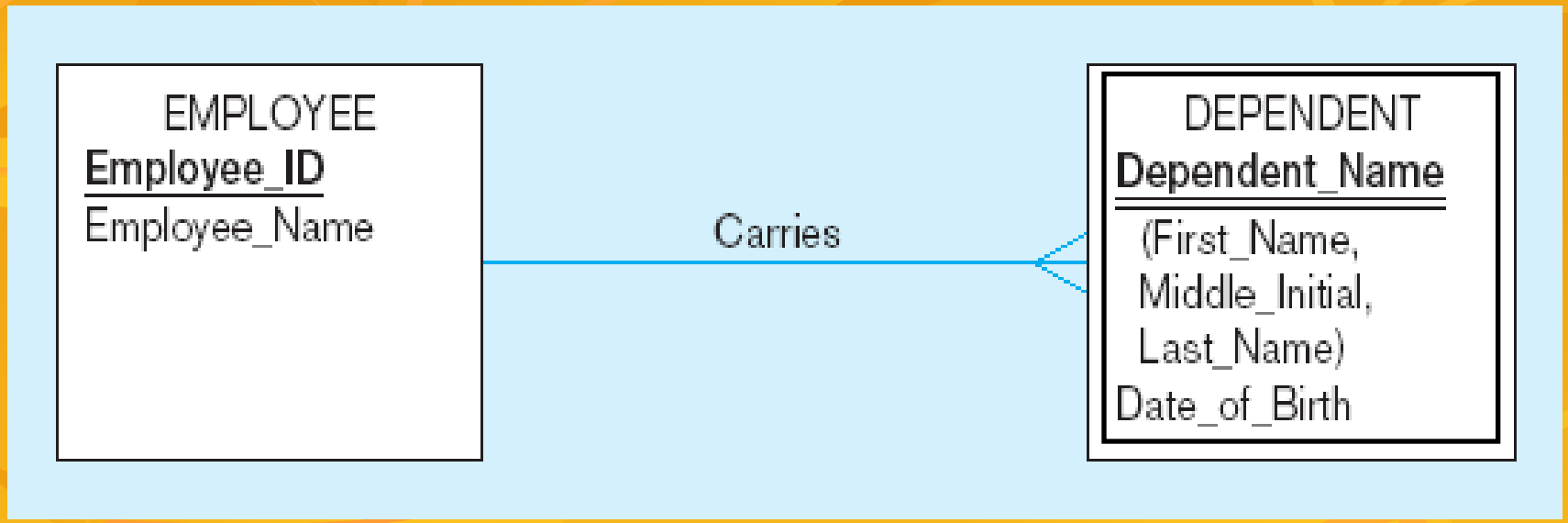
A person is is married to at most one other person, or may not be married at all



Strong vs. Weak Entities, and Identifying Relationships

- 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
- Identifying relationship
  - links strong entities to weak entities

Identifying relationship



Strong entity

Weak entity

Identifier (Key)

- A key is an attribute or a combination of attributes that is used to identify records.
- Sometimes we might have to retrieve data from more than one table, in those cases we require to join tables with the help of keys
- The purpose of the key is to bind data together across tables without repeating all of the data in every table.
- The various types of key are mentioned in next slides, (For examples let suppose we have table -
- Student (‘student_ID’, ‘FName’, ’LName’, ’Course_ID’)

(I) Super Key – An attribute or a combination of attribute that is used to identify the records uniquely is known as Super Key. A table can have many Super Keys.

E.g. of Super Key

1 Student_ID

2 Student_ID, FName

3 Student_ID, FName, Lname

4 Student_ID, FName, Lname, course_ID

5 Fname, LName

So on as any combination which can identify the records uniquely will be a Super Key.

(II) Candidate Key – It can be defined as minimal Super Key or irreducible Super Key. In other words an attribute or a combination of attribute that identifies the record uniquely but none of its proper subsets can identify the records uniquely.

One can describe a candidate key as a super key that contains only the minimum number of columns necessary to determine uniqueness.

