Database Design Entity-Relationship (ER) Modeling

COSC 304 – Introduction to Database Systems

Database Design

The ability to design databases and associated applications is critical to the success of the modern enterprise.

Database design requires understanding both the operational and business requirements of an organization as well as the ability to model and realize those requirements in a database.

Developing database and information systems is performed using a ***development lifecycle***, which consists of a series of steps. Data analysts may have access to design documents to help understand how to use the data properly.

The Importance of Database Design

Some statistics on software projects:

- 80 - 90% do not meet their performance goals
- 80% delivered late and over budget
- 40% fail or abandoned
- 10 - 20% meet all their criteria for success
- Have you been on a project that failed? A) Yes B) No

The primary reasons for failure are improper requirements specifications, development methodologies, and design techniques.

Software and System Development

Software development follows an iterative process with steps:

- ***Specification*** – capturing user and system requirements, goals, and timelines
  - *Top-down design* by specifying entities, attributes, and relationships.
- ***Design*** – develops models for system architecture and behavior. Design database and the associated application. Special focus on transactions and UI.
  - Recommend: Solid DB design before prototyping application.
- ***Implementation*** – build database statements and program code.
- ***Testing*** - executing programs to determine errors and issues.
- ***Maintenance*** – monitoring and maintaining the system after installation.
  - DBMS maintenance includes monitoring performance, security, and upgrades.

Specification

Specification involves:

- understanding how the project fits into the organization
- project goals and success outcomes
- project timelines and deliverables
- measurement criteria for determining project success
- information on users and user requirements
- standards for database/application development that must be followed
- documentation of privacy and security concerns
- legal issues including copyright when dealing with outside developers
- information on how the project will interface with other systems

Specification: Mission Statements

The ***mission statement*** specifies the major project objectives with defined metrics to evaluate if successfully completed.

NASA's mission statement when going to the moon:

"I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to Earth." (John F. Kennedy May 25, 1961)

- NASA fulfilled that goal on July 20, 1969, when Apollo 11's lunar module *Eagle* touched down in the Sea of Tranquility, with Neil Armstrong and Buzz Aldrin aboard. A dozen men would walk on the moon before the Apollo program ended. The last of those men, Gene Cernan, left the desolate lunar surface with these words: "We leave as we came and, God willing, as we shall return, with peace and hope for all mankind."

Specification: The "Project Champion"

The "**Project Champion**" is a manager or senior IT person who is the project's promoter and backer.

Many projects fail because no one takes ownership of them.

- Consequently, they take too long, go over budget, and are never deployed effectively.
- When hiring outside consultants, make sure somebody in the organization is the Project Champion.
- For internal projects, a Project Champion is especially important as there are always conflicts over money, developer time, and political issues on making users work on new applications.

Bottom line: If no one is willing to be the champion for a project, it is likely that project will not achieve its goals.

Specification: Requirements Gathering

Requirements gathering collects details on organizational processes and user issues.

- Often the organization itself does not know this information, and it can only be determined by collecting it from user interviews.
- Through user interviews identify:
  - Who are the users? Group them into classes.
  - What do the users do now? (existing systems/processes)
  - What are the complaints and possibilities for improvement?
- Determine the data used by the organization, identify data relationships, and determine how data is used and generated.
  - Identify unique fields (keys)
  - Determine data dependencies, relationships, and constraints (high-level)
  - Estimate the data sizes and their growth rates

Database Design

Database design is divided into three phases:

- ***Conceptual database design*** - models the collected information at a high-level of abstraction without using a particular data model or DBMS.
  - *Top-down design* by specifying entities, attributes, and relationships.
- ***Logical database design*** - constructs a model of the information in the domain using a particular data model, but independent of the DBMS.
  - Typically use relational model but may also use object-oriented, graphs, JSON, or XML.
  - Since logical design selects a data model, it is now possible to model the information using the features of that model (e.g. keys and foreign keys in relational model).
- ***Physical database design*** - constructs a physical model of information in a given data model for a particular DBMS. Selects a database system and determines how to represent the logical model on that DBMS.
  - E.g. creating tables, indexes, security, data partitioning
  - Physical database design is ***how***, and logical database design is the ***what***.
  - Select a DBMS based on features, performance, price, and interoperability.

Database People: DA and DBA

We have seen these two database people before:

- ***Database administrator*** (***DBA***) - responsible for installing, maintaining, and configuring the DBMS software.
- ***Data administrator*** (***DA***) - responsible for organizational policies on data creation, security, and planning.

The DA is involved in the early phases of design including planning the project and conceptual and logical design.

The DBA performs the physical design and actively manages deployed, production systems.

Another common position is a (data) ***architect*** that selects systems to use, makes design decisions, and evaluates current and future systems at the architectural level.

Entity-Relationship Modeling

Entity-relationship modeling is a top-down approach to conceptual database design that models the data as entities, attributes, and relationships.

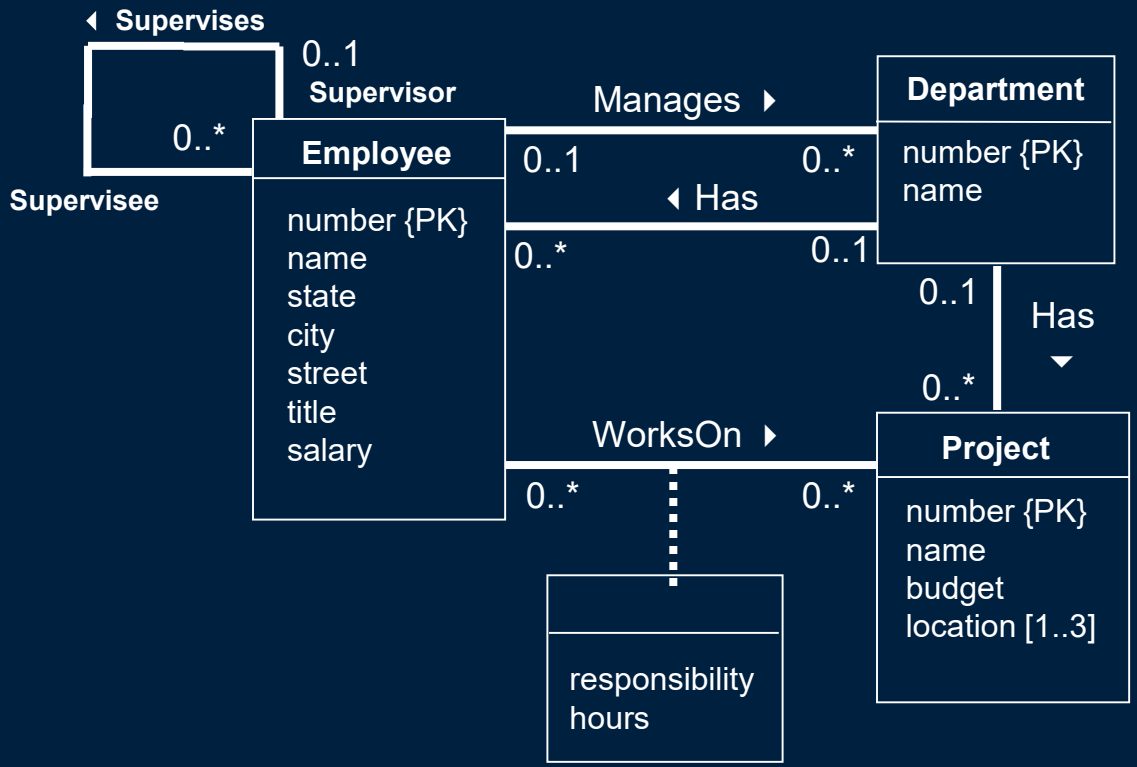
- The entity-relationship (ER) model was proposed by Peter Chen in 1976. We will perform ER modeling using Unified Modeling Language (UML) syntax.

The ER model refines entities and relationships by including properties of entities and relationships called *attributes*, and by defining *constraints* on entities, relationships, and attributes.

The ER model conveys knowledge at a high-level (conceptual level) which is suitable for interaction with technical and non-technical users.

Since the ER model is data model independent, it can later be converted into the desired logical model (e.g. relational model).

ER Model Example in UML notation



Entity Types

An ***entity type*** is a group of objects with the same properties which are identified as having an independent existence.

- An entity type does not always have to be a physical real-world object such as a person or department. It can be an abstract concept such as a project or job.

An ***entity instance*** is a particular example or occurrence of an entity type.

- For example, an entity type is Employee. A entity instance is 'E1 - John Doe'.

Representing Entity Types

Entity types are represented by rectangles with the name of the entity type in the rectangle.

Examples:

- An entity type name is normally a singular noun.
  - That is, use Person instead of People, Project instead of Projects, etc.
- The first letter of each word in the entity name is capitalized by convention.

Entities Question

Question: How many of the following statements are ***true***?

1) Entity types are represented using a rectangle box.

2) An entity is always a physical object.

3) An entity type is named using a plural noun.

4) Employee number is an entity.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Relationships

A ***relationship type*** is a set of associations among entity types. Each relationship type has a name that describes it.

A ***relationship instance*** is a particular occurrence of a relationship type that relates entity instances.

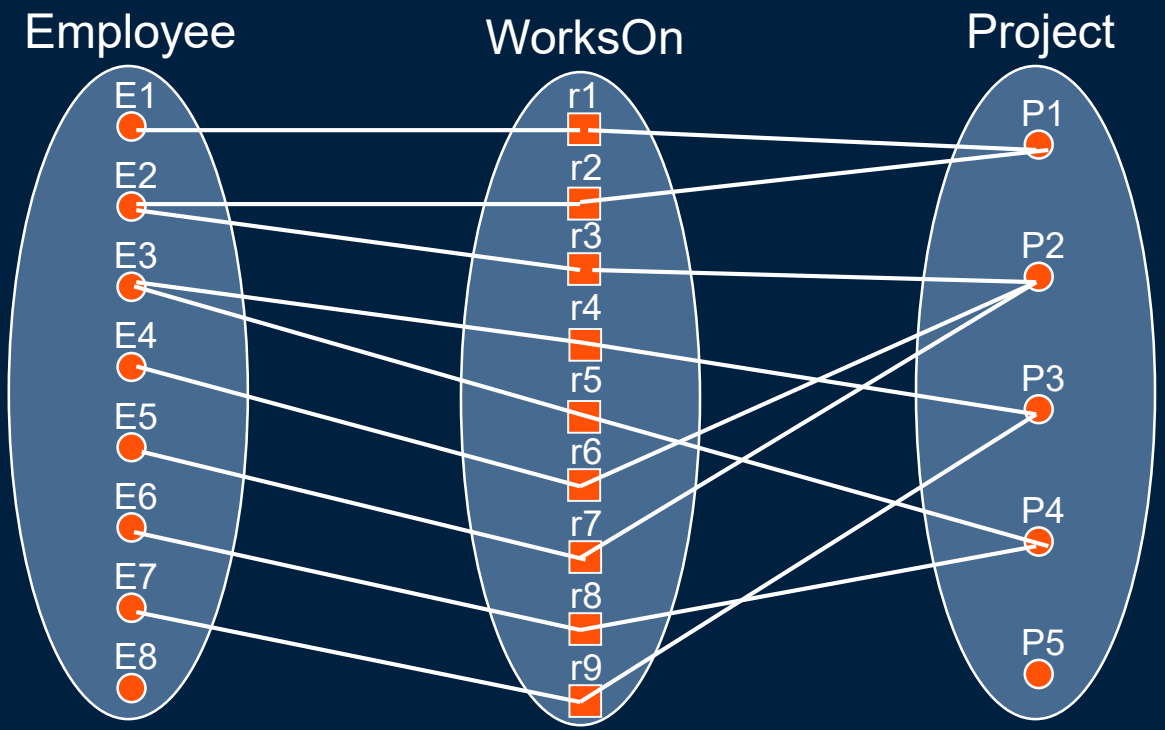
There can be more than one relationship between two entity types.

Representing Relationship Types

The relationship type is represented as a labeled edge between the two entity types. The label is applied only in one direction so an arrow indicates the correct way to read it.

- A relationship type name is normally a verb or verb phrase.
- The first letter of each word in the name is capitalized.
- **Do not put arrows on either end of the line.**

Visualizing Relationships



Note: This is an example of a many-to-many relationship. A project can have more than one employee, and an employee can work on more than one project.

Relationship Degree

The ***degree of a relationship type*** is the number of entity types participating in the relationship.

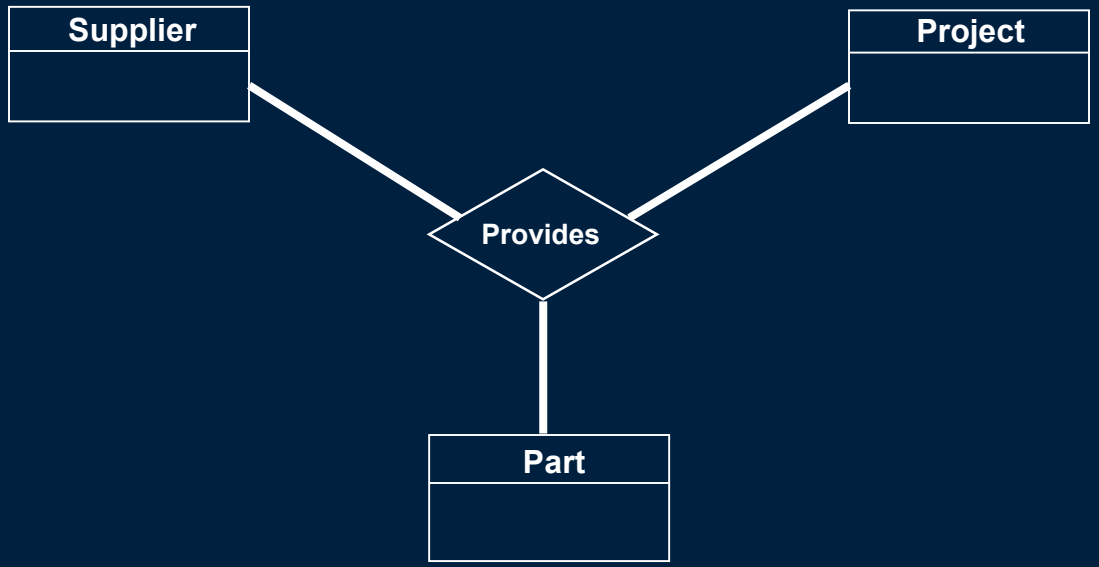
- For example, WorksOn is a relationship type of degree two as the two participating entity types are Employee and Project.
  - Note: This is not the same as degree of a relation which was the number of attributes in a relation.

Relationships of degree two are *binary*, of degree three are *ternary*, and of degree four are *quaternary*.

- Relationships of arbitrary degree N are called *n*-ary.

Use a diamond to represent relationships of degree higher than two.

Ternary Relationship Type Example



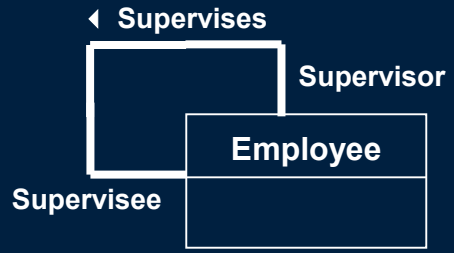
A project may require a part from multiple different suppliers.

Recursive Relationships

A ***recursive relationship*** is a relationship type where the same entity type participates more than once in different roles.

- For example, an employee has a supervisor. The supervisor is also an employee. Each *role* has a *role name*.

Example:



- The degree of a recursive relationship is two as the same entity type participates twice in the relationship.
  - It is possible for an entity type to be in a relationship more than twice.

Relationship Question

Question: How many of the following statements are ***true***?

1) Relationships are represented using a directed edge (with arrows).

2) A relationship is typically named using a verb.

3) It is not possible to have a relationship of degree 1.

4) The degree of a relationship is the number of attributes it has.

5) A diamond is used to represent a relationship of degree larger than two.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Attributes

An ***attribute*** is a property of an entity type or a relationship type.

- For example, entity type Employee has attributes name, salary, title, etc.

Some rules:

- By convention, attribute names begin with a lower case letter.
- Attribute names are typically adjectives.
- Each attribute has a *domain*, which is the set of allowable values for the attribute (data type).
- An attribute may be single valued or have multi-values.
- An attribute may be simple if it contains a single component (e.g. salary) or composite if it contains multiple components (e.g. address).
  - Question: Is the name attribute of Employee simple or composite?
- A ***derived attribute*** is an attribute whose value is calculated from other attributes but is not physically stored.

Representing Attributes

In UML attributes are listed in the rectangle for their entity. Tags are used to denote any special features of the attributes.

- multi-valued attribute: *attributeName* [*minVals*..*maxVals*]
  - e.g. phoneNumber [1..3]
- derived attribute: */attributeName* (e.g. /totalEmp)
  - Derived attribute is not stored in database (calculated on demand)
- partial primary key: {PPK} – for key field of weak entity
  - A ***weak entity type*** is an entity type whose existence depends on another entity type.

Attribute Question

Question: How many of the following statements are ***true***?

1) Attributes are properties of either entities or relationships.

2) An attribute may be multi-valued.

3) A composite attribute contains two or more components.

4) Each attribute has a domain representing its data type.

5) Attribute names are typically verbs.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Representing Attributes and Keys

A ***candidate key*** is a minimal set of attributes that uniquely identifies each instance of an entity type.

A ***primary key*** is a candidate key that is selected by the designer to identify each instance of an entity type.

- Attributes labeled with `{PK}` in diagram.
- Note: No foreign keys in ER model but may see `{FK}` notation in logical diagram.

A ***composite key*** is a key that consists of two or more attributes.

Key Question

Question: How many of the following statements are ***true***?

1) It is possible to have two candidate keys with different numbers of attributes.

2) A composite key has more than 1 attribute.

3) The computer picks the primary key used in the design.

4) A relationship has a primary key.

5) An attribute has a primary key.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Attributes on Relationships

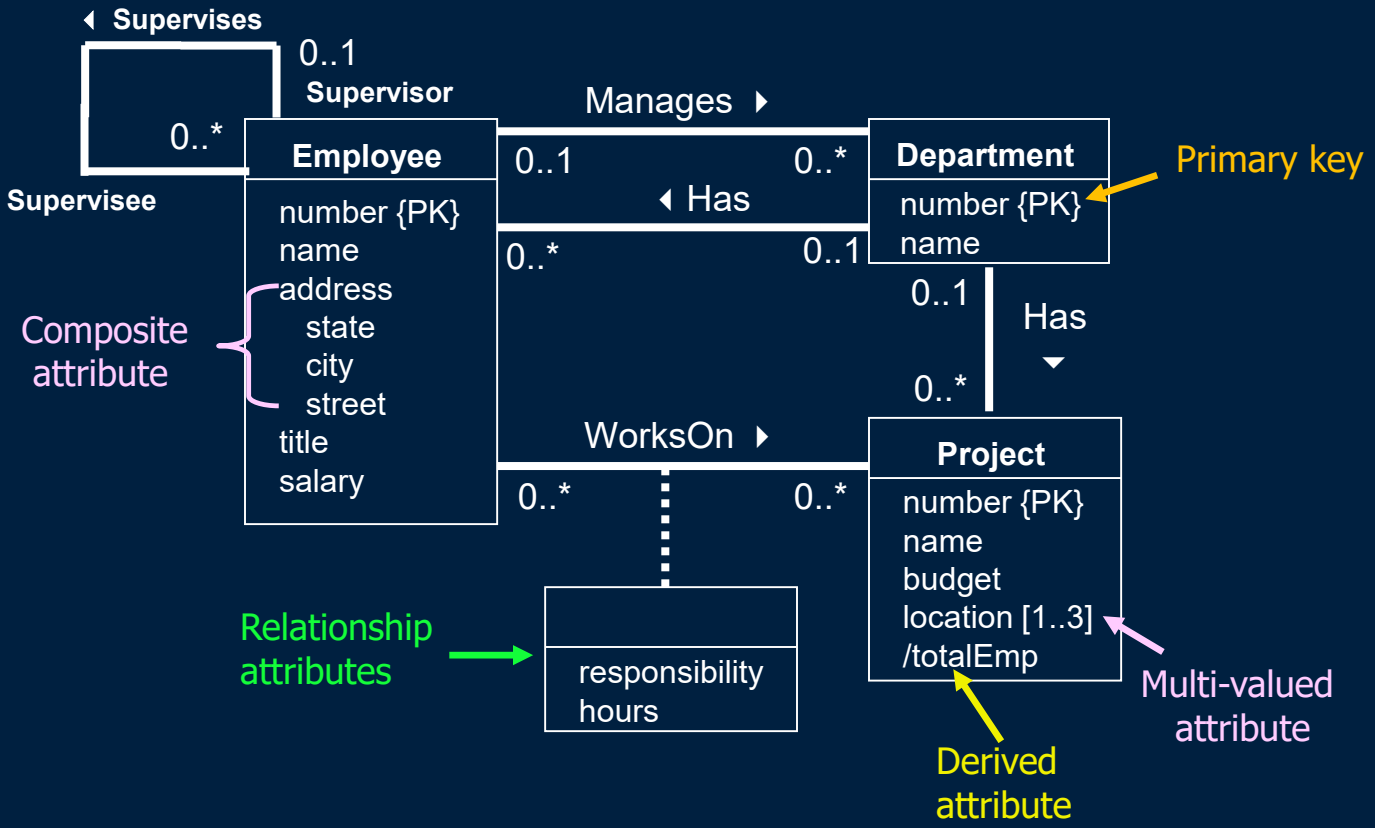
An attribute may be associated with a relationship type.

For example, the WorksOn relationship type has two attributes: responsibility and hours.

Note that these two attributes belong to the relationship and cannot belong to either of the two entities individually (as they would not exist without the relationship).

Relationship attributes are represented as a separate box connected to the relationship using a dotted line.

Attributes in UML Notation



ER Design Question #1

Construct a university database where:

- Each student has an id, name, sex, birth date, and GPA.
- Each professor has a name and is in a department.
- Each department offers courses and has professors. A department has a name and a building location.
- Each course has a name and number and may have multiple sections.
- Each section is taught by a professor and has a section number.
- Students enroll in sections of courses. They may only enroll in a course once (and in a single section). A student receives a grade for each of their course sections.

Conclusion

Database design is divided into three phases:

- Conceptual database design
- Logical database design
- Physical database design

Effective requirements gathering is an important skill to master. Projects should have a well-defined mission statement and project champion to increase their probability of success.

ER (conceptual) design is performed at a high-level of abstraction involving entities, relationships, and attributes.

- There are a variety of different diagram syntax. We used UML syntax.

Objectives

- Describe the three steps in database design including the results of each step.
- Describe differences between conceptual, logical, and physical data models.
- Describe how the roles of DBA and DA fit into database design. What do these people do?
- Define and identify on an ER diagram: entity type, relationship type, degree of a relationship, recursive relationship, attribute, multi-valued attribute, derived attribute
- Define and identify on an ER diagram: primary key, partial primary key

Be able to model a domain explained in an English paragraph in an ER diagram using UML notation. 


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Database Design Entity-Relationship (ER) Modeling: Relationships

COSC 304 – Introduction to Database Systems

ER Modeling
Relationship Cardinalities

Relationship cardinalities or ***multiplicities*** are used to restrict how entity types participate in relationships in order to model real-world constraints.

The ***multiplicity*** is the number of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.

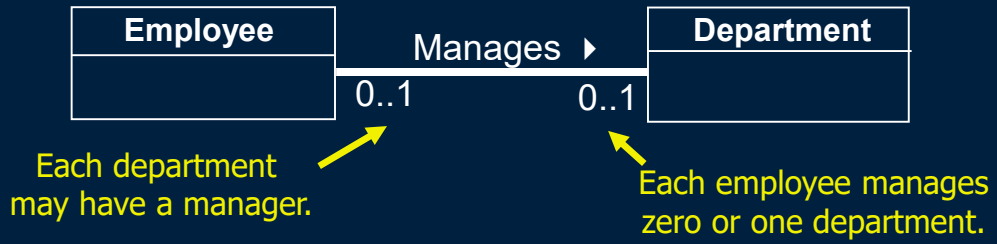
For binary relationships, there are three common types:

- one-to-one (1:1)
- one-to-many (1:* or 1:N)
- many-to-many (*:* or N:M)

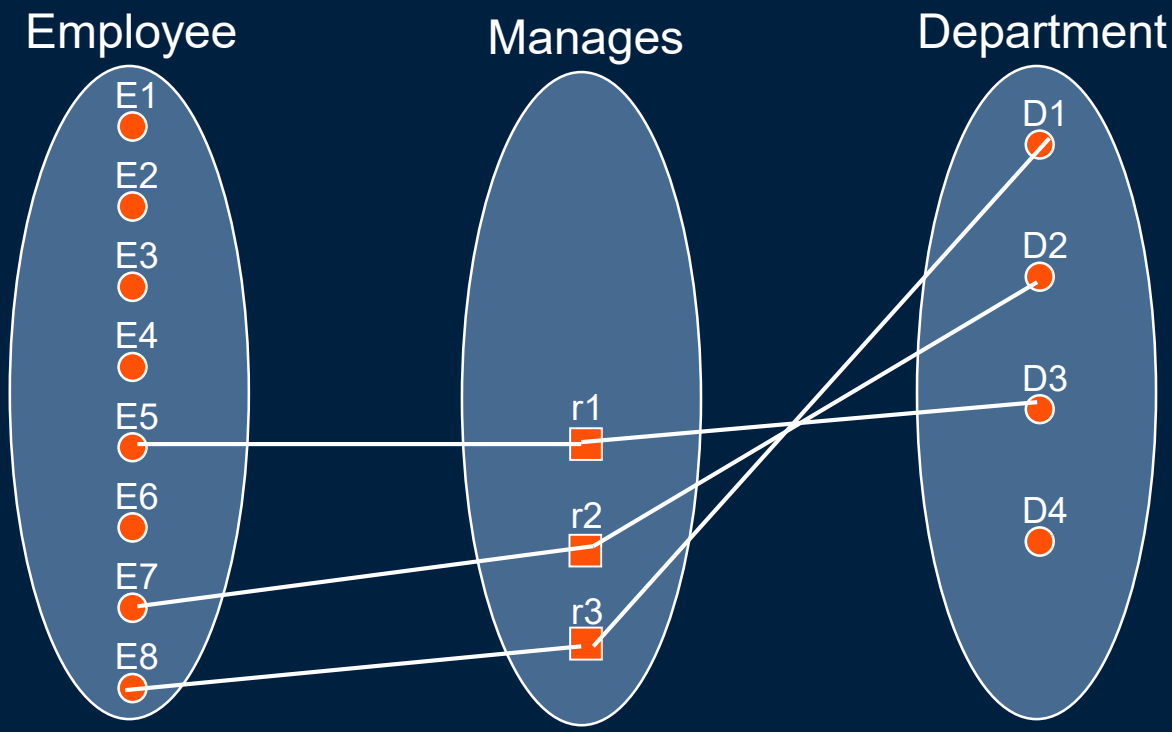
One-to-One Relationships

In an one-to-one relationship, each instance of an entity class E1 can be associated with **at most one** instance of an entity class E2 and vice versa.

Example: A department may have only one manager, and a manager may manage only one department.



One-to-One Relationship Example



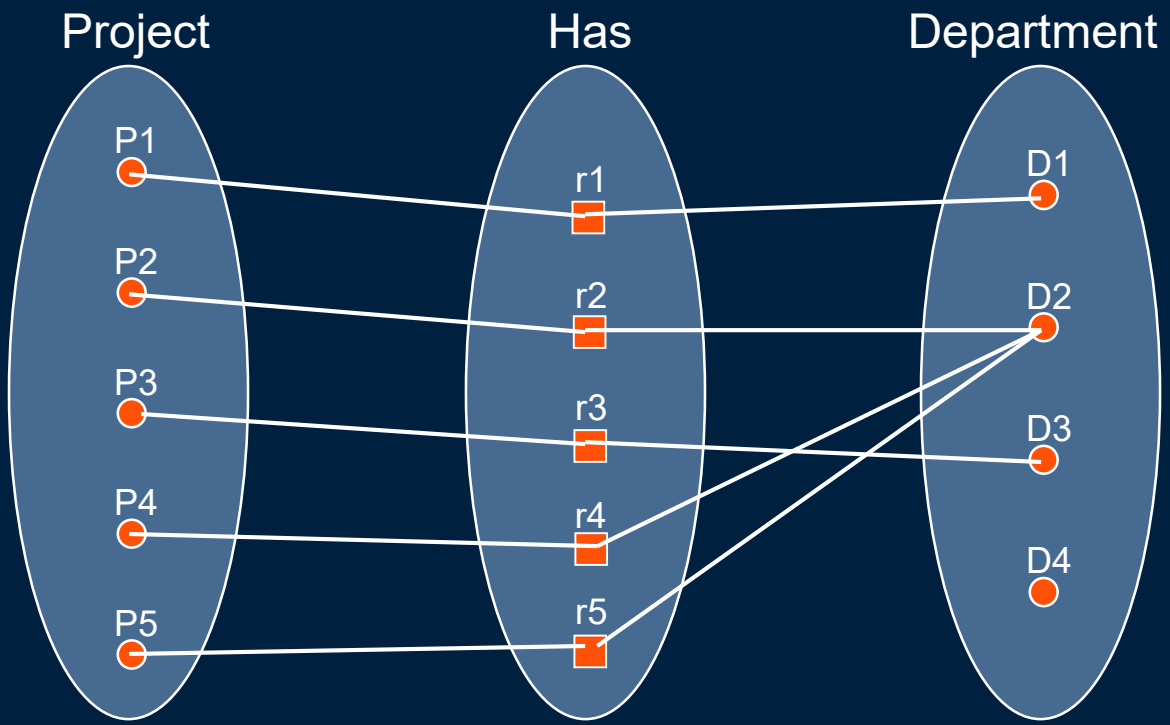
Relationship explanation: A department may have only one manager. A manager (employee) may manage only one department.

One-to-Many Relationships

In a one-to-many relationship, each instance of an entity class E1 can be associated with **more than one** instance of an entity class E2. However, E2 can only be associated with **at most one** instance of entity class E1.

Example: A department may have multiple projects, but a project may have only one department.

One-to-Many Relationship Example

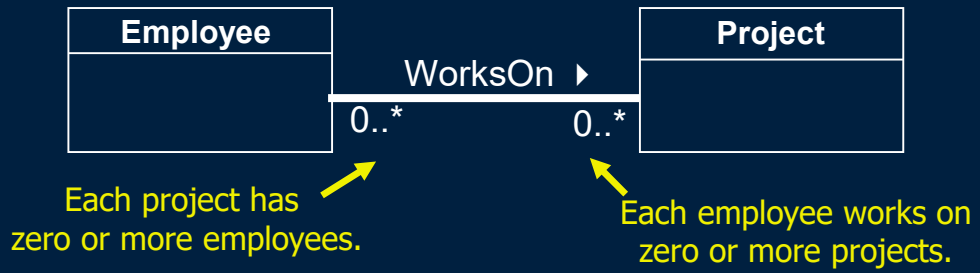


Relationship: One-to-many relationship between department and project.

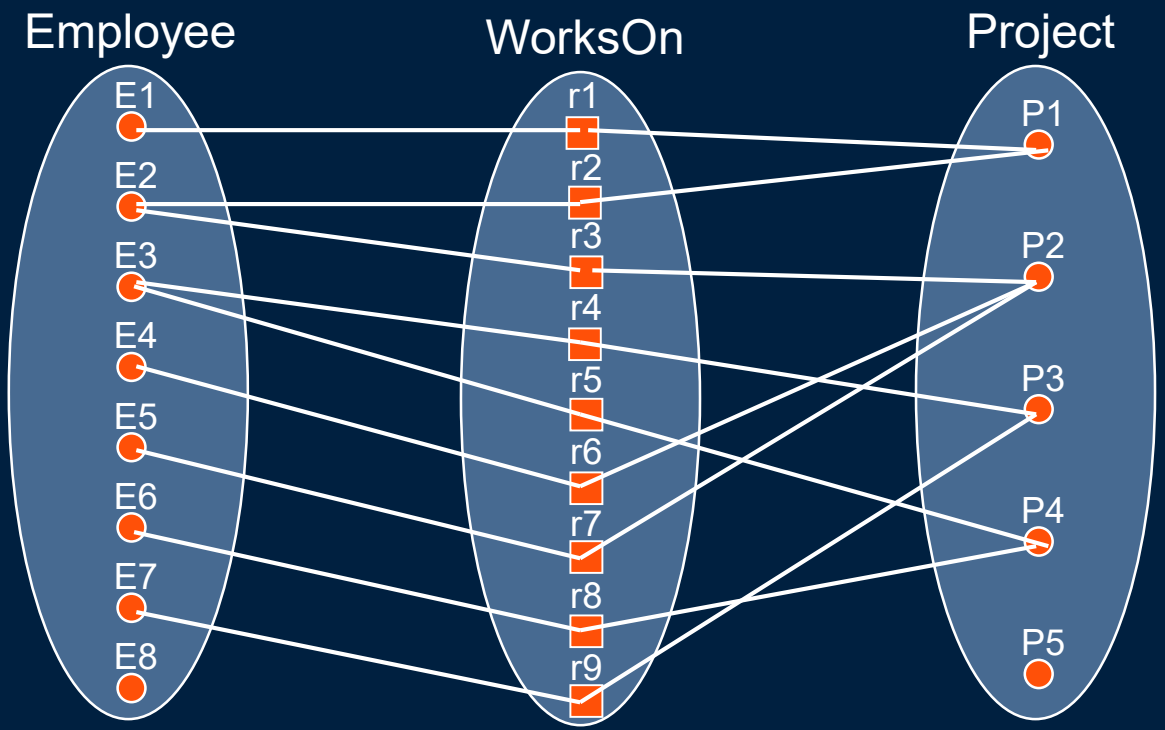
Many-to-Many Relationships

In a many-to-many relationship, each instance of an entity class E1 can be associated with **more than one** instance of an entity class E2 and vice versa.

Example: An employee may work on multiple projects, and a project may have multiple employees working on it.



Many-to-Many Relationship Example



Participation Constraints

Cardinality is the *maximum* number of relationship instances for an entity participating in a relationship type.

Participation is the *minimum* number of relationship instances for an entity participating in a relationship type.

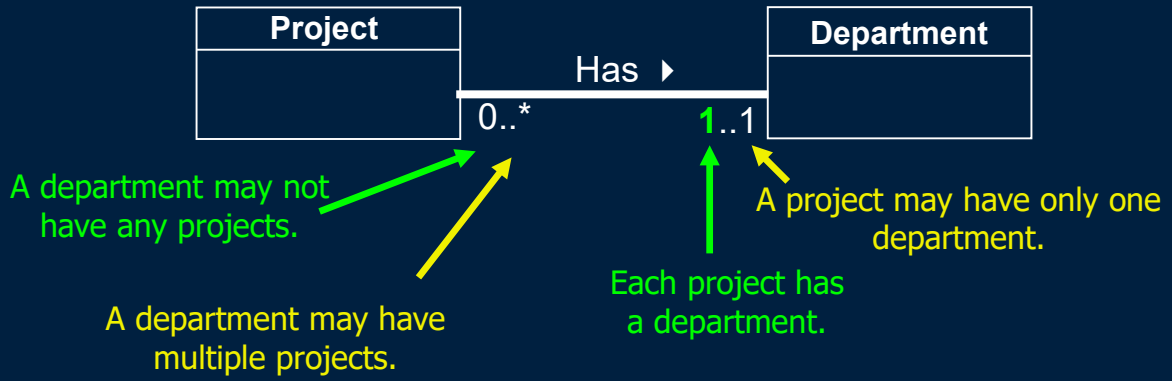
- Participation can be *optional* (zero) or *mandatory (1 or more)*.

If an entity's participation in a relationship is mandatory (also called *total* participation), then the entity's existence depends on the relationship.

- Called an *existence dependency*.

Participation Constraints Example

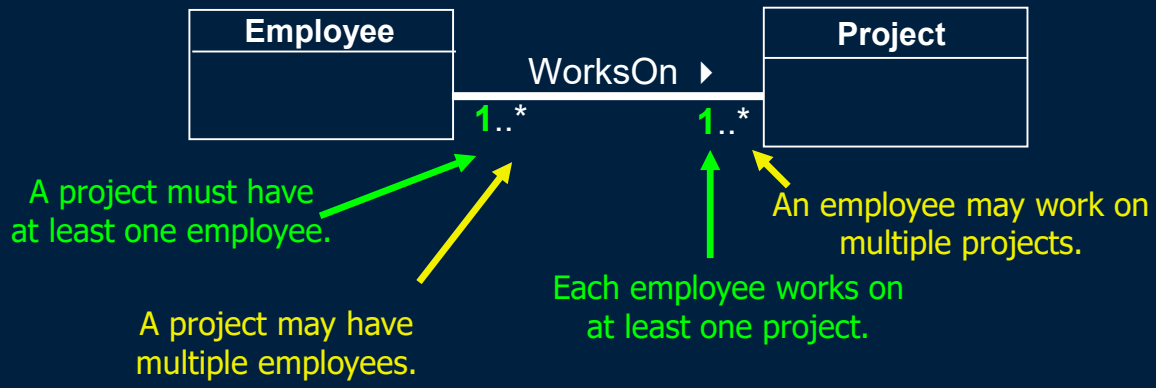
Example: A project is associated with one department, and a department may have zero or more projects.



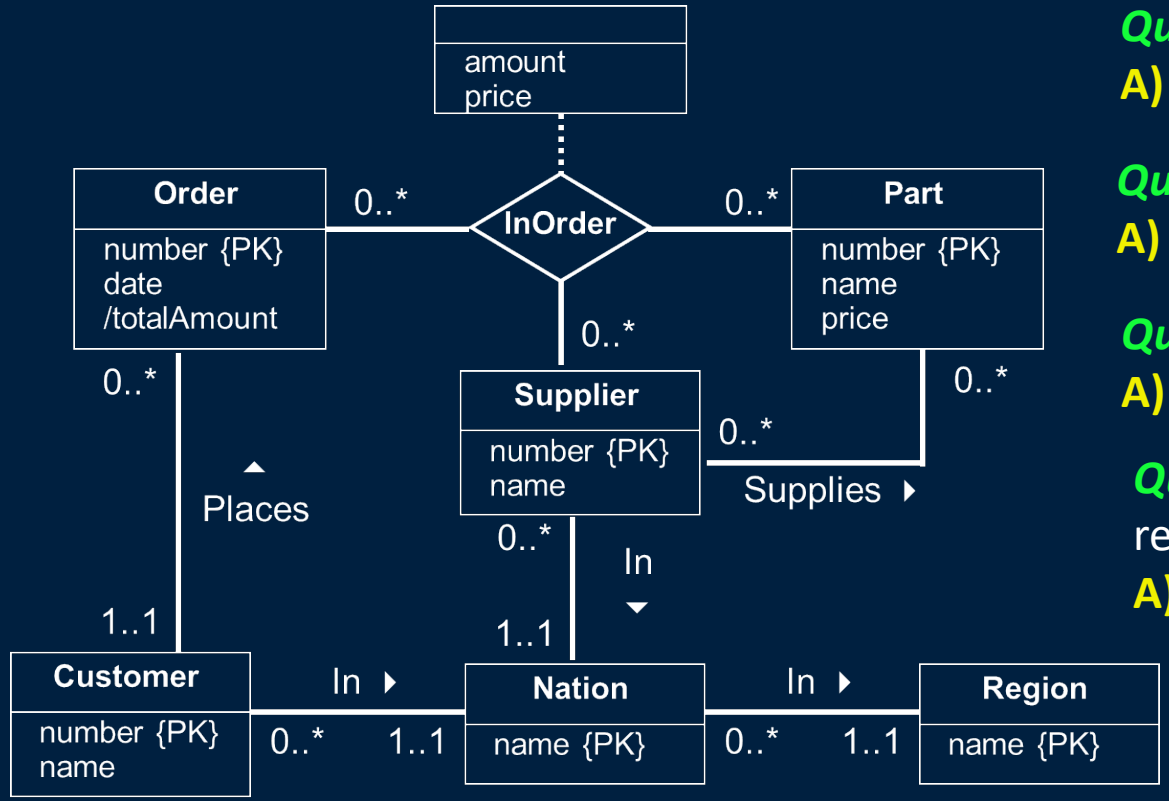
Note: Every project must participate in the relationship (mandatory).