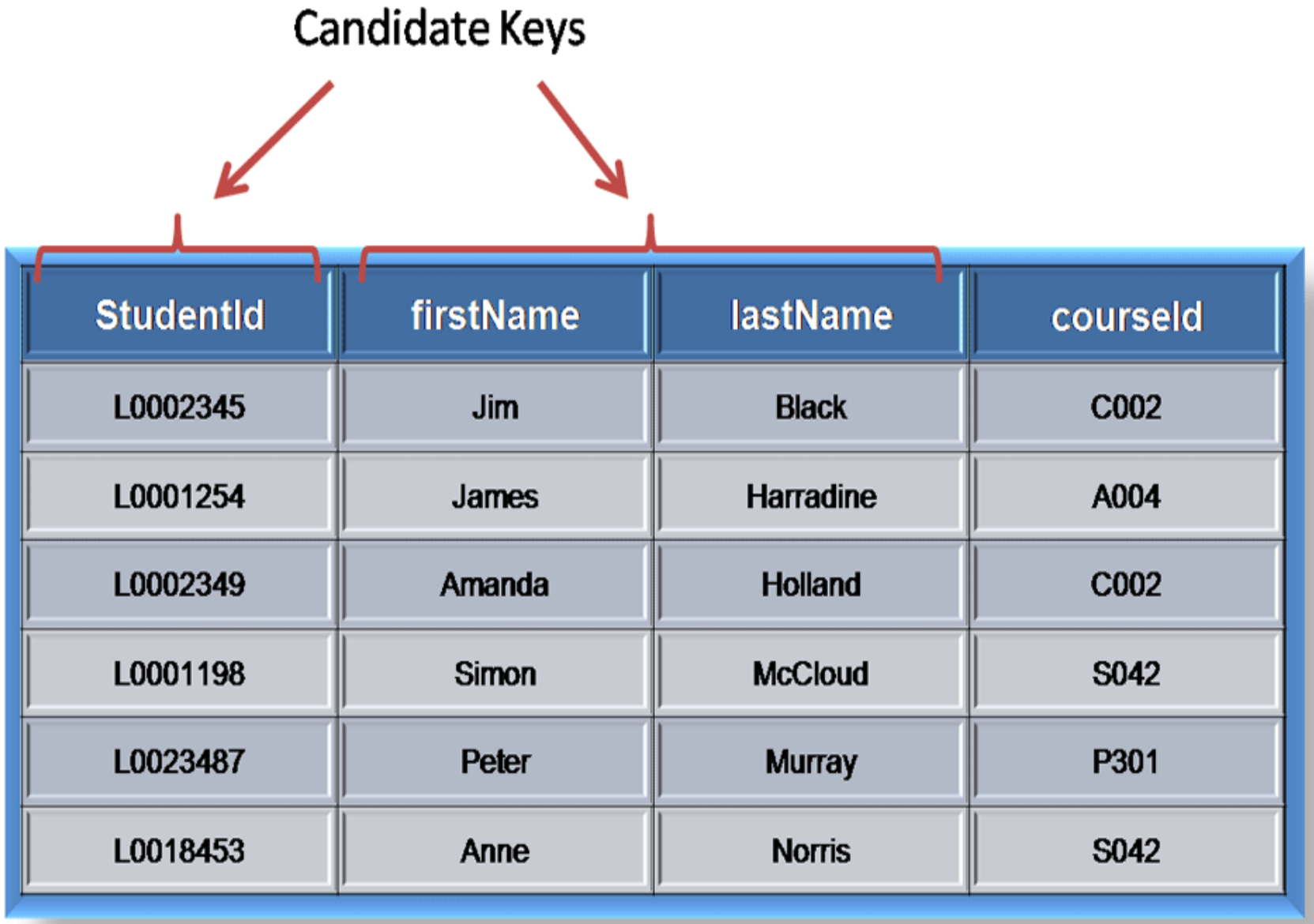
E.g. of Candidate Key

1. Student_ID that uniquely identifies the students in a student table

2. FName, Lname **when combined, uniquely identify the student in a student table**

These would both be candidate keys.

- In order to be eligible for a candidate key it must pass certain criteria.
- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table


Candidate Keys
StudentId
firstName
lastName
courseId
L0002345
Jim
Black
C002
L0001254
James
Harradine
A004
L0002349
Amanda
Holland
C002
L0001198
Simon
McCloud
S042
L0023487
Peter
Murray
P301
L0018453
Anne
Norris
S042

(III) Primary Key – A Candidate Key that is used by the database designer for unique identification of each row in a table is known as Primary Key. A Primary Key can consist of one or more attributes of a table.

As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table.