Participation Constraints Example 2

Example: A project must have one or more employees, and an employee must work on one or more projects.



TPC-H ER Diagram Questions



Question 1: How many entity types?
A) 5 **B)** 6 **C)** 7 **D)** 8 **E)** 9

Question 2: How many relationships?
A) 4 **B)** 5 **C)** 6 **D)** 7 **E)** 8

Question 3: How many primary keys?
A) 4 **B)** 5 **C)** 6 **D)** 7 **E)** 8

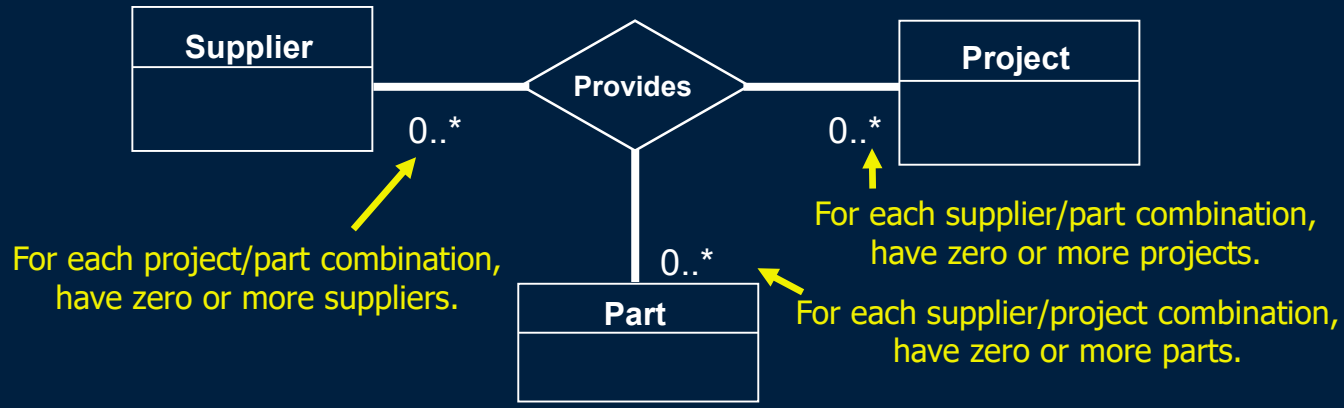
Question 4: How many 1-N relationships?
A) 4 **B)** 5 **C)** 6 **D)** 7 **E)** 8

Question 5: How many foreign keys? **A)** 3 **B)** 4 **C)** 5 **D)** 9 **E)** 0

Multiplicity of Non-Binary Relationships

The multiplicity in a complex relationship of an entity type is the number of possible occurrences of that entity-type in the *n*-ary relationship when the other (*n*-1) values are fixed.

Example: A supplier may provide zero or more parts to a project. A project may have zero or more suppliers, and each supplier may provide to the project zero or more parts.



Relationship Cardinality Question

Question: How many of the following statements are ***true***?

1) An entity of type A must be related to an entity of type B.

2) An entity of type A may be related to more than one entity B.

3) This is a 1-to-many relationship between A and B.

4) An entity of type B must be related to an entity of type A.

5) An entity of type B must be related to more than one entity of type A.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Multiplicity Practice Question

Consider the university database developed before. Write multiplicities into the ER diagram given that:

- A department must offer at least 2 courses and no more than 20 courses. Courses are offered by only one department.
- A course may have multiple sections, but always has at least one section.
- A student may enroll for courses (but does not have to).
- A professor may be in multiple departments (at least 1), and a department must have at least 3 professors.
- A section is taught by at least one professor, but may be taught by more than one. A professor does not have to teach.

Strong and Weak Entity Types

A ***strong entity type*** is an entity type whose existence is not dependent on another entity type.

- A strong entity type always has a primary key of its own attributes that uniquely identifies its instances.

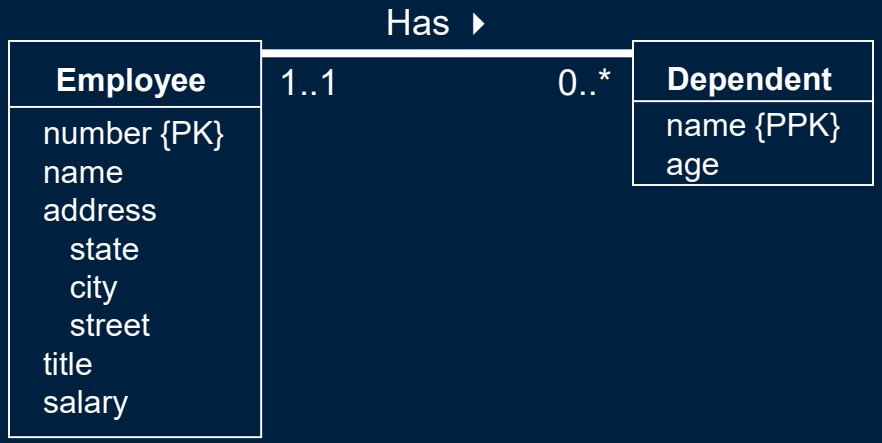
A ***weak entity type*** is an entity type whose existence is dependent on another entity type.

- A weak entity type does not have a set of its own attributes that uniquely identifies its instances.

A common example of strong and weak entity types are employees and their dependents:

- An employee is a strong entity because it has an employee number to identify its instances.
- A dependent (child) is a weak entity because the database does not store a key for each child, but rather they are identified by the parent's employee number and their name.

Weak Entities in UML



Strong and Weak Entity Question

Question: How many of the following statements are ***true***?

1) A weak entity has its own primary key.

2) A strong entity has its own primary key.

3) A weak entity must be associated (identified) by a strong entity.

4) A weak entity can have a relationship with another entity besides its identifying strong entity.

5) The attribute(s) of a weak entity used to identify it with its associated strong entity are noted as `{PPK}` in the UML model.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

ER Modeling – Entity, Relationship, or Attribute?

Basic challenge is when to model a concept as an entity, a relationship, or an attribute. In general:

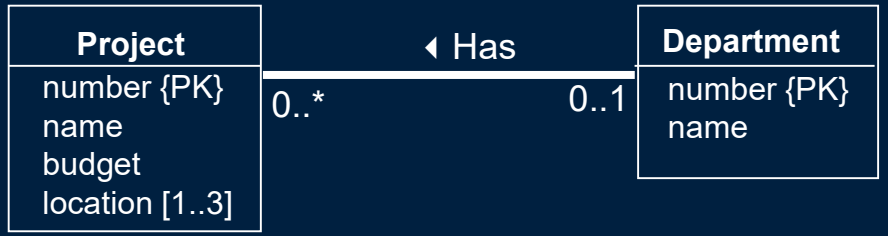
- Entities are nouns.
  - You should be able to identify a set of key attributes for an entity.
- Attributes are properties and may be nouns or adjectives.
  - Use an attribute if it relates to one entity and does not have its own key.
  - Use an entity if the concept may be shared by entities and has a key.
- Relationships should generally be binary.
  - Note that non-binary relationships can be modeled as an entity instead.

Good design will avoid redundancy and limit use of weak entities.

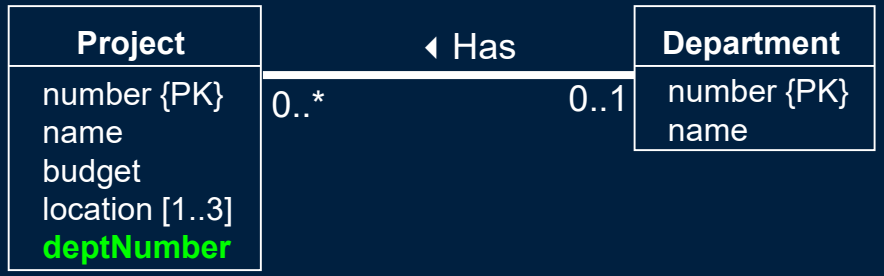
- Human-made (e.g. SIN) or auto-generated keys used instead of weak entities.

Good Design Practices
Avoiding Redundancy Example

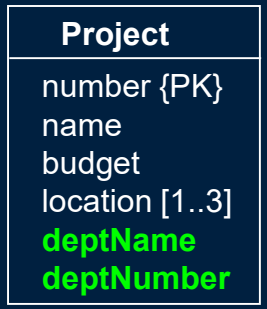
Good:



Wrong:



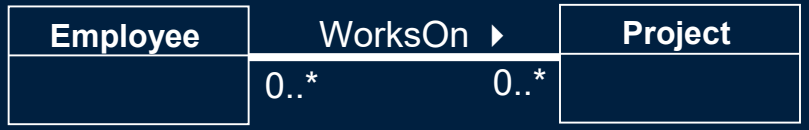
Bad:



Many-to-Many Relationship Simplification

A many-to-many relationship can be converted into one entity with two 1:N relationships between the new entity and the original entities participating in the relationship.

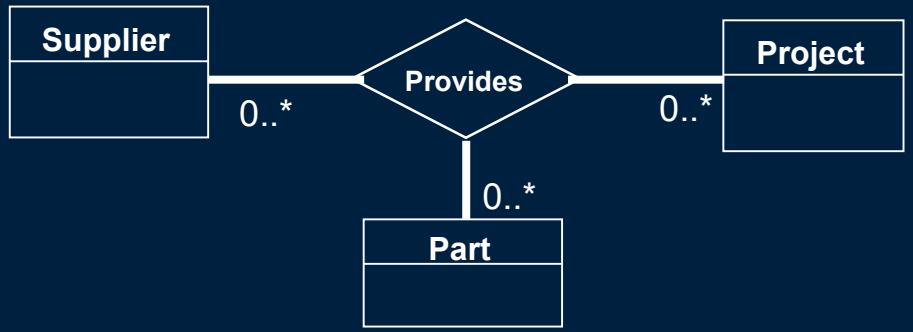
Original:



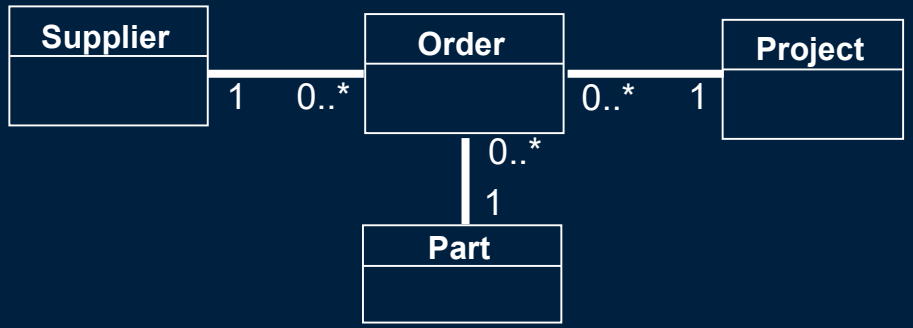
Simplified:

Higher Degree Relationships Simplified using a Weak Entity

Original:

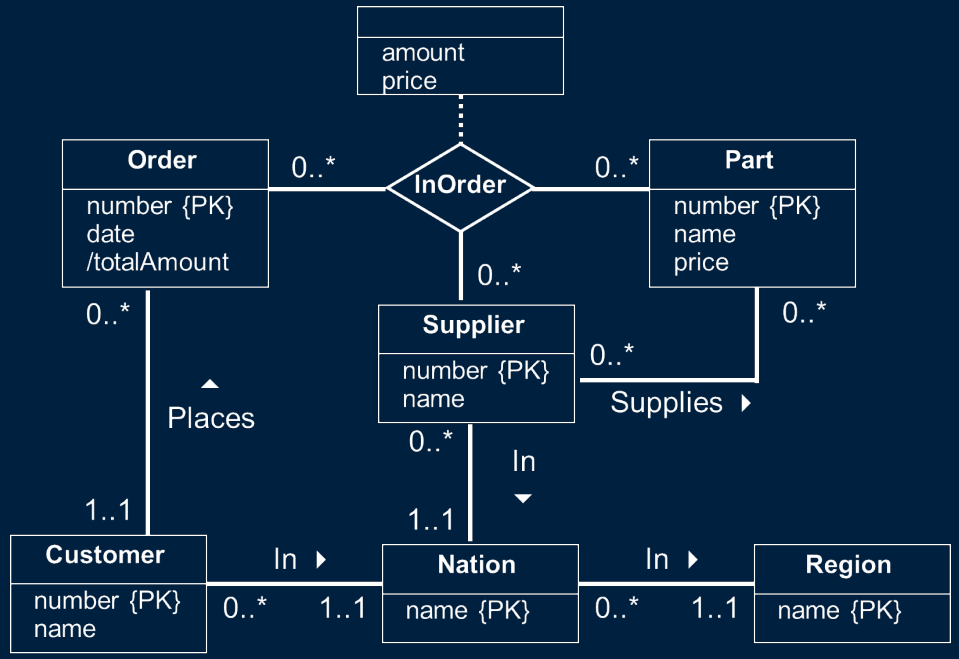


With weak entity:



ER Design Example
TPC-H Standard Schema

- Each order has a numeric key, a customer, a date, a total order amount, and a list of parts.
- A part in an order has an amount and a price paid and is supplied by a certain supplier.
- Each part has a numeric key, a name, and a price and may be supplied by multiple suppliers.
- A supplier has a key and a name and may supply multiple parts.
- A customer has a key and a name.
- Each supplier and customer is located in a nation.
- Each nation is located in a region (continent).



ER Design Question #2

Construct a fish store database where:

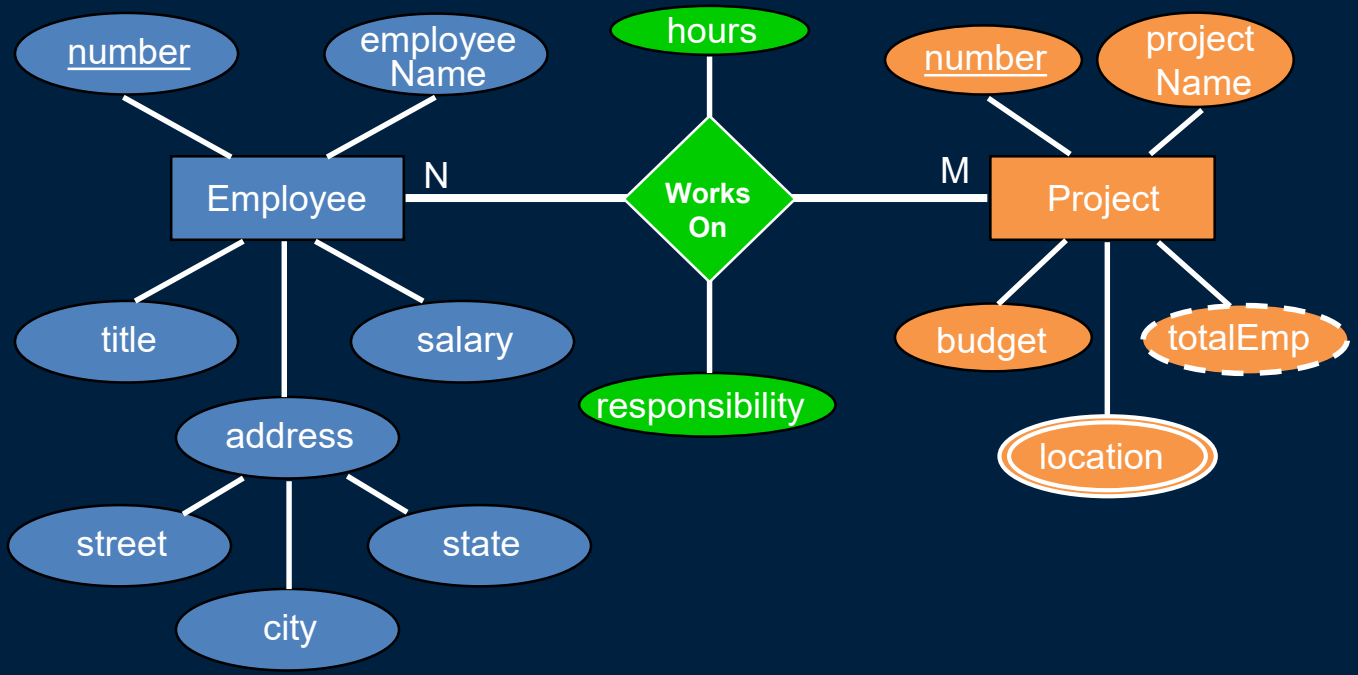
- A fish store maintains a number of aquaria (tanks), each with a number, name, volume and color.
- Each tank contains a number of fish, each with an id, name, color, and weight.
- Each fish is of a particular species, which has a id, name, and preferred food.
- Each individual fish has a number of events in its life, involving a date and a note relating to the event.

ER Design Question #3

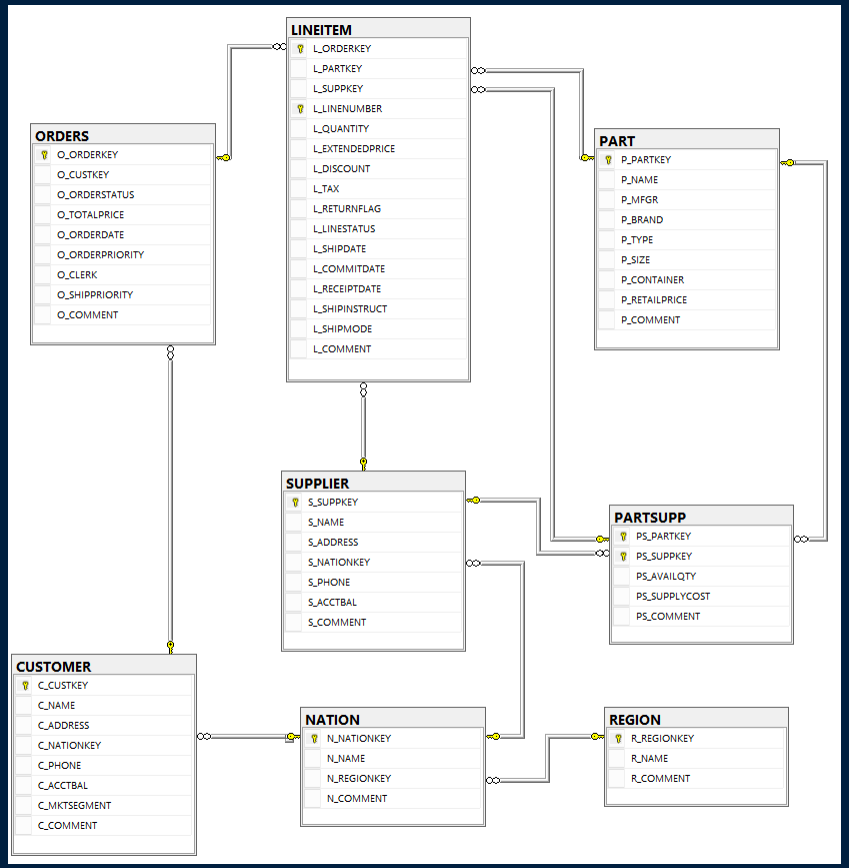
Construct an invoice database where:

- An invoice is written by a sales representative for a single customer and has a unique ID. An invoice has a date and total amount and is comprised of multiple detail lines, containing a product, price and quantity.
- Each sales representative has a name and can write many invoices, but any invoice is written by a single representative.
- Each customer has a unique id, name, and address and can request many invoices.
- Products have descriptions and weights and are supplied by vendors. Each product has a unique name for a particular vendor. A product is supplied by only one vendor.
- A vendor has an id and an address.

ER Model - Historical Notation



Other Notations – Logical/Relational Diagram



Conclusion

Conceptual design is performed at a high-level of abstraction involving entities, relationships, and attributes.

- An entity type is a group of entities with the same properties.
  - Entities may be strong (have unique key) or weak (no unique key).
- A relationship type is an association between entities.
  - A relationship may involve two or more entities and may be recursive.
  - A relationship has two types of constraints:
    - Participation - minimum # of times an entity must be involved in relationship
    - Cardinality - maximum # of times an entity can be involved in relationship
  - Common relationship multiplicities are: 1:1, 1:*, *:*.
- Attributes are properties of entities or relationships.

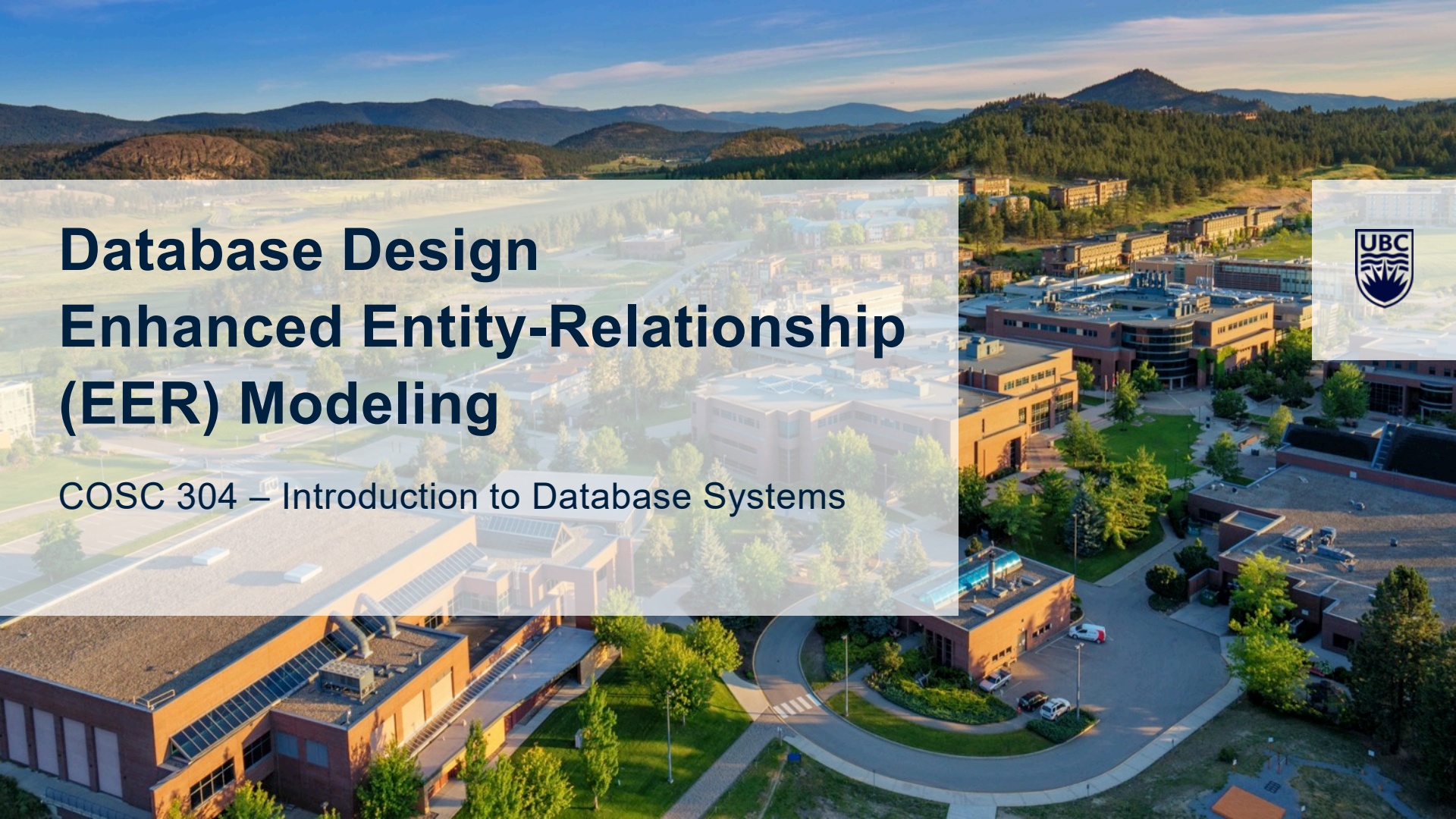
Good design requires practice.

Objectives

- Define and identify on an ER diagram: entity type, relationship type, degree of a relationship, recursive relationship, attribute, multi-valued attribute, derived attribute
- Define and identify on an ER diagram: primary key, partial primary key
- Define and identify on an ER diagram: cardinality and participation constraints
- Explain the difference between a strong entity type and a weak entity type.
- Explain multiplicity and participation and how they are used in modeling.

Model a domain explained in an English paragraph in an ER diagram using UML notation.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA



Database Design Enhanced Entity-Relationship (EER) Modeling

COSC 304 – Introduction to Database Systems

Enhanced Entity-Relationship Modeling

Enhanced Entity-Relationship (EER) modeling is an extension of ER modeling to include object-oriented concepts such as:

- superclasses and subclasses
- specialization and generalization
- aggregation and composition

These modeling constructs may allow more precise modeling of systems that are object-oriented in nature such as:

- GIS (Geographical Information Systems)
- CAD/CAM systems (Computer-Aided Design/Manufacturing)

Review: Superclasses and Subclasses

The object-oriented ideas of inheritance and superclasses and subclasses are taught during programming in an OO language such as Java.

A ***superclass*** is a general class that is extended by one or more subclasses.

A ***subclass*** is a more specific class that extends a superclass by inheriting its methods and attributes and then adding its own methods and attributes.

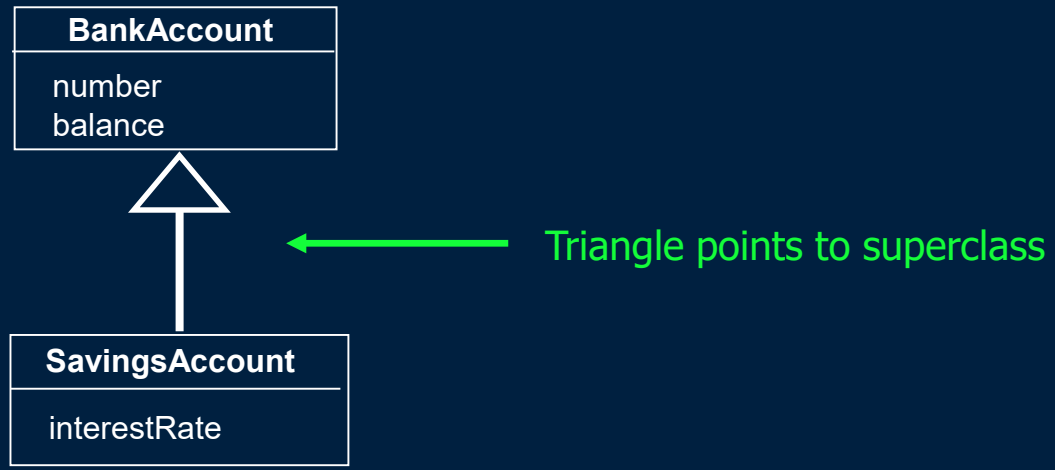
Inheritance is the process of a subclass inheriting all the methods and attributes of a superclass.

Superclasses and Subclasses Example

Java code:

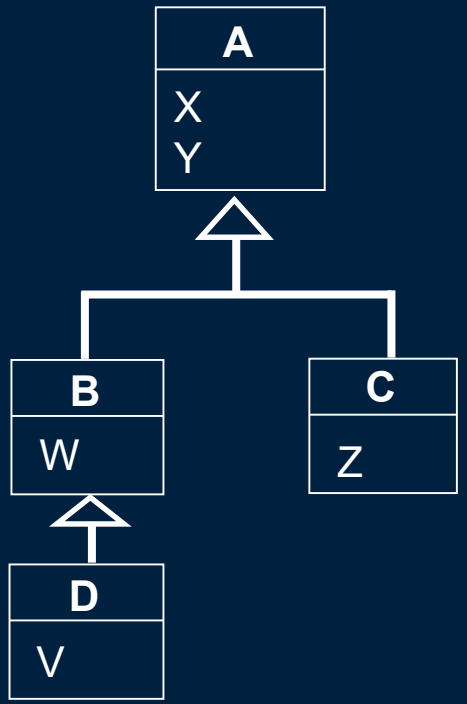
```
public class SavingsAccount extends BankAccount
```

UML class diagram:



Superclasses and Subclasses Question

Question: How many of the following statements are ***true***?



1) D is a superclass.

2) D has 1 attribute.

3) B and C are subclasses of A.

4) B inherits V from D.

5) D inherits attribute X.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

When to use EER Modeling?

Many database projects do not need the object-oriented modeling features of EER modeling.

EER modeling is useful when the domain being modeled is object-oriented in nature and inheritance reduces the complexity of the design. Common cases:

- 1) When using *attribute inheritance* can reduce the use of nulls in a single entity relation (that contains multiple subclasses).
- 2) Subclasses can be used to explicitly model and name subsets of entity types that participate in their own relationships.

When to use EER Modeling? Using Attribute Inheritance

Emp Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Note that the `title` attribute indicates what job the employee does at the company. Consider if each job title had its own unique information that we would want to record such as:

- `EE`, `PR` - programming language used (`lang`), DB used (`db`)
- `SA`, `ME` - MBA? (`MBA`), `bonus`

When to use EER Modeling? Using Attribute Inheritance (2)

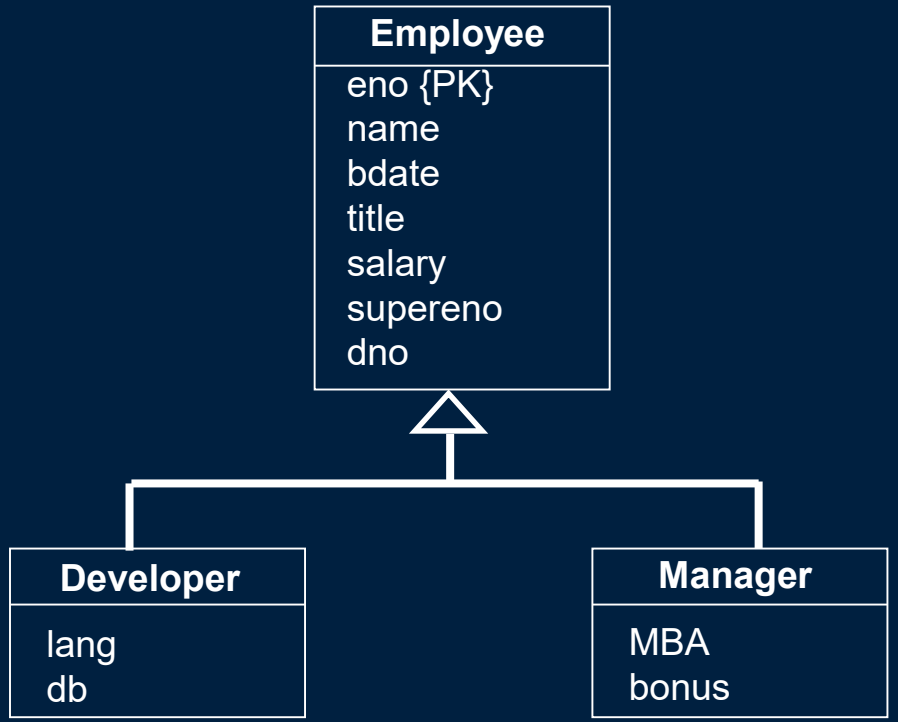
We could represent all these attributes in a single relation:

eno	ename	bdate	title	salary	supereno	dno	lang	db	MBA	bonus
E1	J. Doe	01-05-75	EE	30000	E2		C++	MySQL		
E2	M. Smith	06-04-66	SA	50000	E5	D3			N	2000
E3	A. Lee	07-05-66	ME	40000	E7	D2			N	3000
E4	J. Miller	09-01-50	PR	20000	E6	D3	Java	Oracle		
E5	B. Casey	12-25-71	SA	50000	E8	D3			Y	4000
E6	L. Chu	11-30-65	EE	30000	E7	D2	C++	DB2		
E7	R. Davis	09-08-77	ME	40000	E8	D1			N	3000
E8	J. Jones	10-11-72	SA	50000		D1			Y	6000

Note the wasted space as attributes that do not apply to a particular subclass are `NULL`.

When to use EER Modeling? Using Attribute Inheritance (3)

A better solution would be to make two subclasses of `Employee` called `Developer` and `Manager`:



When to use EER Modeling? Using Attribute Inheritance (4)

Resulting relations:

Employee Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Developer Relation

eno	lang	db
E1	C++	MySQL
E4	Java	Oracle
E6	C++	DB2

Manager Relation

eno	MBA	bonus
E2	N	2000
E3	N	3000
E5	Y	4000
E7	N	3000
E8	Y	6000

Generalization and Specialization

Subclasses and superclasses are created by using either generalization or specialization.

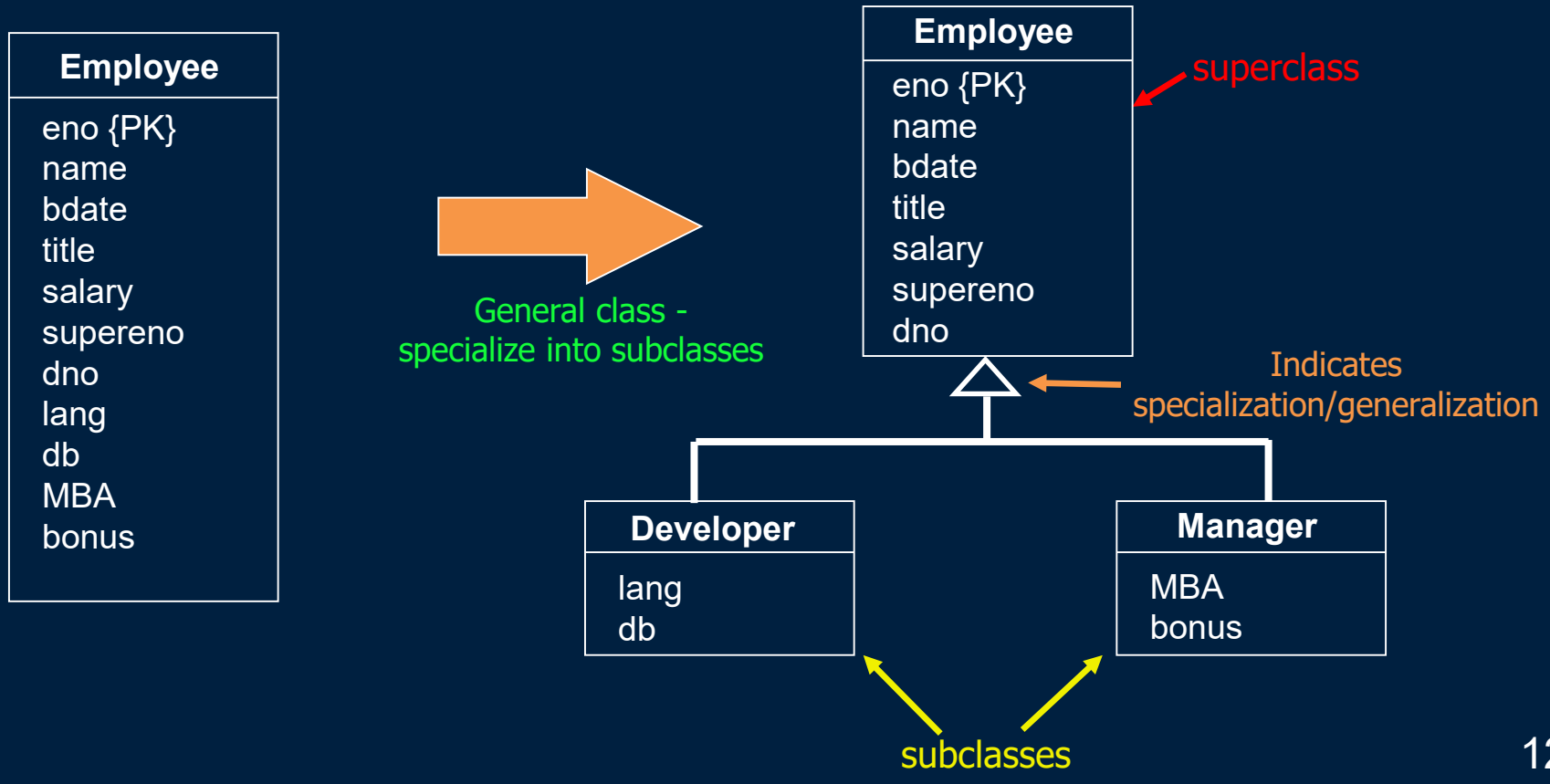
Specialization is the process of creating more specialized subclasses of an existing superclass.

- Top-down process: Start with a general class and then subdivide it into more specialized classes.
  - The specialized classes may contain their own attributes. Attributes common to all subclasses remain in the superclass.