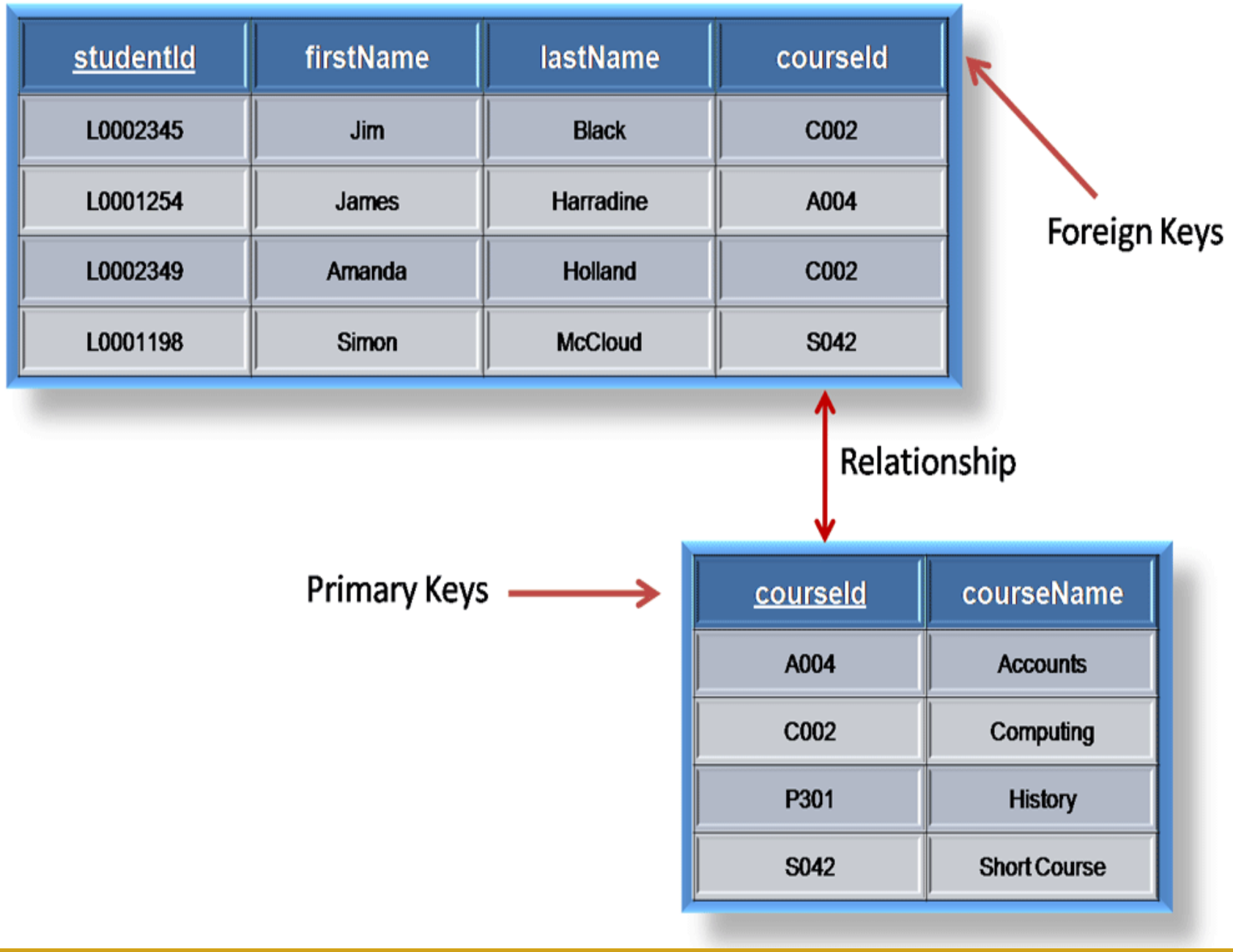
E.g. of Primary Key –

a student id might be a primary key in a student table

Primary Keys

StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

(IV) Foreign Key – A foreign key is an attribute or combination of attribute in one base table that points to the candidate key (generally it is the primary key) of another table. The purpose of the foreign key is to ensure referential integrity of the data i.e. only values that are supposed to appear in the database are permitted.


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

(V) Composite Key – If we use multiple attributes to create a Primary Key then that Primary Key is called Composite Key (also called a Compound Key or Concatenated Key).

E.g. of Composite Key, if we have used "FName, LName" as a Primary Key then it will be our Composite Key.

When choosing a primary key from the pool of candidate keys always choose a single simple key over a composite key.

(VI) Alternate Key – Alternate Key can be any of the Candidate Keys except for the Primary Key.

E.g. of Alternate Key is “FName, LName” as it is the only other Candidate Key which is not a Primary Key

- **(VII) Secondary Key –** The attributes that are not even the Super Key but can be still used for identification of records (not unique) are known as Secondary Key.
  E.g. of Secondary Key can be Name, Address, Salary, Department_ID etc. as they can identify the records but they might not be unique.

E R diagram

- Entity-Relationship (E-R) Diagram
  - A detailed, logical representation of the entities, associations and data elements for an organization or business