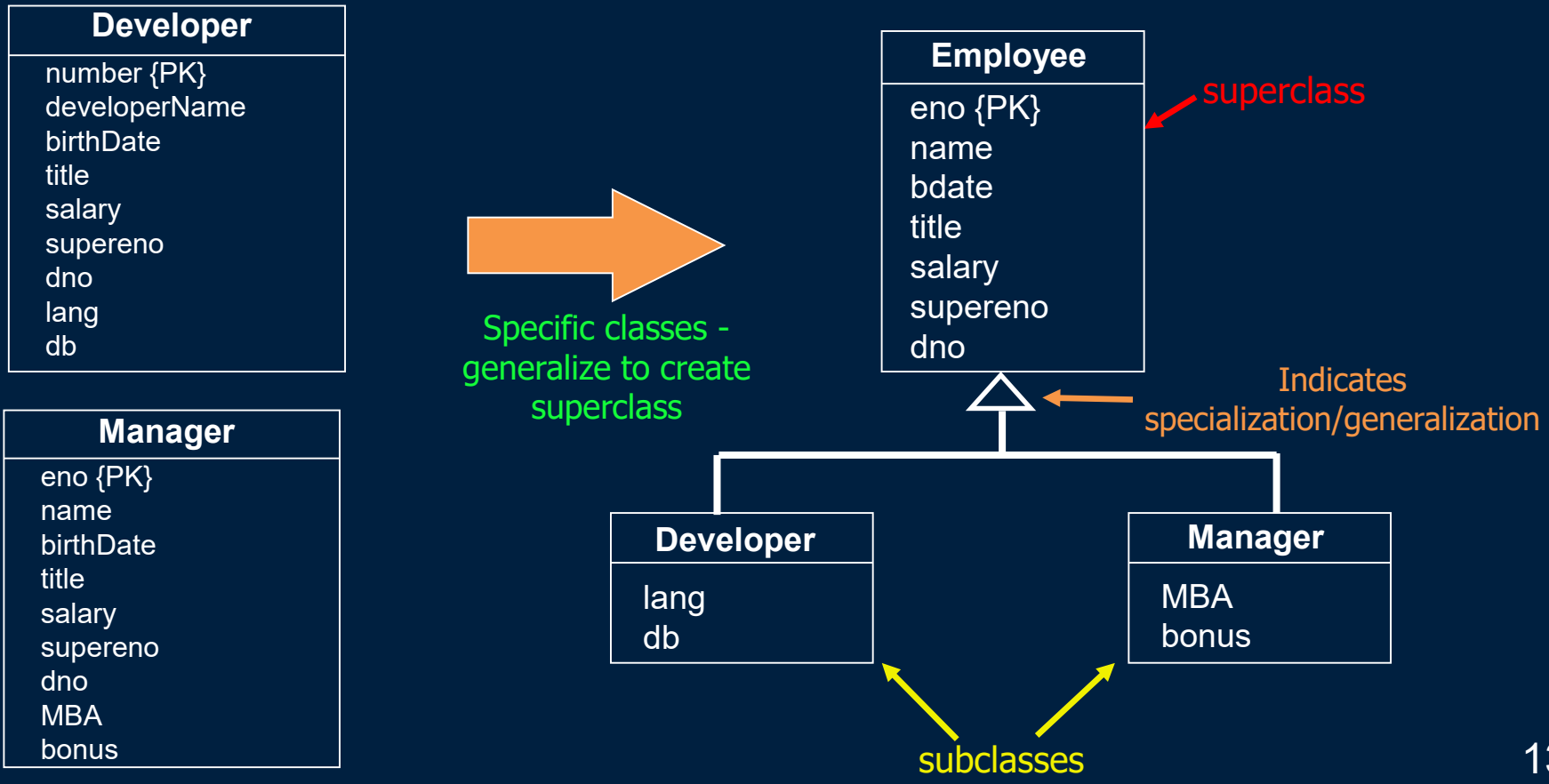
Generalization is the process of creating a more general superclass from existing subclasses.

- Bottom-up process: Start with specialized classes and try to determine a general class that contains the attributes common to all of them.

Specialization Example



Generalization Example



Constraints on Generalization and Specialization

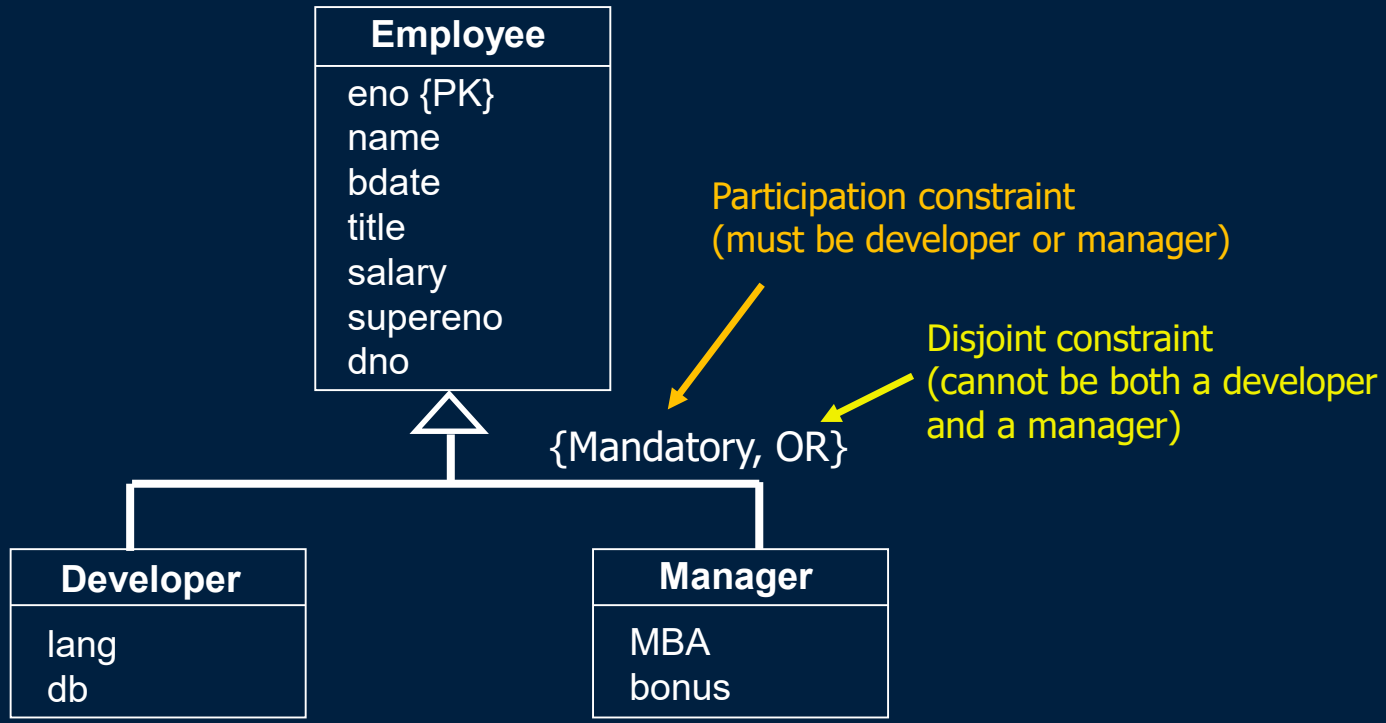
There are two types of constraints associated with generalization and specialization:

- ***Participation constraint*** - determines if every member in a superclass must participate as a member of one of its subclasses.
  - It may be `optional` for a superclass member to be a member of one of its subclasses, or it may be `mandatory` that a superclass member be a member of one of its subclasses.

- ***Disjoint constraint*** - determines if a member of a superclass can be a member of one or more than one of its subclasses.
  - If a superclass object may be a member of only one of its subclasses this is denoted by `OR` (subclasses are *disjoint*).
  - Otherwise, `AND` is used to indicate that it may be in more than one of its subclasses.

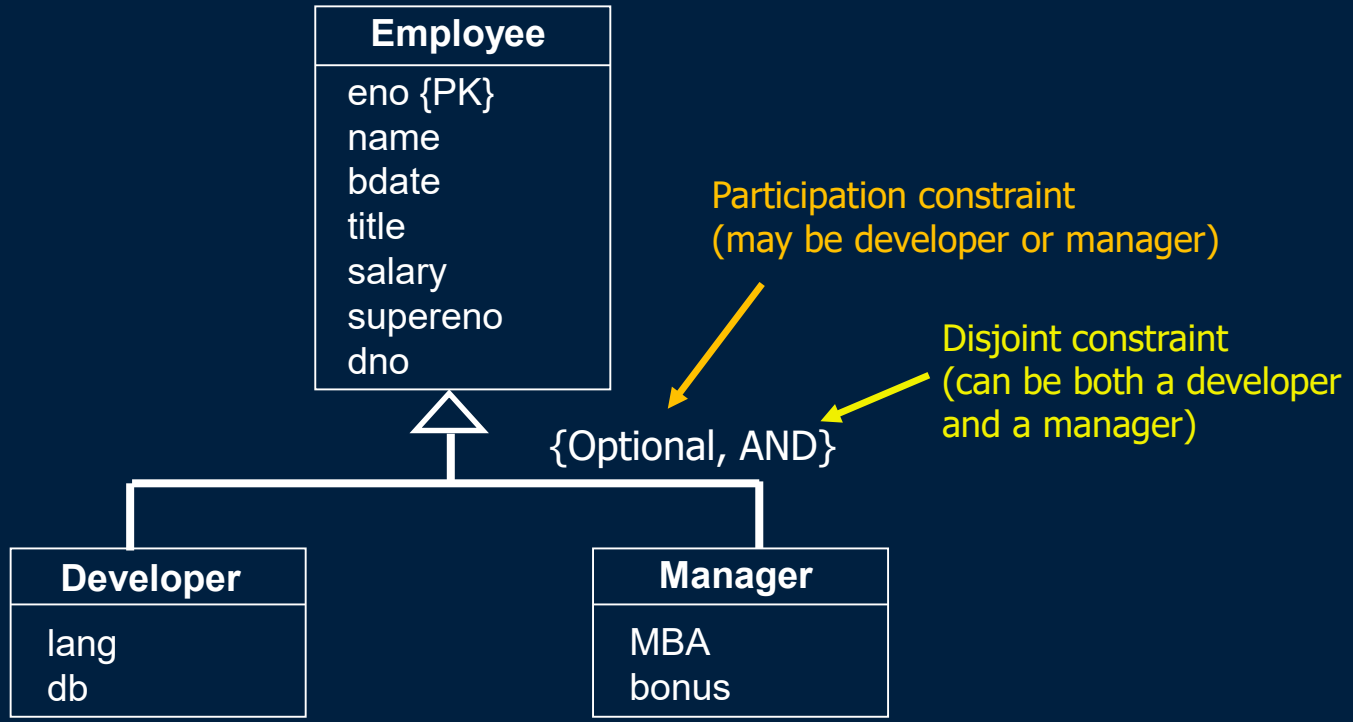
Constraints Example

An employee must be either a developer or a manager, but cannot be both.



Constraints Example (2)

An employee may specialize as a developer or manager. An employee may be both a manager and developer.



General Predicate Constraints

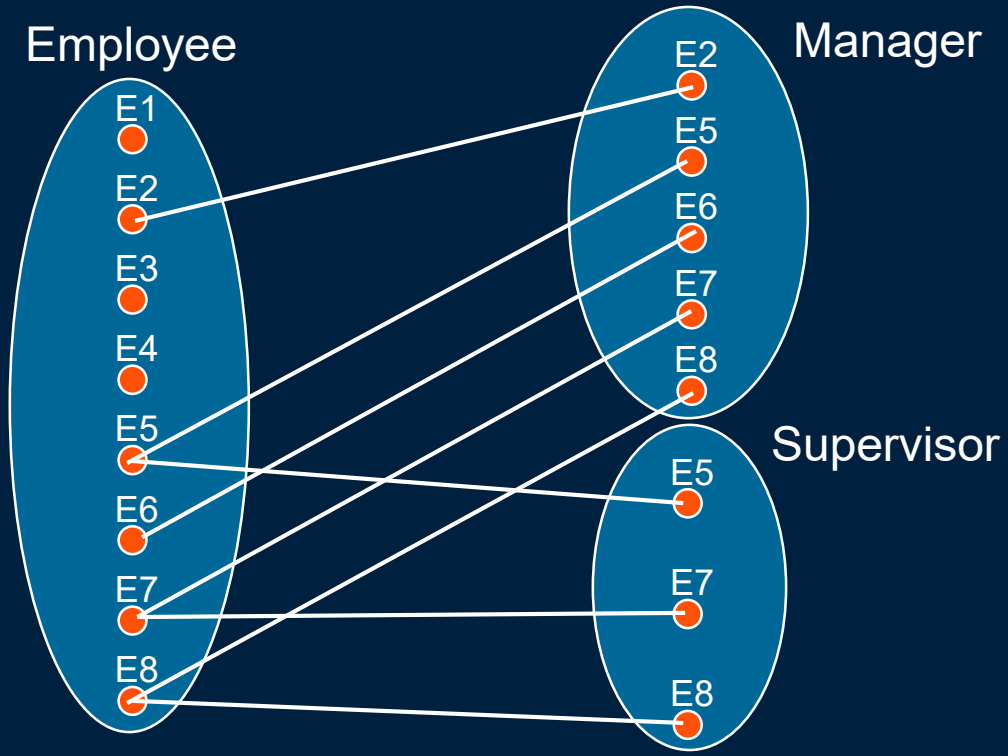
Predicate-defined constraints specify when an object participates in a subclass using a certain rule.

- For example, a subclass called `RichEmployees` can be defined with a membership predicate such as `salary >100000`.

Attribute-defined subclasses are a particular type of predicate-defined constraint where the value of an attribute(s) determines if an object is a member of a subclass.

- For example, the `title` field could be used as a *defining attribute* for the `Developer` and `Manager` subclasses.
  - `Emp` is in `Developer` if `title = 'EE' or 'PR'`
  - `Emp` is in `Manager` if `title = 'ME' or 'SA'`

Constraints Question



Note: What is the participation and the disjoint constraints for superclass `Employee` (with subclasses `Manager` and `Supervisor`) given these instances?

Relationship Constraints vs. Inheritance Constraints

There is a parallel between relationship constraints on relationships and inheritance constraints on superclasses and subclasses.

- Minimum # of occurrences – called participation constraint in both cases
- Maximum # of occurrences – called cardinality constraint for relationships and disjoint constraint for subclasses

Possible combinations:

Subclass Constraints	Relationship Constraints
Optional, AND	0..*
Optional, OR	0..1
Mandatory, AND	1..*
Mandatory, OR	1..1

EER Question

Question: How many of the following statements are ***true***?

1) Generalization is a bottom-up process.

2) In an UML diagram, the inheritance arrow points towards the superclass.

3) OPTIONAL and MANDATORY are possible choices for the participation constraint.

4) If the disjoint constraint is AND, a given object can be a member of multiple subclasses.

5) If the participation constraint is OPTIONAL, the disjoint constraint must be AND.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Multiple Inheritance

If each class only has one superclass, then the class diagram is said to be a ***specialization*** or ***type hierarchy***.

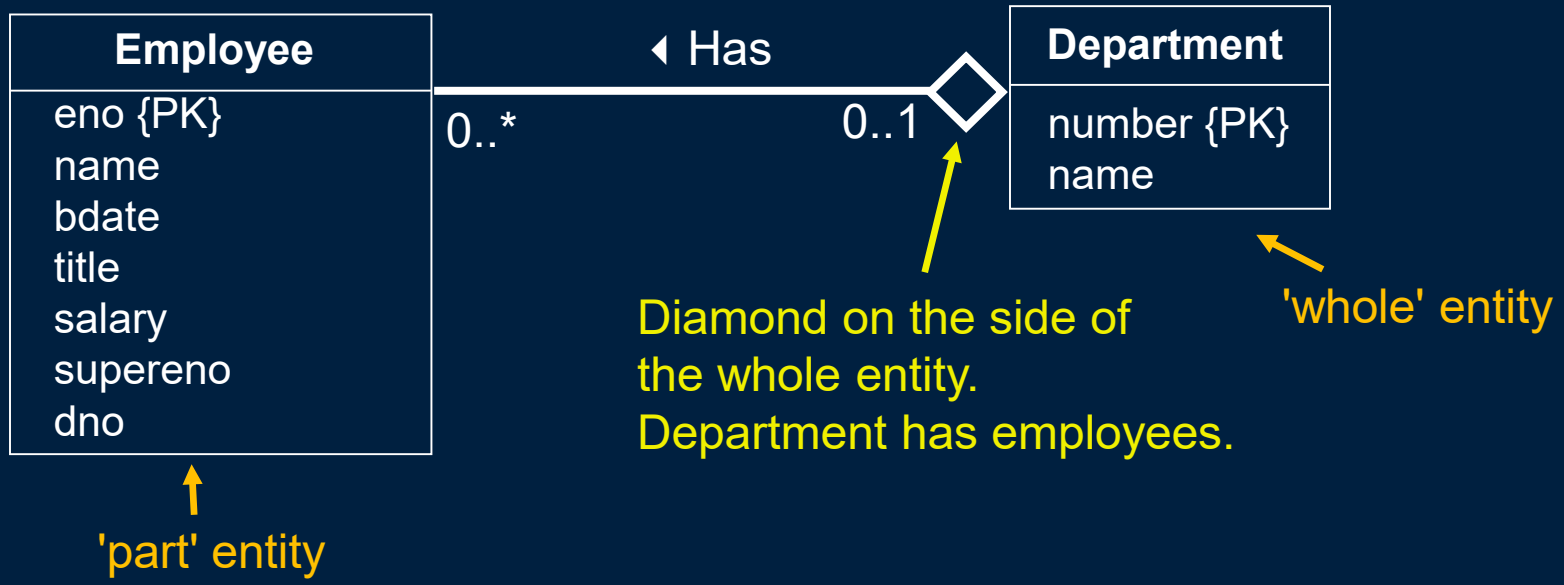
If a class may have more than one superclass, then the class diagram is said to be a ***specialization*** or ***type lattice***.

Although multiple inheritance is powerful, it should be avoided if possible.

Aggregation

Aggregation represents a `'HAS-A'` or `'IS-PART-OF'` relationship between entity types. One entity type is the *whole*, the other is the *part*.