Notation Guide

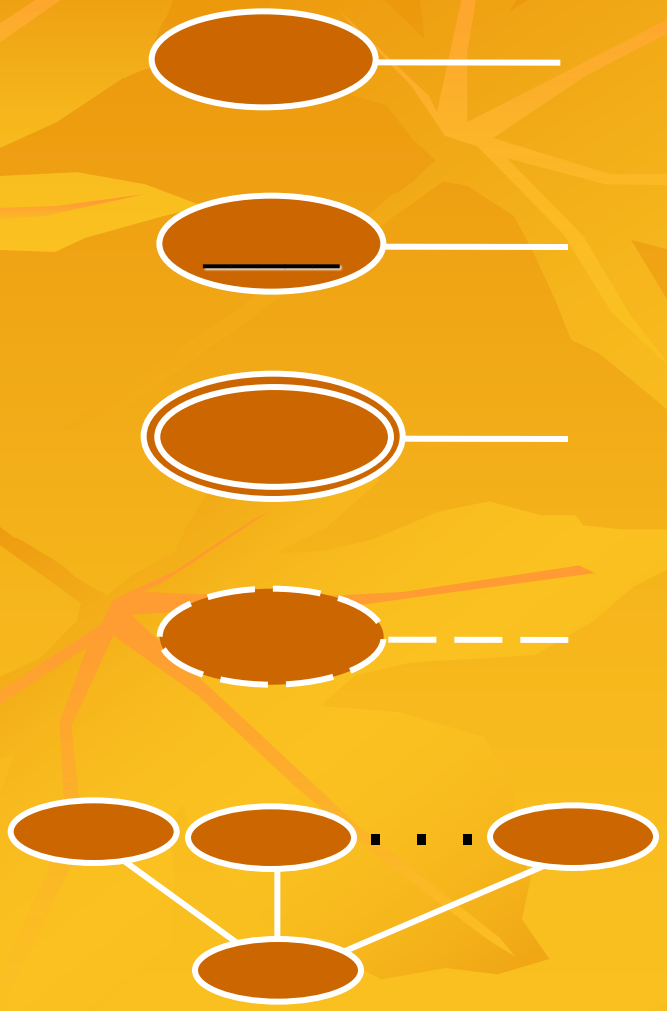
- ENTITY TYPE

- WEAK ENTITY TYPE

- RELATIONSHIP TYPE

- IDENTIFYING RELATIONSHIP TYPE

… *Notation Guide*



- ATTRIBUTE
- KEY ATTRIBUTE
- MULTIVALUED ATTRIBUTE
- DERIVED ATTRIBUTE
- COMPOSITE ATTRIBUTE

ER Diagram Basics

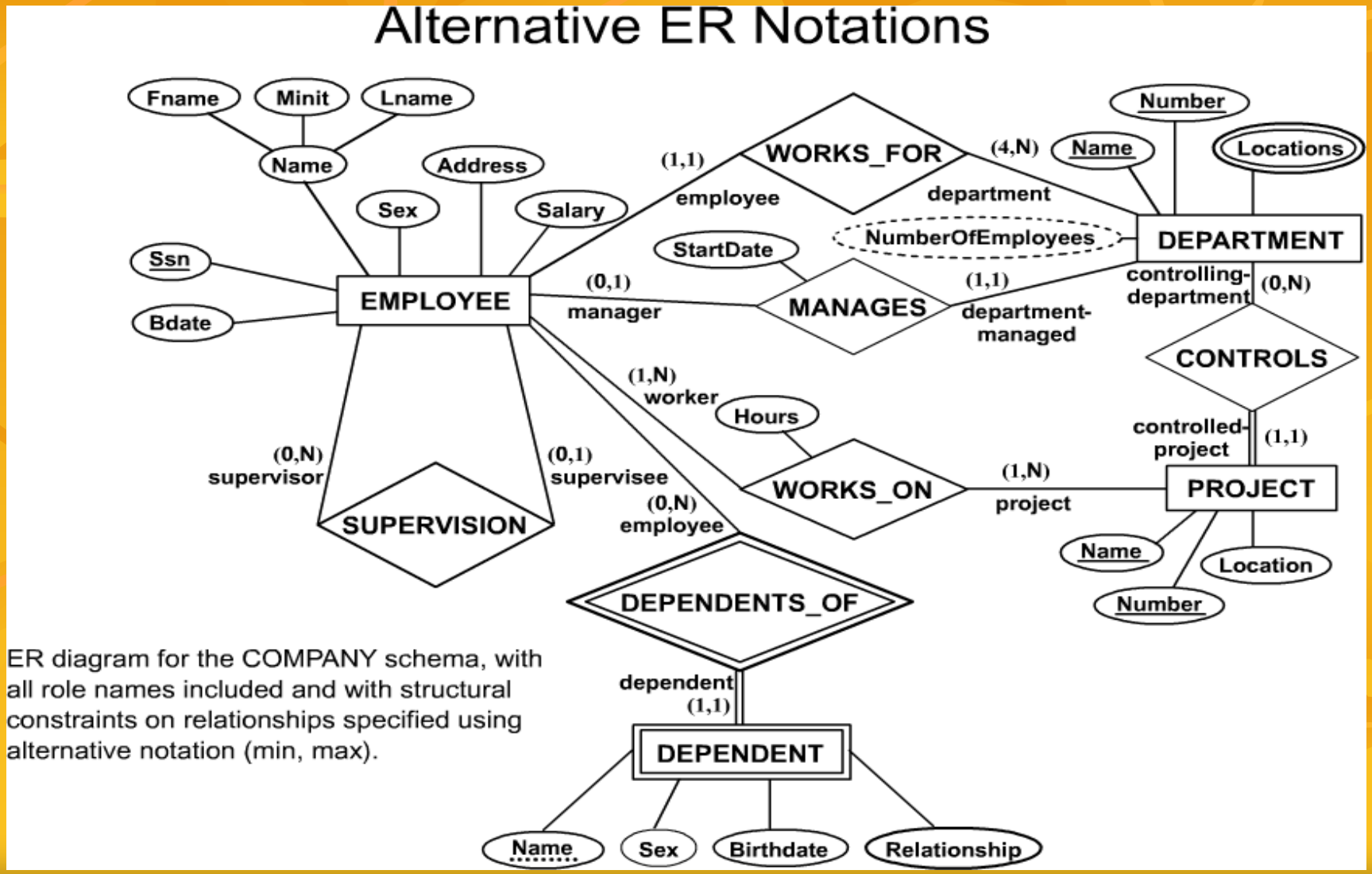
E-R Diagram Example

- Example: A library database contains a listing of authors that have written books on various subjects (one author per book). It also contains information about libraries that carry books on various subjects.

  Entity sets: authors, subjects, books, libraries

  Relationship sets: wrote, carry, indexed

Converting an E-R Diagram into tables



ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using alternative notation (min, max).

- **STEP 1:** For each non-weak entity, create a relation (or table) that includes all of the simple attributes of that entity.
- Do not include multivalued attributes or derived attributes at this time. If you have a composite attribute, include only the component attributes.
- Choose one of the candidate keys to be the primary key of the table.

EMPLOYEE

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary

DEPARTMENT

Name	Number

PROJECT

Name	Number	Location

- **STEP 2:** For each weak entity, create a relation that includes all simple attributes v of the weak entity.
- In addition, include as a foreign key attribute the primary key of the owning entity.
- The primary key of this relation will be the combination of the primary key of the owning entity and the partial key of the weak entity.

EXAMPLE: The relation for the weak entity DEPENDENT would look like this after step 2:

DEPENDENT

				FK
Name	Sex	Birthdate	Relationship	Emp_SSN

- **STEP 3:** For each binary 1:1 relationship, identify the two entities (E1, E2) that participate in that relationship. Take the primary key from E2 and include it as a foreign key in E1(double line). If the relationship has simple attributes, include those in the relation for E1.

DEPARTMENT (E1)

		FK	
Name	Number	Mgr_SSN	Mgr_Startdate

EMPLOYEE (E2) -- NOTE: The EMPLOYEE relation is unchanged from step 1

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary

- **STEP 4:** For each non-weak binary 1:N relationship, identify the entity E1 that is at the N-side (the "many" side) of the relationship. The other entity in the relationship is E2. Include as a foreign key in E1 the primary key of E2.

EXAMPLE: We have three 1:N relationships: *Works_for*, *Controls*, and *Supervision*.

Works_for: EMPLOYEE is on the N-side of the relationship, so after doing step 4 for *Works_for*, it will look like the following:

EMPLOYEE

								FK
Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Dept_Num

Controls: PROJECT is on the N-side of the relationship, so after doing step 4 for *Controls*, it will look like the following:

PROJECT

			FK
Name	Number	Location	Dept_Num

Supervision: EMPLOYEE is on the N-side of the relationship in the supervisee role, so after doing step 4 for *Supervision*, it will look like the following:

EMPLOYEE

								FK	FK
Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Dept_Num	Super_SSN

- **Step 5:** For each binary M:N relationship, create a new relation to represent the relationship. Include in this relation as foreign keys the primary keys of each of the entities that participates in the relationship. The combination of these foreign keys will make up the primary key for the relation.

EXAMPLE: We have one M:N relationship, *Works_On*. After step 5, we will have a relation for WORKS_ON that looks like this:

WORKS_ON

FK	FK	
Emp_SSN	Proj_Num	Hours

- **Step 6:** For each multivalued attribute, create a new relation that includes that attribute, plus the primary key of the entity to whom that attribute belongs as a foreign key. The primary key of this new relation will be the combination of the foreign key and the attribute itself.

EXAMPLE: We have only one multivalued attribute, the Locations attribute of DEPARTMENT. We create a relation called DEPT_LOCATIONS

DEPT_LOCATIONS

	FK
Location	Dept_Num

Enhanced-ER (EER) Model Concepts

- Includes all modeling concepts of basic ER
- Additional concepts: subclasses/superclasses, specialization/generalization, categories, attribute inheritance
- The resulting model is called the enhanced-ER or Extended ER (E2R or EER) model
- It is used to model applications more completely and accurately if needed
- It includes some object-oriented concepts, such as inheritance

Subclasses and Super classes

- An entity type may have additional meaningful subgroupings of its entities
- Example: EMPLOYEE may be further grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE,…
  - Each of these groupings is a subset of EMPLOYEE entities
  - Each is called a subclass of EMPLOYEE
  - EMPLOYEE is the superclass for each of these subclasses
- These are called superclass/subclass relationships.
- Example: EMPLOYEE/SECRETARY, EMPLOYEE/TECHNICIAN
- These are also called IS-A relationships (SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, …).

Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass *inherits* all attributes of the entity as a member of the superclass
- It also inherits all relationships

Specialization

This is the process of maximising the differences between members of an entity by identifying their distinguishing characteristics.

- Staff(staff_no,name,address,dob)
- Manager(bonus)
- Secretary(wp_skills)
- Sales_personnel(sales_area, car_allowance)

- ***Specialization*** constructs the lower level entity sets that are a subset of a higher level entity set.

Generalisation

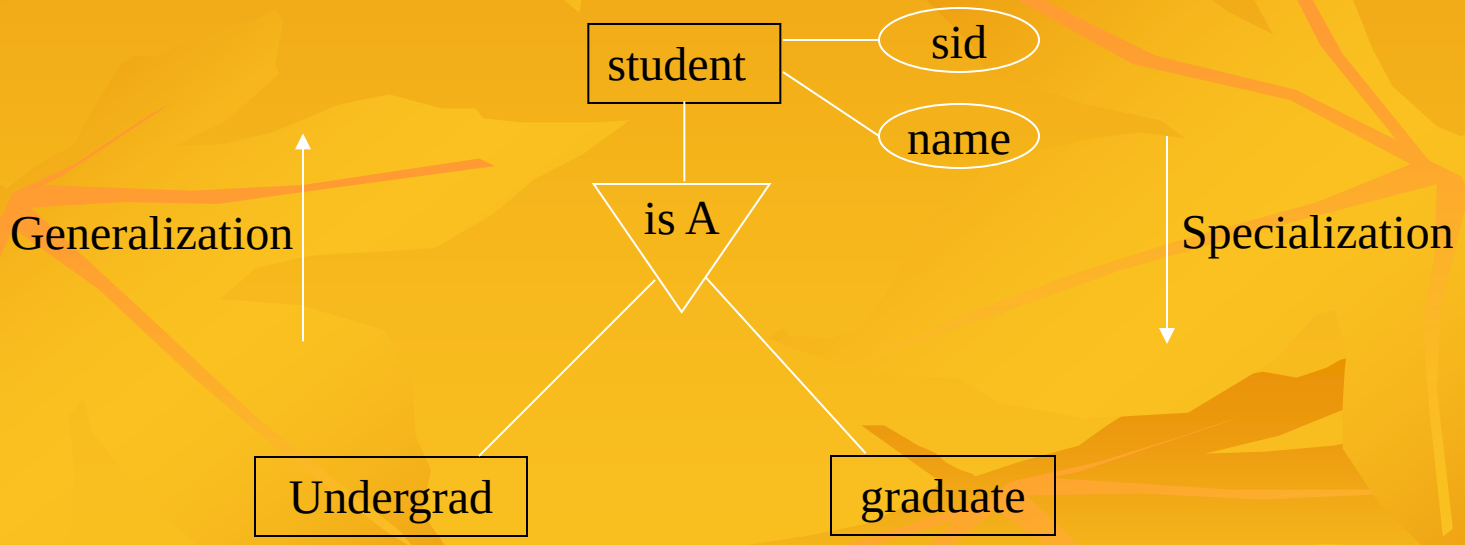
Generalisation is the process of minimising the differences between entities by identifying common features.

This is the identification of a generalised superclass from the original subclasses. This is the process of identifying the common attributes and relationships.

Generalization and Specialization

- Arrow pointing to the generalized superclass represents a generalization
- Arrows pointing to the specialized subclasses represent a specialization
- We do not use this notation because it is often subjective as to which process is more appropriate for a particular situation
- We advocate not drawing any arrows in these situations
- A superclass or subclass represents a set of entities
- Shown in rectangles in EER diagrams (as are entity types)
- Sometimes, all entity sets are simply called classes, whether they are entity types, superclasses, or subclasses