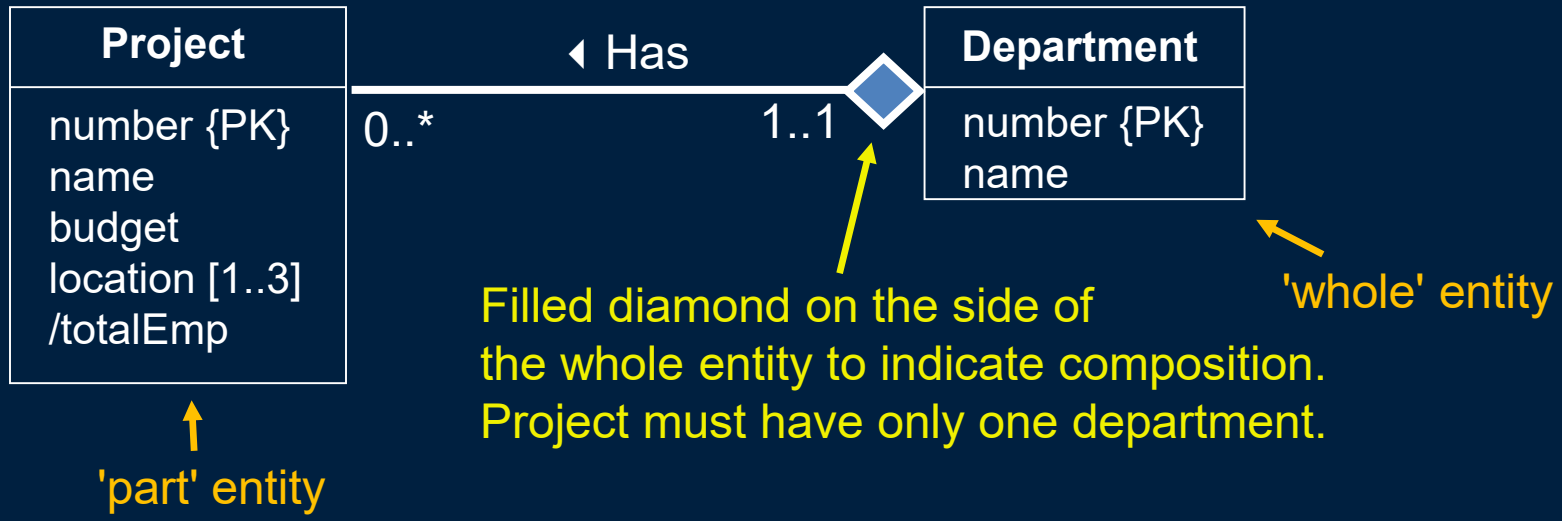
Example:



Composition

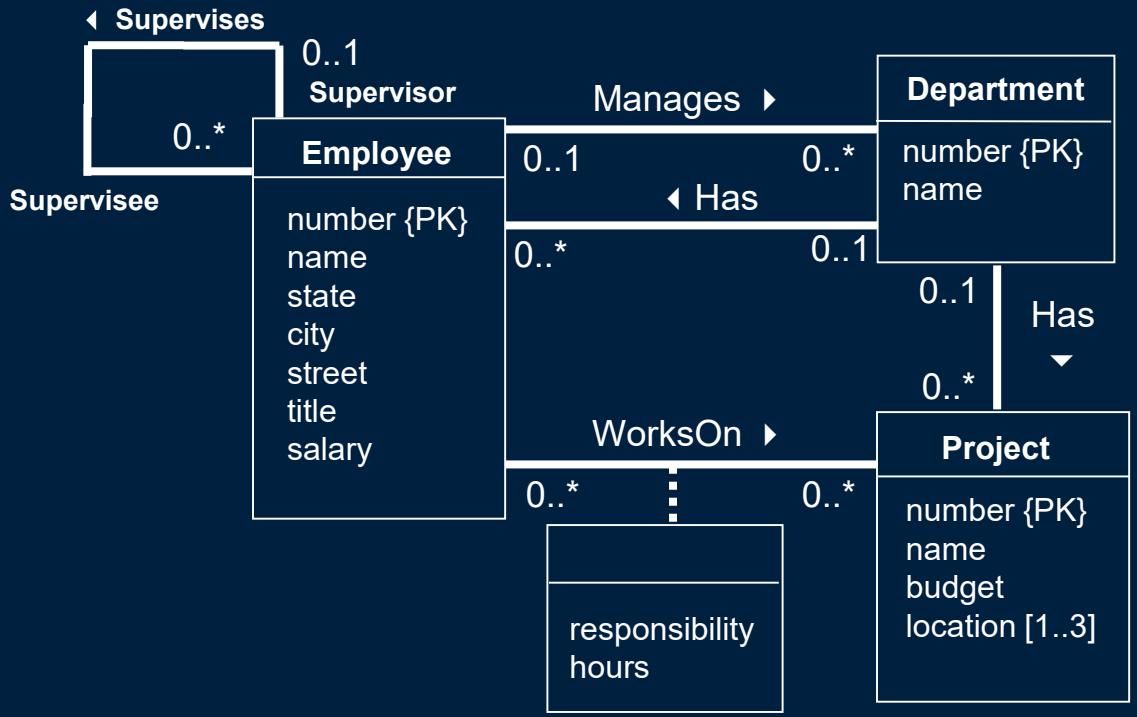
Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Example:

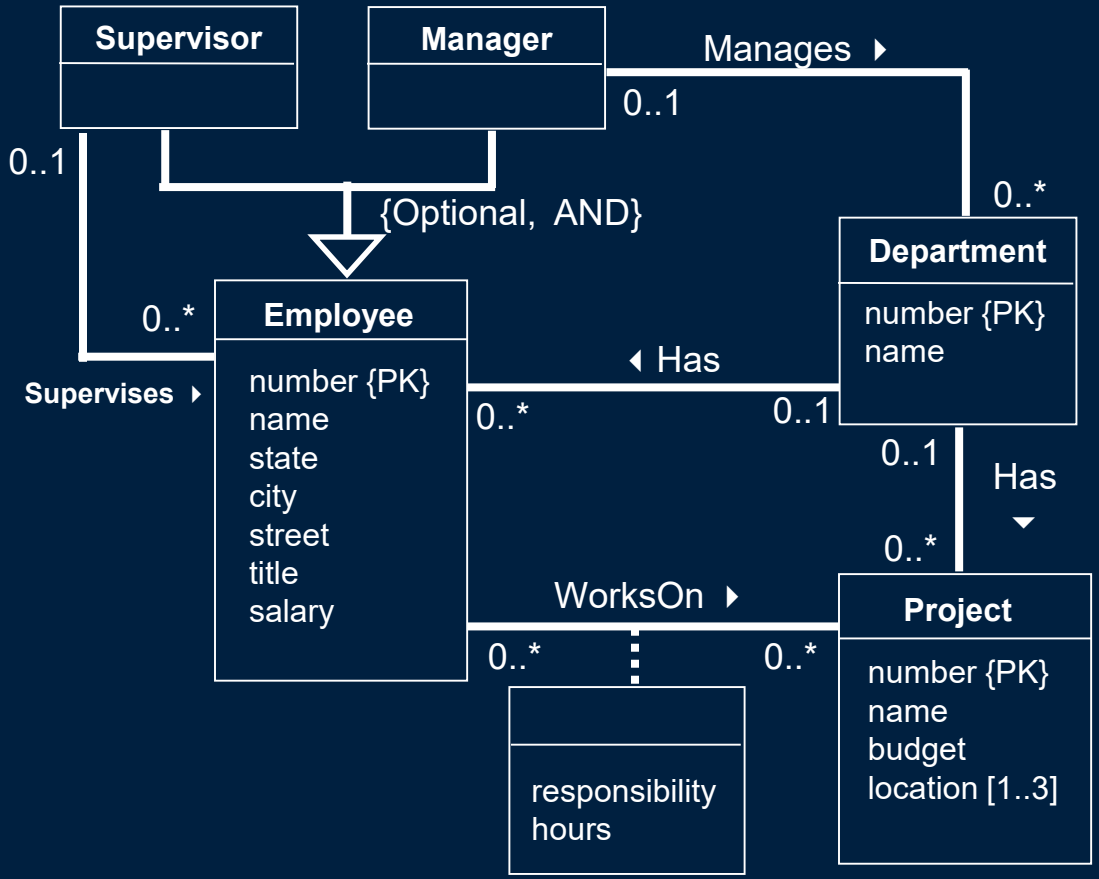


Note: The min-max constraint on the whole side of the relationship (in this case department) must always be 1..1 when modeling composition. Why?

Original ER Model Example



EER Model Example



Conclusion

The ***Enhanced Entity-Relationship*** model (***EER***) model allows for object-oriented design features to be captured.

Generalization and ***specialization*** are two complementary processes for constructing superclasses and subclasses.

Participation and ***disjoint constraints*** apply to subclasses.

- Participation of a superclass may be `mandatory` or `optional` in a subclass.
- A superclass may only be a member of one subclass (disjoint constraint indicated by `OR`) or multiple (indicated by `AND`).

Aggregation and composition are used to model `HAS-A` or `PART-OF` relationships.

The features of EER modeling are rarely needed in most database design projects.

Objectives

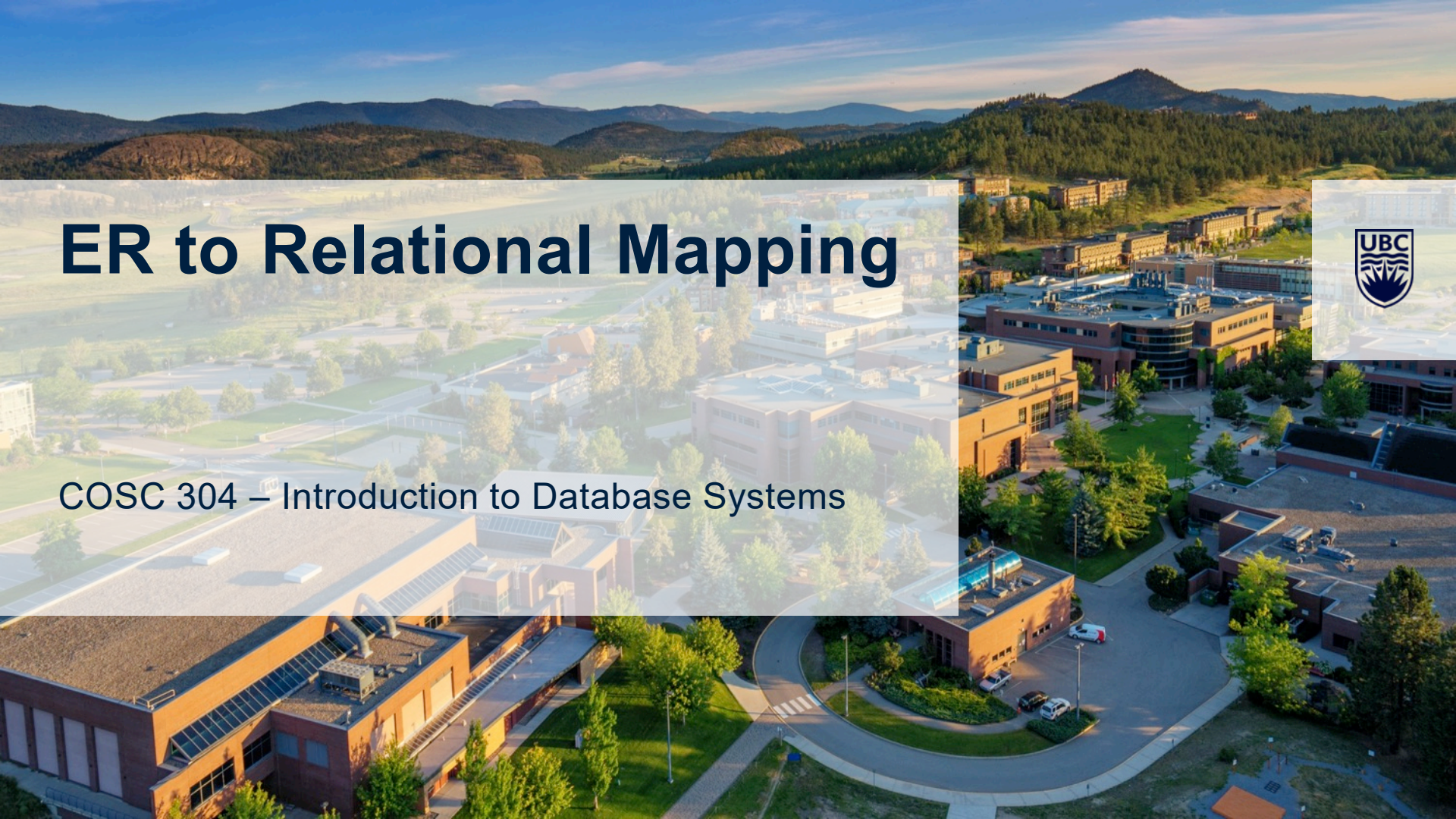
Given an EER diagram, recognize the subclasses, superclasses, and constraints using the notation.

Explain the difference between the participation constraint and the disjoint constraint.

Explain the difference between aggregation and composition.

Given an EER diagram, list the attributes of each class including attributes inherited from superclasses.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA



ER to Relational Mapping

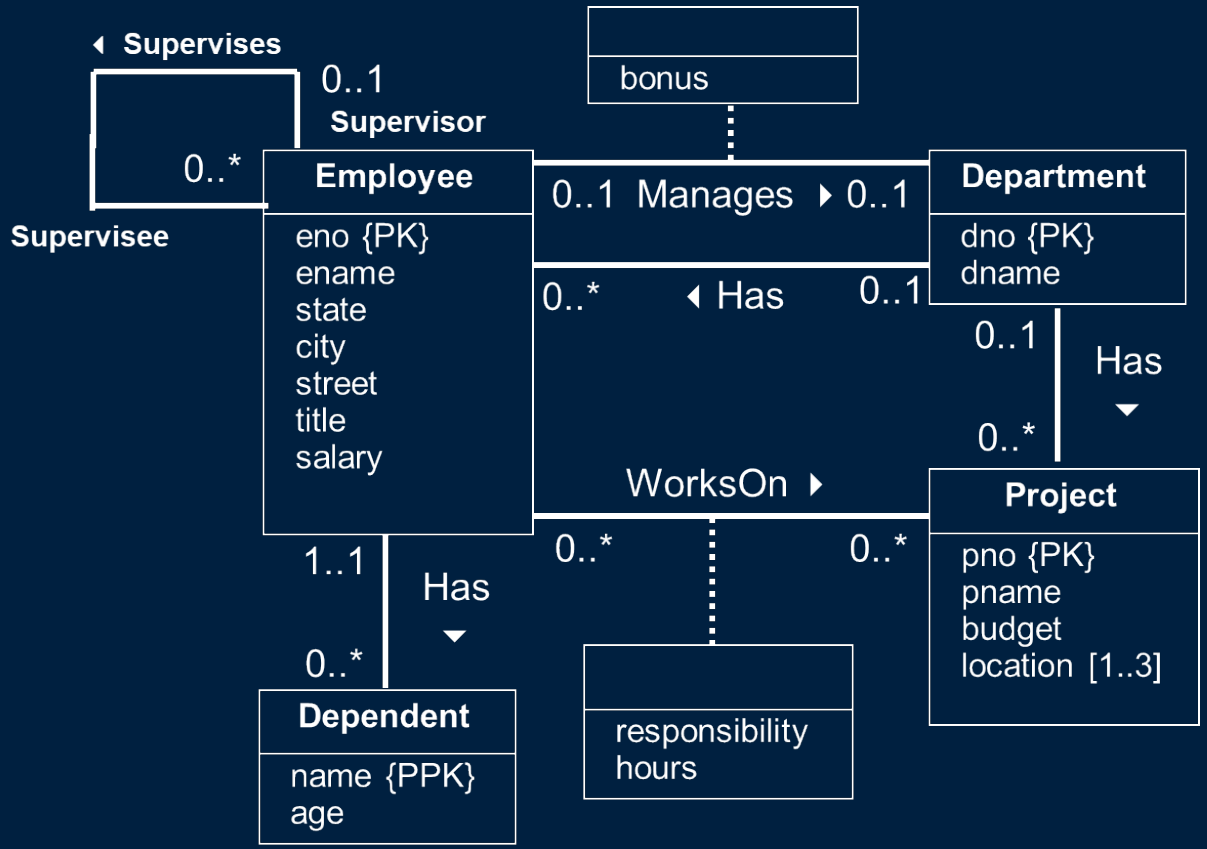
COSC 304 – Introduction to Database Systems

ER Model to Relational Schemas

Converting an ER model to a relational database schema involves 8 steps.

In general, these steps convert entities to relations and ER relationships to relations. For 1:1 and 1:N relationships, foreign keys are used instead of separate relations.

ER to Relational Example



ER to Relational Mapping
Step #1: Convert Strong Entities

Step #1: Convert each strong entity to a relation.

→ Employee (eno, ename, state, city, street, title, salary)

- Notes:
  - Attributes of the entity type become attributes of the relation.
  - Multi-valued attributes are handled separately (in step #6).
  - The primary key of the relation is the key attributes for the entity.

ER to Relational Mapping
Current Relational Schema - Step #1

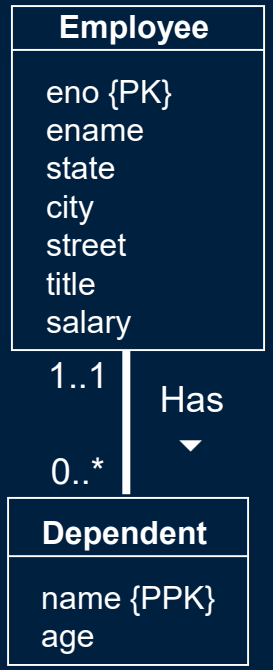
Employee (eno, ename, state, city, street, title, salary)

Project (pno, pname, budget)

Department (dno, dname)

ER to Relational Mapping
Step #2: Convert Weak Entities

Step #2: Convert each weak entity into a relation with foreign keys to its identifying relations (entities). Example:



Employee (eno, ename, state, city, street, title, salary)

Dependent (eno, name, age)

Dependent.eno is a FK to Employee.eno

ER to Relational Mapping
Current Relational Schema - Step #2

Dependent (eno, name, age)

Employee (eno, ename, state, city, street, title, salary)

Project (pno, pname, budget)

Department (dno, dname)

ER to Relational Mapping
Steps #3-5: Convert Relationships

Steps 3 to 5 convert *binary* relationships of cardinality:

- 1:1 - Step #3
- 1:N - Step #4
- M:N - Step #5

M:N relationships are the most general case, and the conversion algorithm for these relationships can be applied to 1:1 and 1:N as well.

- However, for performance reasons, it is more efficient to perform different conversions for each relationship type.
- In general, each ER relationship can be mapped to a relation. However, for 1:1 and 1:N relationships, it is more efficient to combine the relationship with an existing relation instead of creating a new one.

Relationships that are not binary are handled in step #7.

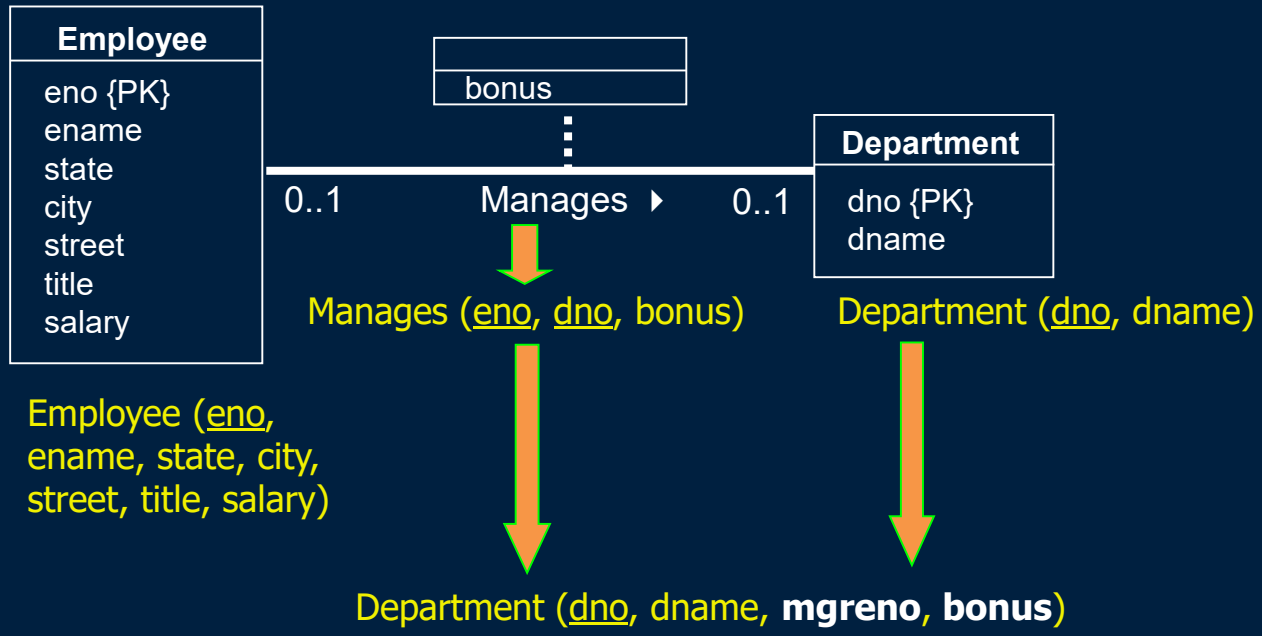
ER to Relational Mapping
Step #3: Convert 1:1 Relationships

Step #3: Convert binary 1:1 relationships into a `UNIQUE` foreign key reference from one relation to the other.

Chose one of the relations, say *R*, and:

- Add the attributes of the relationship to *R*.
- Add the primary key attributes of the other relation to *R*, and create a foreign key reference to the other relation.
- Declare these added primary key attributes of *R* to be `UNIQUE`.
- Note: You can select either relation, but it is best to select the relation that is guaranteed to always participate in the relationship or the one that will participate the most in the relationship.