GENERALIZATION AND SPECIALIZATION

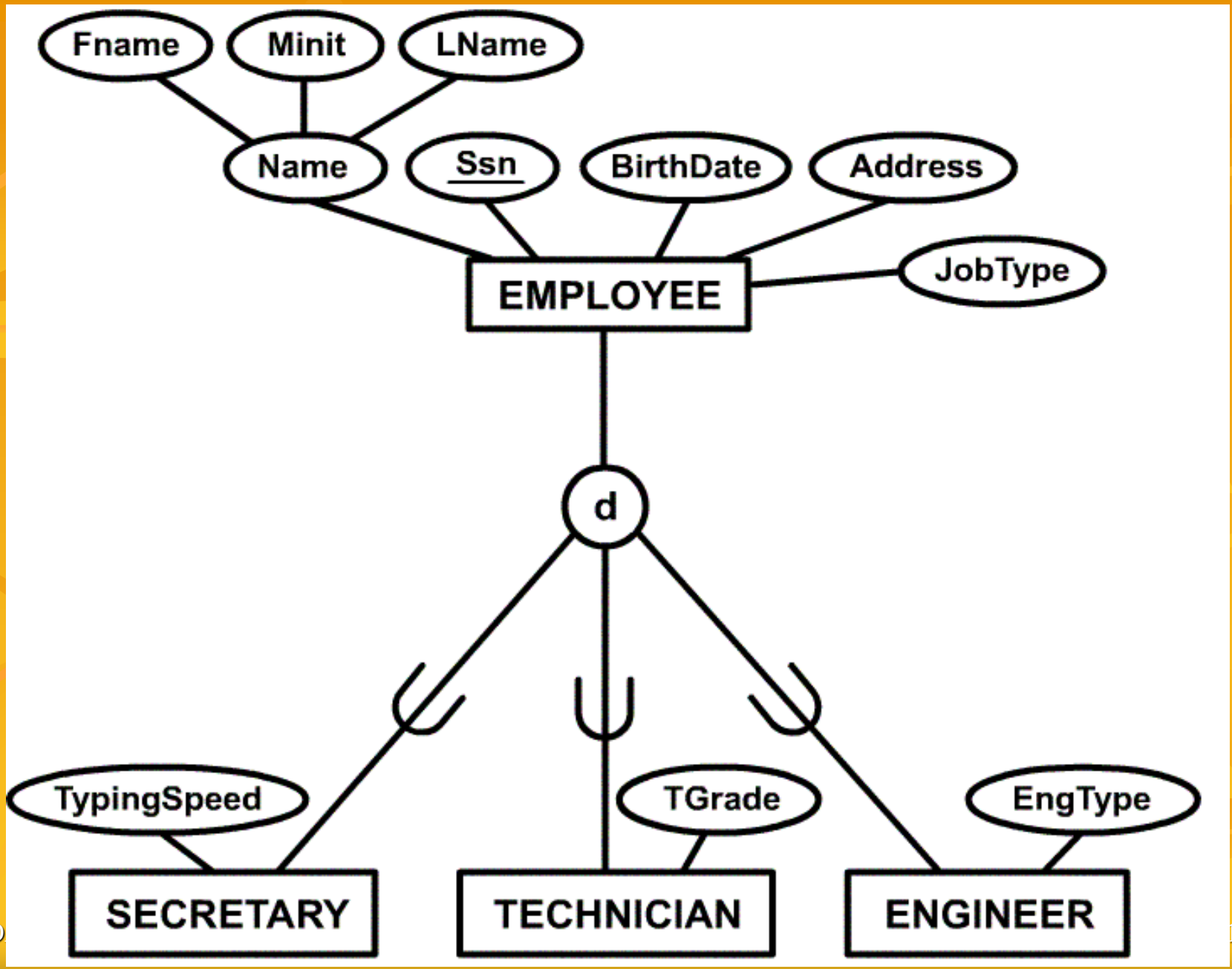


Constraints on Specialization and Generalization

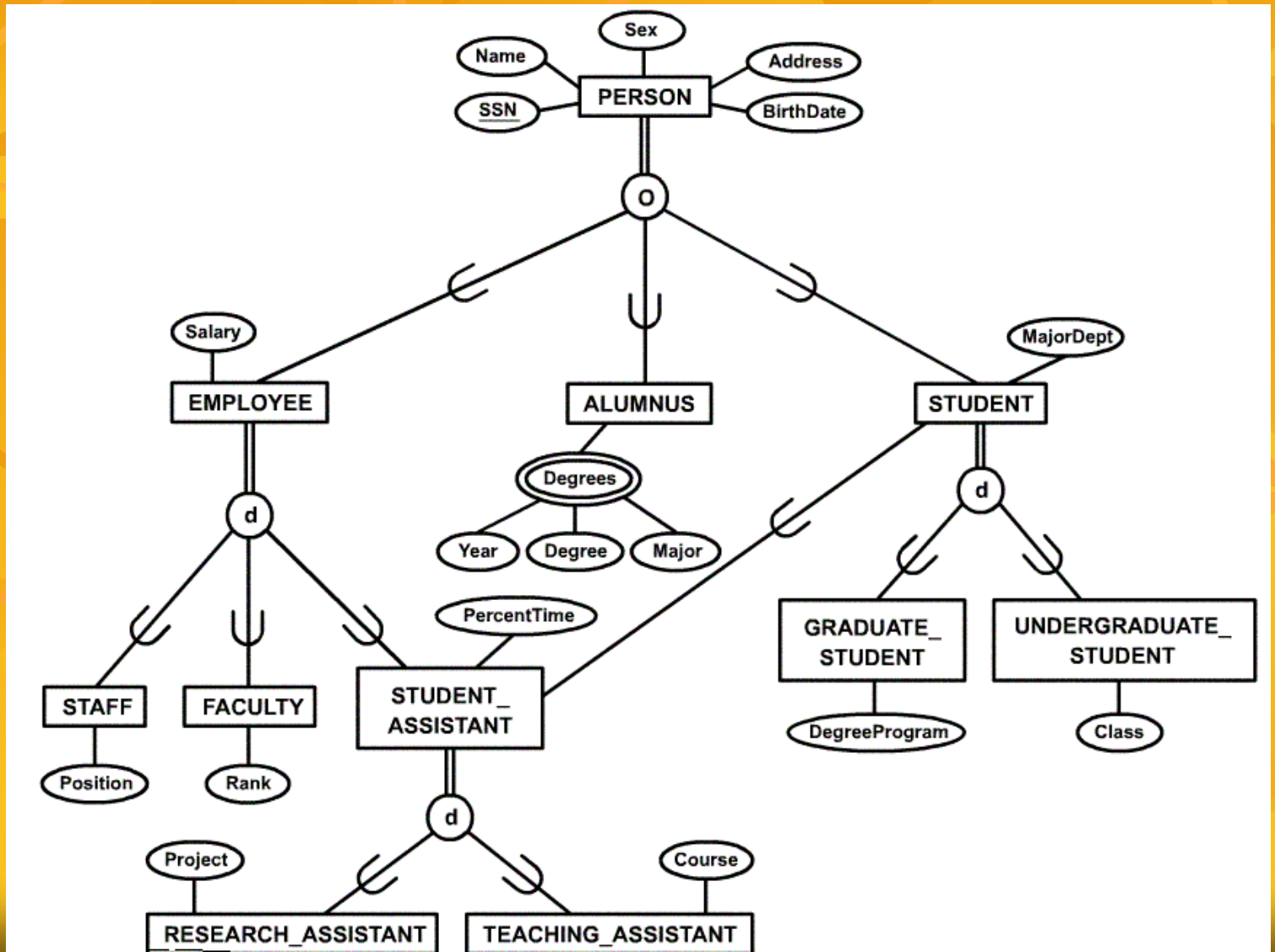
- Two conditions apply to a specialization/generalization:
- **Disjointness Constraint**:
  - Specifies that the subclasses of the specialization must be disjointed (an entity can be a member of at most one of the subclasses of the specialization)
  - Specified by d in EER diagram
  - If not disjointed, overlap; that is the same entity may be a member of more than one subclass of the specialization
  - Specified by o in EER diagram
- **Completeness Constraint**:
  - Total specifies that every entity in the superclass must be a member of some subclass in the specialization/ generalization
  - Shown in EER diagrams by a double line
  - Partial allows an entity not to belong to any of the subclasses
  - Shown in EER diagrams by a single line

Constraints on Specialization and Generalization

- Hence, we have four types of specialization/generalization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial
- Note: Generalization usually is total because the superclass is derived from the subclasses.


Fname
Minit
LName
Name
Ssn
BirthDate
Address
JobType
EMPLOYEE
d
TypingSpeed
TGrade
EngType
SECRETARY
TECHNICIAN
ENGINEER

Specialization / Generalization Example (UNIVERSITY)



Relational model

Objective:

- What is relational model?
- Codd's rule
- Different types of constraints
  - Nulls, Entity, Referential Integrities, Enterprise Constraints
- Schema diagram

What is Relational Model?

- Relational model is most widely used data model for commercial data-processing. The reason it's used so much is, because it's simple and easy to maintain.
- The model is based on a collection of tables. Users of the database can create tables, insert new tables or modify existing tables.

History of Relational Modeling

- Introduced by Ted Codd in 1970
- Ted Codd was an **IBM** Researcher laid the foundation for database theory
- Many database concepts & products based on his model

Relational Model

- Relational model provides
- A simple, limited approach to structuring data, yet is reasonably flexible
- Easy to query without programming
- A limited, yet useful, collection of operations on data
- Good design is essential
- Integrity is essential

Relational Model Basic

The relational model gives us a single way to represent data: as a two-dimensional table called a relation.

- Attributes
- Schemas
- Tuples
- Domains

Attributes

Attributes of a relation serve as names for the columns of the relation. Usually, an attribute describes the meaning of entries in the column below.

Table = *relation*.
Column headers = *attributes*.

Title	Year	Length
Star Wars	1977	124
Might Ducks	1991	104
Wayne's World	1992	95

Schemas

- The name of a relation and the set of attributes for a relation is called a schema.
- We show the schema for the relation with the relation name followed by a parenthesized list of its attributes.
- So the schema for previous slide is

Movies (title, year, length)

Tuples

- The rows of a relation, other than the header row containing The attribute names are called tuples.
- A tuple has one component for each attribute of the relation.

Tuple

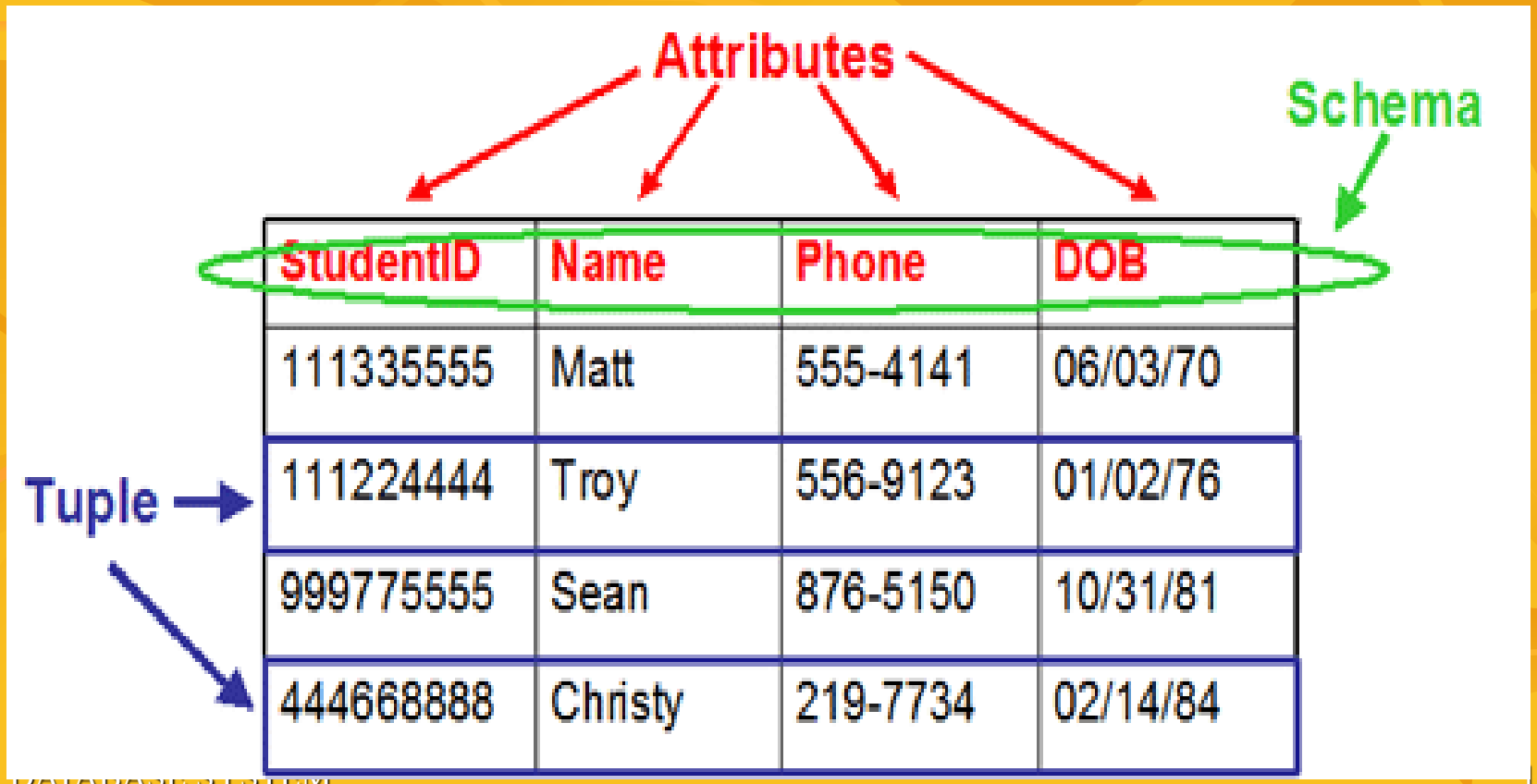
Title	Year	Length
Star Wars	1977	124
Might Ducks	1991	104
Wayne's World	1992	95