ER to Relational Mapping
Step #3: Convert 1:1 Relationships (2)



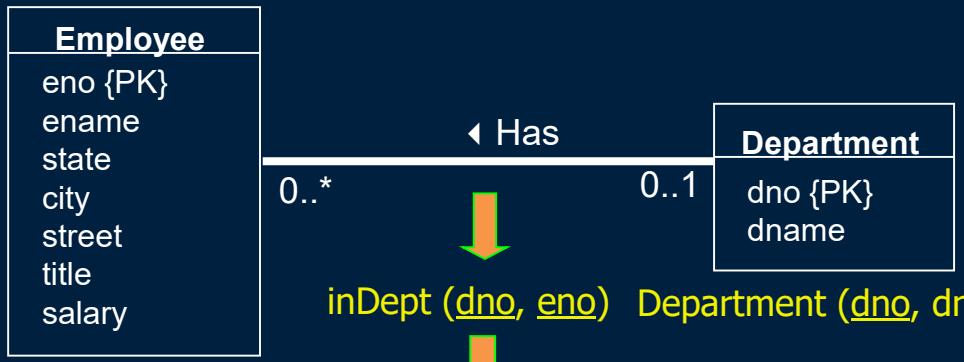
Note: Renamed eno to mgreno for clarity.

ER to Relational Mapping
Step #4: Convert 1:N Relationships

Step #4: Convert binary 1:N relationships into a foreign key reference from the N-side relation to the 1-side relation.

Note: Unlike 1:1 relationships, you must select the N-side of the relationship as the relation containing the foreign key and relationship attributes.

ER to Relational Mapping
Step #4: Convert 1:N Relationships



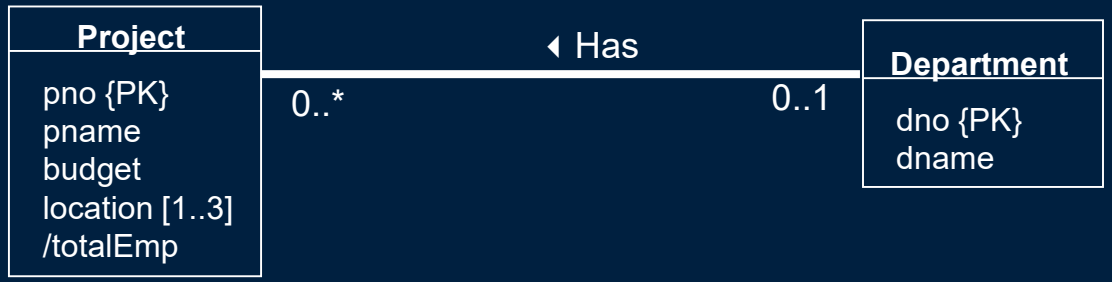
inDept (<u>dno</u>, <u>eno</u>)

Department (<u>dno</u>, dname, mgreno, bonus)

Employee (<u>eno</u>, ename, state, city, street, title, salary)

Employee (<u>eno</u>, ename, state, city, street, title, salary, **dno**)

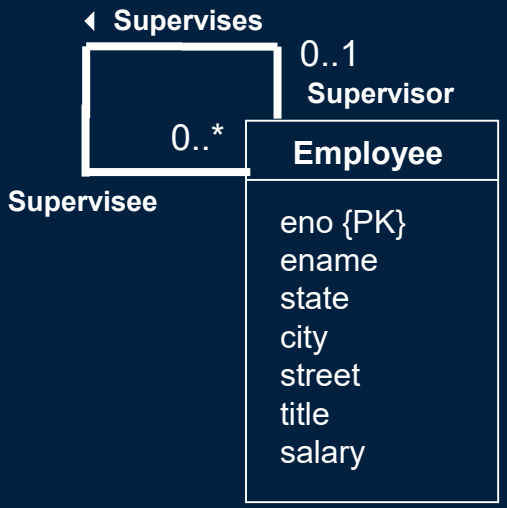
ER to Relational Mapping
Step #4: Convert 1:N Relationships



Question: What table has the foreign key field?

A) Project **B)** Department **C)** None

ER to Relational Mapping
Step #4: Convert 1:N Relationships

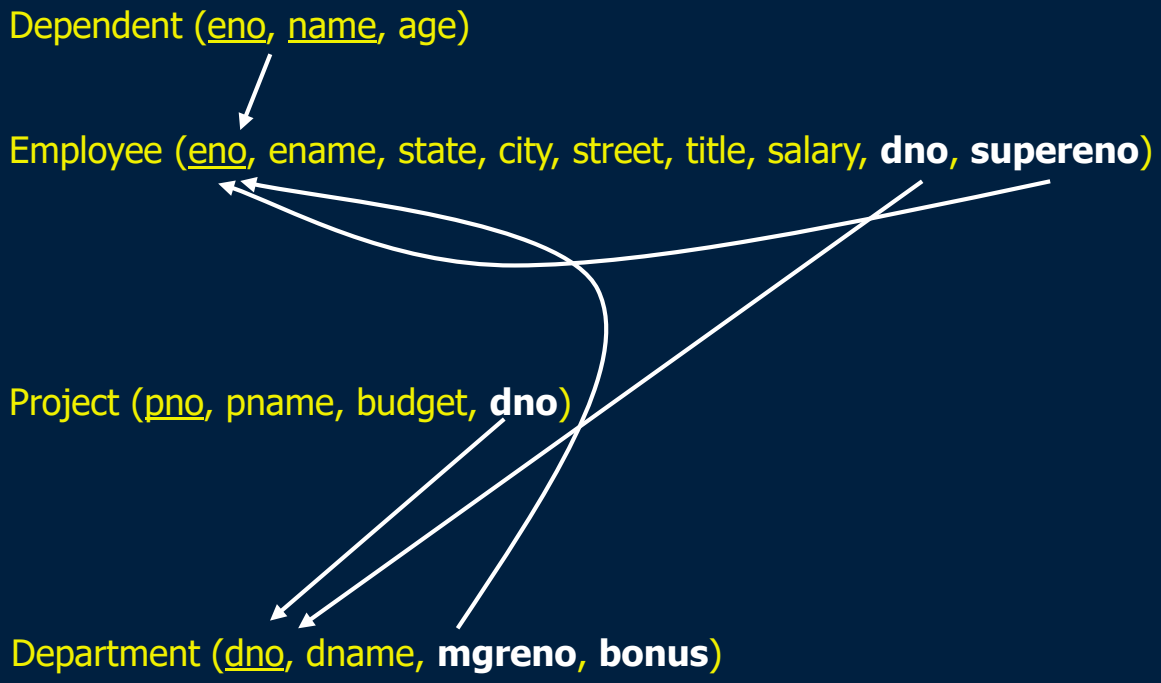


Supervises (<u>supereno</u>, <u>eno</u>)

Employee (<u>eno</u>, ename, state, city, street, title, salary, dno)

Employee (<u>eno</u>, ename, state, city, street, title, salary, dno, **supereno**)

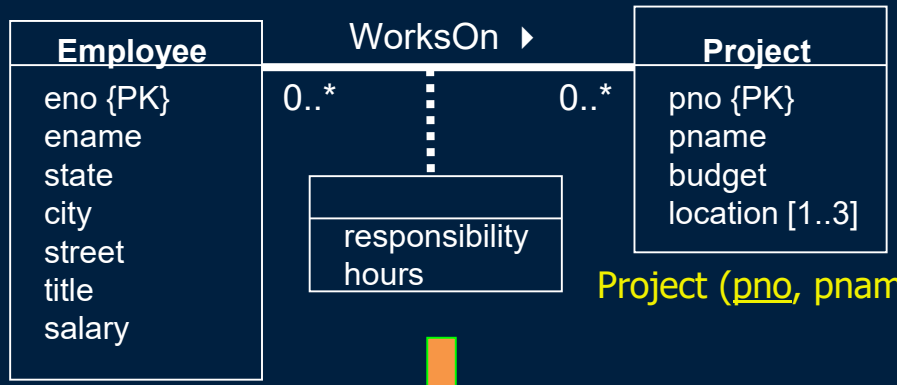
ER to Relational Mapping
Current Relational Schema - Step #4



ER to Relational Mapping
Step #5: Convert M:N Relationships

Step #5: Convert binary M:N relationships into a new relation with foreign keys to the two participating entities.

- The primary key consists of the primary keys of the two relations.

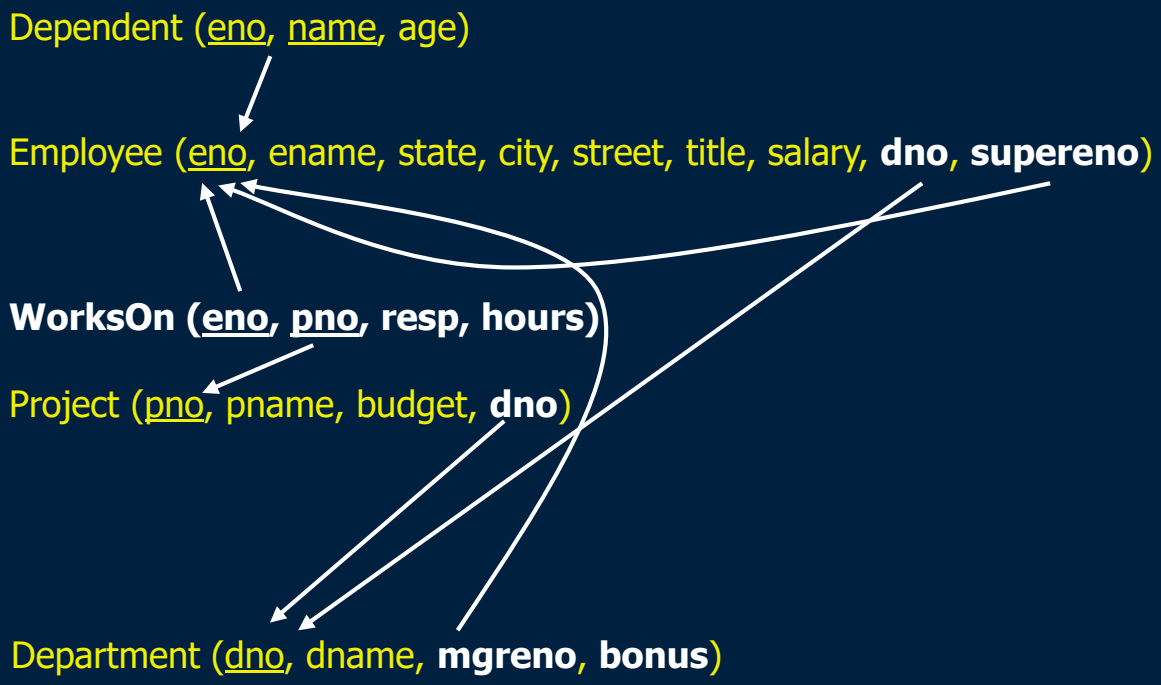


Project (pno, pname, budget, dno)

Employee (eno, ename, state, city, street, title, salary, dno, supereno)

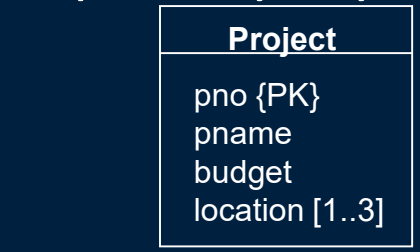
WorksOn (eno, pno, resp, hours)

ER to Relational Mapping
Current Relational Schema - Step #5



ER to Relational Mapping
Step #6: Convert Multi-Valued Attributes

Step #6: Convert a multi-valued attribute into a relation with composite (two or more attributes) primary key consisting of the attribute value plus the primary key of the attribute's entity.

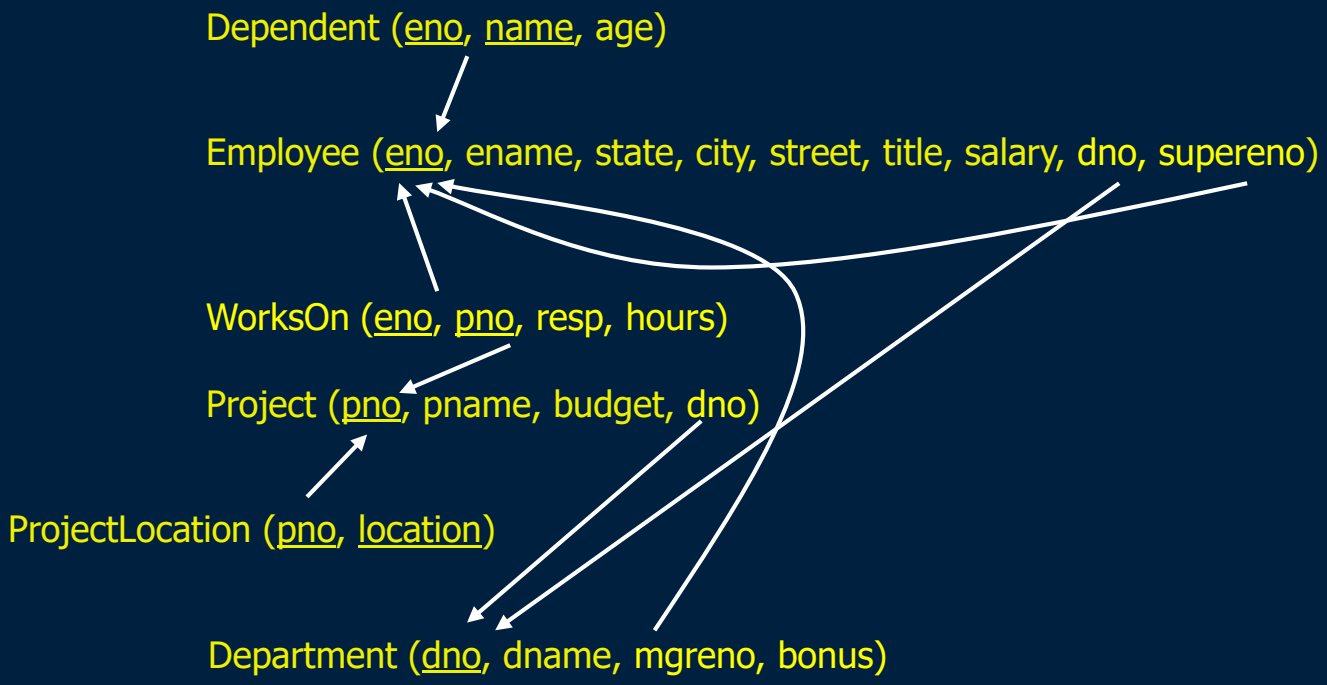


Project (pno, pname, budget, dno)

ProjectLocation (pno, location)

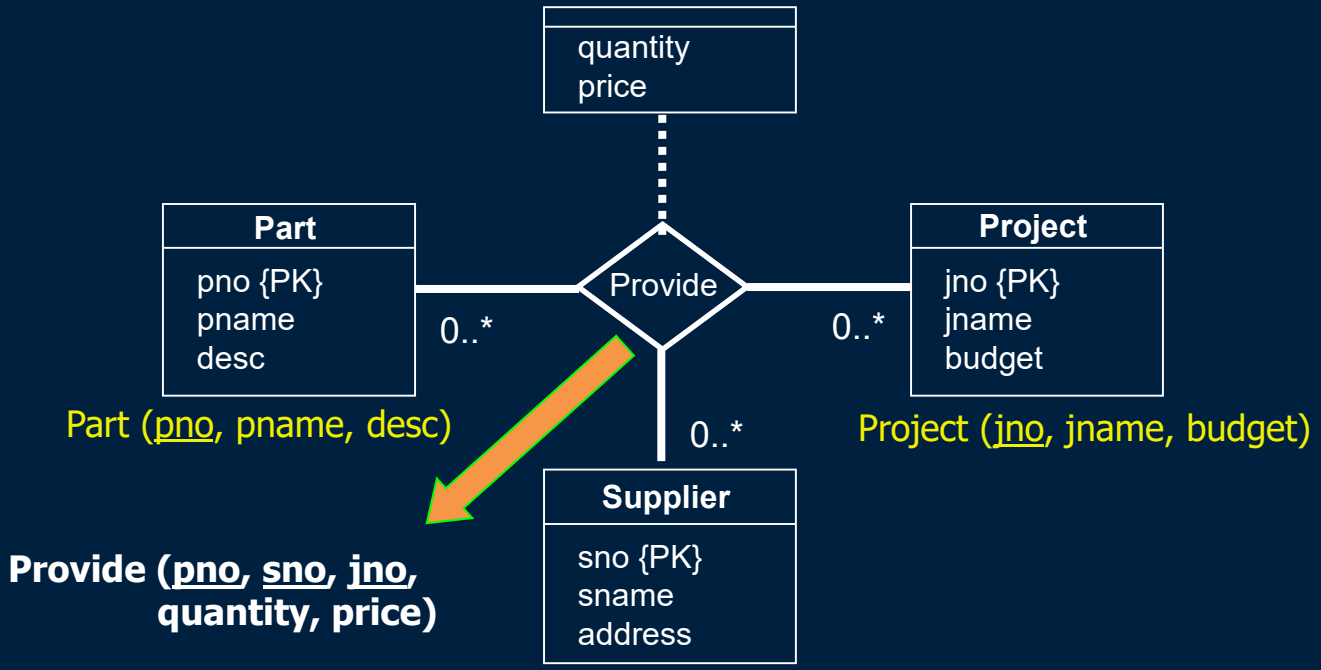
Project (pno, pname, budget, dno)

ER to Relational Mapping
Final Relational Schema



ER to Relational Mapping
Step #7: Convert n-ary Relationships

Step #7: Convert *n*-ary relationships by creating a new relation with primary keys from all entities and foreign keys back to them.



EER to Relational Mapping

An additional step is necessary to convert subclasses and superclasses to the relational model.

We have several different approaches:

- 1) Create a separate relation for each superclass and subclass.
  - Most general technique that we will use.
- 2) Create relations for subclass only.
  - Only works if superclass has mandatory participation.
- 3) Create a single relation with one type attribute.
  - Attribute is used to indicate the type of object (subclass) in the row.
  - Works only if the subclasses are disjoint.
- 4) Create a single relation with multiple type attributes.
  - Have a boolean valued attribute for each subclass. True if in subclass.
  - Works if subclasses may be overlapping.

ER to Relational Mapping
Step #8: Convert Subclasses

Step #8: Convert subclasses and superclasses by creating a relation for each subclass and superclass. Link the subclasses to the superclass using foreign key references.

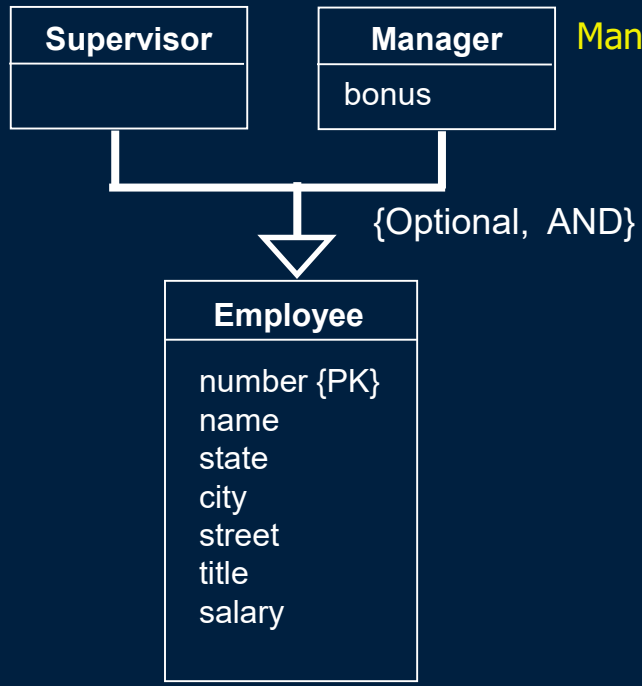
Given a superclass C and set of subclasses S_1, S_2, …, S_n:

- Create a relation R for C.
- The primary key for R is the primary key of the superclass.
- Create relations R_1, R_2, …, R_n for subclasses S_1, S_2, …, S_n.
- The primary key for each R_i is the primary key of the superclass.
- For each R_i, create a foreign key to R using the primary key attributes.