Domains

- Each attribute of a relation is associated with a particular elementary type called domain.
- The components of any tuple of the relation must have, in each component, a value that belongs to the domain of the corresponding column.
- Example:
  - with **`title`** string is associated
  - with **`year`** integer is associated

Relational Model Example

- Represent data as a two-dimensional table called a relation



StudentID	Name	Phone	DOB
111335555	Matt	555-4141	06/03/70
111224444	Troy	556-9123	01/02/76
999775555	Sean	876-5150	10/31/81
444668888	Christy	219-7734	02/14/84

Codd's Rules

- **Codd's 12 rules** are a set of rules proposed by E. F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered *relational*
- For a system to qualify as a relational database management system (RDBMS), that system must use its *relational* facilities to *manage* the *database*.

Codd's Rules

- **Rule 1:** The *information rule*:
  - All information in the database is to be represented in one and only one way, namely by values in column positions within rows of tables.
- **Rule 2:** The *guaranteed access rule*:
  - All data must be accessible. This rule is essentially a restatement of the fundamental requirement for primary keys.
- **Rule 3:** *Systematic treatment of null values*:
  - The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information", distinct from all regular values, and independent of data type. It is also implied that such representations must be manipulated by the DBMS in a systematic way.

Codd's Rules

- **Rule 4:** *Active online catalog based on the relational model*:
  - The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language. That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.
- **Rule 5:** The *comprehensive data sublanguage rule*:
  - The system must support at least one relational language that
    - Has a linear syntax
    - Can be used both interactively and within application programs,
    - Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).
- **Rule 6:** The *view updating rule*:
  - All views that are theoretically updatable must be updatable by the system.

Codd's Rules

- **Rule 7:** *High-level insert, update, and delete*:
  - The system must support set-at-a-time *insert*, *update*, and *delete* operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.
- **Rule 8:** *Physical data independence*:
  - Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure.
- **Rule 9:** *Logical data independence*:
  - Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure. Logical data independence is more difficult to achieve than physical data independence.

Codd's Rules

- **Rule 10:** *Integrity independence*:
  - Integrity constraints must be specified separately from application programs and stored in the catalog. It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.
- **Rule 11:** *Distribution independence*:
  - The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully :
    - when a distributed version of the DBMS is first introduced; and
    - when existing distributed data are redistributed around the system.
- **Rule 12:** The *nonsubversion rule*:
  - If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint.

Integrity Constraints

- Data integrity allows to define certain data quality requirements that the data in the database needs to meet. If a user tries to insert data that doesn't meet these requirements, Oracle will not allow so.
- Integrity constraints provides a way of ensuring that changes made to the database by the authorized users do not result in a loss of data consistency

Integrity Constraints

- **Constraint types**

  There are many integrity constraints in database
  - Null/ Not Null
  - Unique
  - Entity integrity
  - Referential integrity
  - Check
  - Default

- Not Null:
  - A column in a table can be specified **not null**. It's not possible to insert a null in such a column.
- Unique:
  - The unique constraint doesn't allow duplicate values in a column. If the unique constraint encompasses two or more columns, no two equal combinations are allowed.
- Check :
  - A check constraint allows to state a minimum requirement for the value in a column. E.g. column in table allows only numbers that are between 0 and 100

- Entity integrity constraint (Primary key):
  - To identify each row in a table, the table must have a primary key.
  - A *primary key* combines a unique and a not null constraint. A table can have at most one primary key
- Referential integrity constraint (Foreign key)
  - On a column ensures that the value in that column is found in the primary key of another table.
  - Student(studno,name,tutor,year)
  - Staff(lecturer,roomno,appraiser)

STUDENT

studno	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

STAFF

lecturer	roomno	appraiser
kahn	IT206	watson
bush	2.26	capon
goble	2.82	capon
zobel	2.34	watson
watson	IT212	barringer
woods	IT204	barringer
capon	A14	watson
lindsey	2.10	woods
barringer	2.125	*null*

Relational databases schema and schema diagram

- A relational database schema s is a set of relation schemas s={R1,R2, R3, …RM} and a set of integrity constraints
- The figure in next slide shows a relational database schema

  COMPANY = {Employee, department, dept_location, project, works_on}
- The underlined attributes represent primary keys

A simplified COMPANY relational database schema

Figure 14.1 Simplified version of the COMPANY relational database schema.