ER to Relational Mapping
Step #8: Convert Subclasses

Step #8: Create a relation for each subclass and superclass. Link the subclasses to the superclass using foreign key references.

Supervisor (eno)



Manager (eno, bonus)

Employee (eno, ename, state, city, street, title, salary, supereno)

Summary of ER to Relational Mapping

ER Model	Relational Model
Entity Type	Relation
1:1 or 1:N Relationship Type	Foreign key (from N-side to 1-side)
M:N Relationship Type	"Relationship" relation and 2 foreign keys
n-ary Relationship Type	"Relationship" relation and *n* foreign keys
Multi-valued attribute	Relation and foreign key
Key attribute	Primary key attribute

ER to Relational Mapping Question

Question: How many of the following statements are ***true***?

1) The M:N relationship mapping rule could be applied to 1:1 and 1:N relationships, as it is more general.

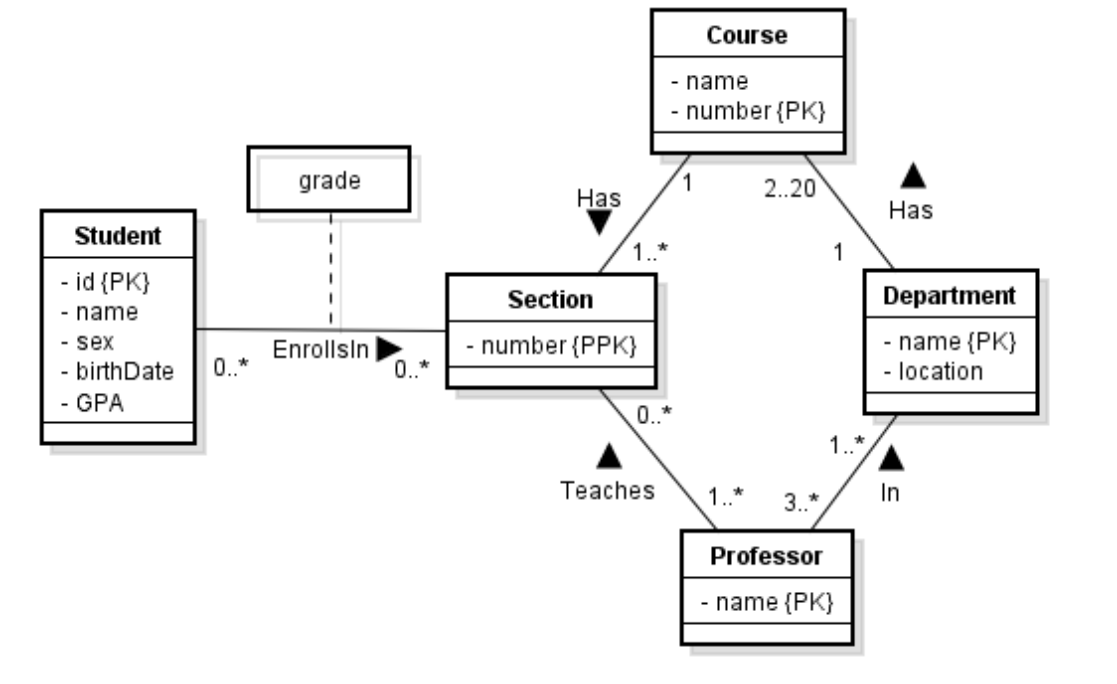
2) A M:N relationship is mapped to a relation in the relational model.

3) The designer has a choice on which side to put the foreign key when mapping a 1:N relationship.

4) When mapping a multi-value attribute, the new table containing the multi-value attribute will have a composite key.

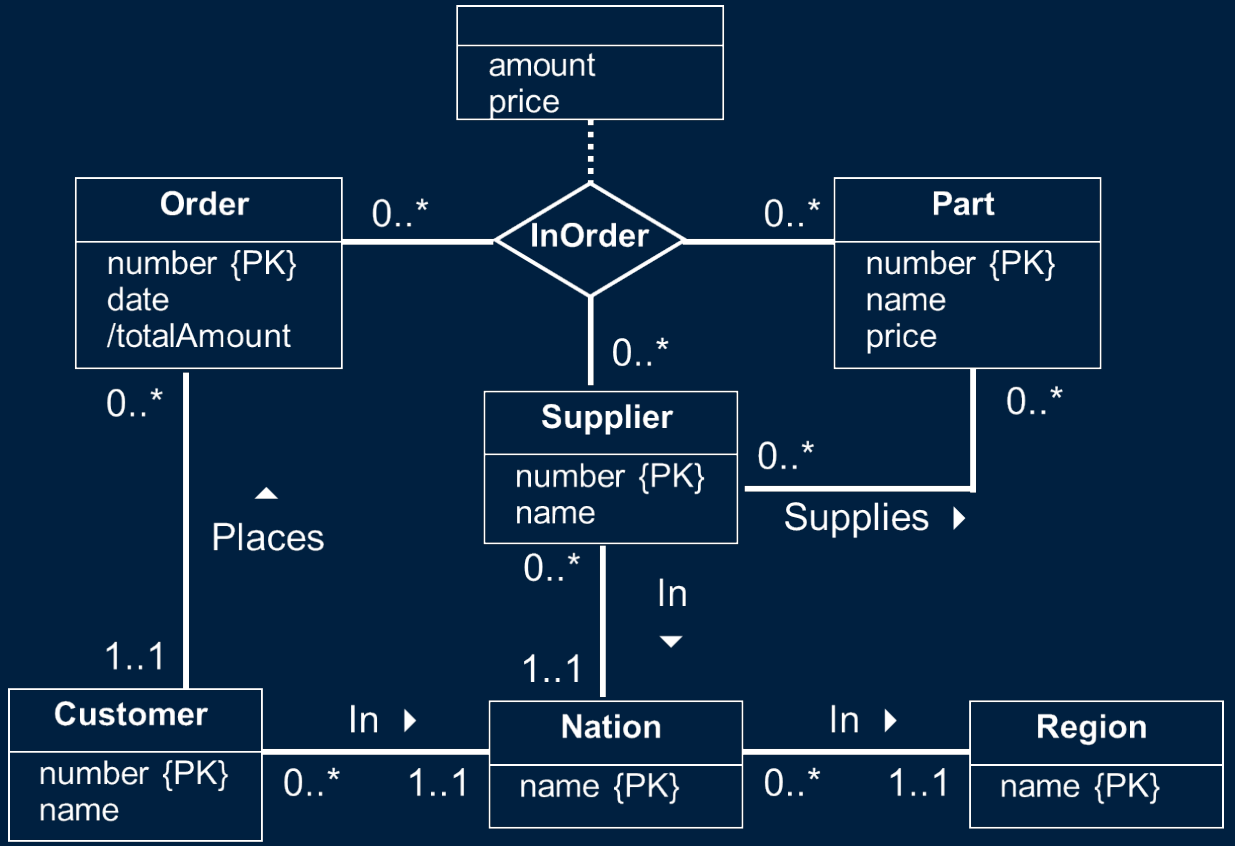
A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

ER to Relational Mapping
University Question



Question: Convert to the relational model.

ER to Relational Mapping
TPC-H Standard Question



Conclusion

There is a straightforward algorithm for converting ER models to relational schemas.

The algorithm involves 7 steps for converting regular ER models, and 8 steps for converting EER models.

In general, these steps convert entities to relations and ER relationships to relations. For 1:1 and 1:N relationships, foreign keys can be used instead of separate relations.

Objectives

Given an ER/EER diagram, convert it into a relational schema using the eight steps.

Discuss the different ways of converting subclasses/superclasses into relational schemas.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Database Programming

COSC 304 – Introduction to Database Systems

Database Programming Overview

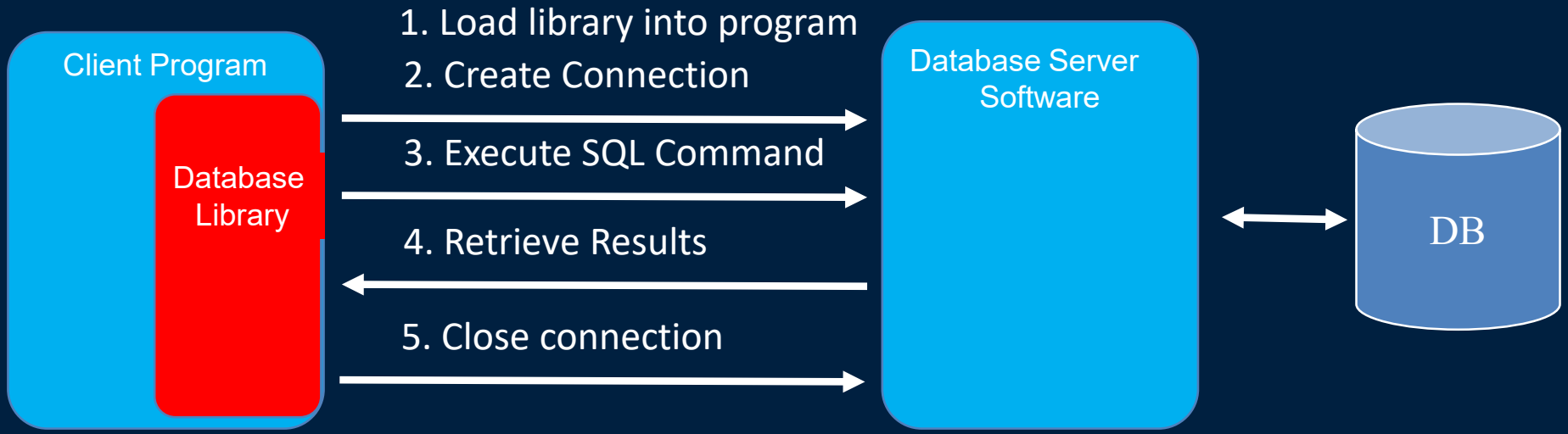
Interaction with a database is often through programs. Programming with a database requires:

- A database server and its connection information
- A programming language for writing the code to query the database
- A library or driver for connecting to the particular database system

General process for programming with a database:

1) Load the database access library
2) Create a connection to the database
3) Execute a SQL command
4) Retrieve database results produced by the command
5) Close the database connection

SQL Programming Architecture



1) Load Library Error:
- Library not found in path
- Wrong library

2) Create Connection Errors:
- Invalid server URL
- Incorrect user/password
- Network issues
- Wrong library

3) Execute SQL Errors:
- Incorrect SQL
- Wrong database/table
- Improper library use

4) Retrieve Results Errors:
- Wrong column name
- Wrong column index
- Off-by-one
- Improper library use

5) Close Connection:
- FORGET TO DO IT!

SQL Programming Architecture Question

Question: **True or False:** The user has sent a command to the database and an error is returned. This error is often related to the user providing an incorrect server URL.

A) true

B) false

Database Programming in Java

JDBC is the most popular method for accessing databases using Java programs.

JDBC is an ***application programming interface*** (***API***) that contains methods to connect to a database, execute queries, and retrieve results.

For each DBMS, the vendor writes a JDBC driver that implements the API. Application programs can access different DBMSs simply by changing the driver used in their program.

JDBC Interfaces

The JDBC API consists of a set of interfaces.

- A Java *interface* is an abstract class consisting of a set of methods with no implementation.
- To create a JDBC driver, the DBMS vendor must implement the interfaces by defining their own classes and writing the code for the methods in the interface.

The main interfaces in the JDBC API are:

- `Driver` - The main class of the entire driver.
- `Connection` - For connecting to the DB using the driver.
- `Statement` - For executing a query using a connection.
- `ResultSet` - For storing and manipulating results returned by a `Statement`.
- `DatabaseMetaData` - For retrieving metadata (schema) information from a database.

JDBC Program Example

```
import java.sql.*;
public class TestJDBCMySQL
{    public static void main(String[] args)
     {    String url = "jdbc:mysql://cosc304.ok.ubc.ca/WorksOn";
          String uid = "user";
          String pw = "testpw";
          try (Connection con = DriverManager.getConnection(url, uid, pw);
               Statement stmt = con.createStatement();)
          {
               ResultSet rst = stmt.executeQuery("SELECT ename,salary FROM emp");
               System.out.println("Employee Name,Salary");
               while (rst.next())
                System.out.println(rst.getString("ename")+","+rst.getDouble("salary"));
          }
          catch (SQLException ex)
          { System.out.println(ex); }
     }
}
```

Import JDBC API

DB Connection Info

Make DB connection

Execute statement

Create statement

Iterate through ResultSet

Using try-with-resources syntax so Statement and Connection objects closed at end of try

JDBC `Driver` Interface

The `Driver` interface is the main interface that must be implemented by a DBMS vendor when writing a JDBC driver.

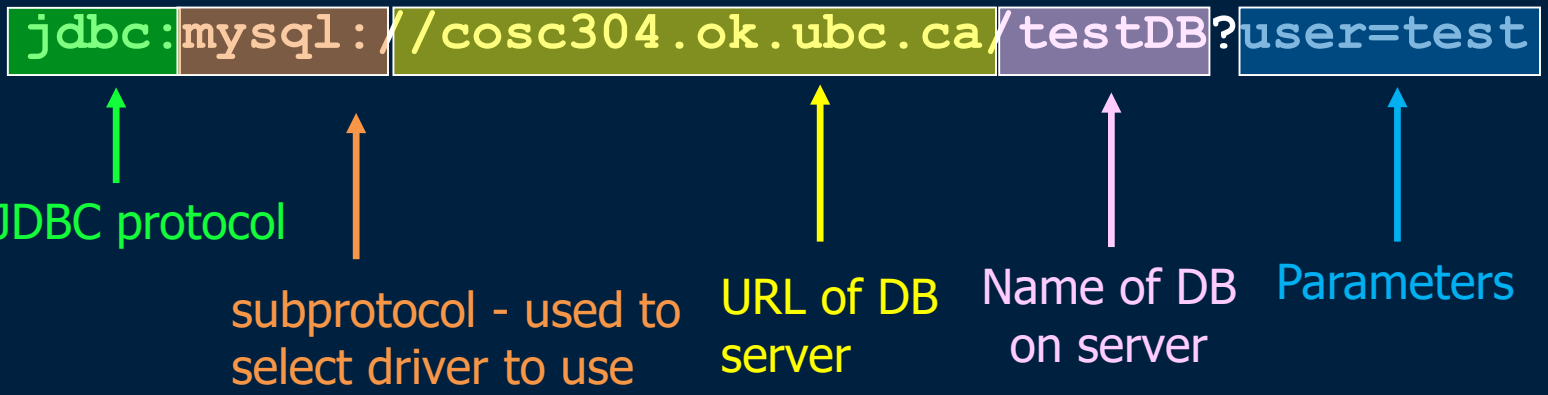
- The class itself does not do very much except allow a connection to be made to a database through the driver.
- Note that you do not call the `Driver` class directly to get a connection. Drivers self-register with the `DriverManager` so `Class.forName()` is no longer needed.
- When you call `DriverManager.getConnection()`, the `DriverManager` will attempt to locate a suitable driver using the URL.

JDBC URL

JDBC specifies connection information in a URL that has the form:

```
protocol:dbprotocol://server/database?params
```

Example:



JDBC `Connection` Interface

The `Connection` interface contains methods for managing a connection or session.

- A connection is opened after the call to `getConnection()` and should be closed when you are done.
- The `Connection` interface is used to create statements for execution on the database.

```
Connection con = DriverManager.getConnection(url, uid, pw);
Statement stmt = con.createStatement();
...
con.close();
```

`Connection` Interface and MetaData

A `Connection` to a database can also be used to retrieve the database metadata (or schema).

- This is useful when you are writing generic tools where you do not know the schema of the database that you are querying in advance.

The method `getMetaData()` can be used to retrieve a `DatabaseMetaData` object.

DatabaseMetaData Example

```
String []tblTypes = {"TABLE"}; // What table types to retrieve
try {
    DatabaseMetaData dmd = con.getMetaData();  // Get metadata
    ResultSet rs1, rs2, rs5;

    System.out.println("List all tables in database: ");
    rs1 = dmd.getTables(null, null, "%", tblTypes);
    while (rs1.next()) {
        String tblName = rs1.getString(3);
        Statement stmt = con.createStatement();
        rs2 = stmt.executeQuery("SELECT COUNT(*) FROM "+tblName);
        rs2.next();
        System.out.println("Table: "+tblName+" # records: "+rs2.getInt(1));
        rs5 = dmd.getColumns(null, null, tblName, "%");
        System.out.println("  Attributes: ");
        while (rs5.next()) {
            System.out.println(rs5.getString(4));
        }
    } // end outer while
} // end try
```

JDBC `Statement` Interface

The `Statement` interface contains abstract methods for executing a single static SQL statement and returning the results it produces.

```
Statement stmt = con.createStatement();
ResultSet rst = stmt.executeQuery("SELECT ename, salary
                                   FROM emp");
```

- The `Statement` object is created by calling `Connection.createStatement()`.
- The statement is then executed by calling `executeQuery()` and passing the SQL string to execute.

JDBC `Statement` Interface (2)

There are two important variations of executing statements that are important and are used often.

- 1) The `Statement` executed is an `INSERT`, `UPDATE`, or `DELETE` and no results are expected to be returned:

```
rowcount = stmt.executeUpdate("UPDATE emp Set salary=0");
```

- 2) The `Statement` executed is an `INSERT` which is creating a new record in a table whose primary key field is an autonumber field:

```
rowcount = stmt.executeUpdate("INSERT Product
                                    VALUES ('Prod. Name')",
                                    Statement.RETURN_GENERATED_KEYS );
ResultSet autoKeys = stmt.getGeneratedKeys();
```

`PreparedStatement` and `CallableStatement`

There are two special types of `Statement` objects:

- `PreparedStatement` - extends `Statement` and is used to execute precompiled SQL statements.
  - Useful when executing the same statement multiple times with different parameters as the DBMS can optimize its parsing and execution.
  - Also useful to prevent SQL injection attacks.

```
String SQL = "UPDATE emp SET salary = ? WHERE ID = ?";
PreparedStatement pstmt = con.prepareStatement(SQL);
pstmt.setBigDecimal(1, 55657.34);   // Set parameters
pstmt.setString(2,"E1");
int rowcount = pstmt.executeUpdate();
```

- `CallableStatement` - extends `PreparedStatement` and is used to execute stored procedures.
  - Stored procedures are precompiled SQL code stored at the database that take in parameters for their execution.

JDBC `ResultSet`

The `ResultSet` interface provides methods for manipulating the result returned by the SQL statement.

- Remember that the result is a relation (table) which contains rows and columns.
- The methods provide ways of navigating through the rows and then selecting columns of the current row.
- A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.
- The `next()` method moves the cursor to the next row, and because it returns false when there are no more rows in the `ResultSet` object, it can be used in a while loop to iterate through the result set.

JDBC `ResultSet` (2)

By default a `ResultSet` is not *updatable* and the cursor can move *forward-only* (can only use `next()` method).

```
while (rst.next())
{    System.out.println(rst.getString("ename")+","+rst.getDouble(2));
}
```

- Remember, the first call to `next()` places the row cursor on the first row as the cursor starts off before the first row.
- Use the `getType()` methods to retrieve a particular type.
  - `getArray()`, `getBlob()`, `getBoolean()`, `getClob()`, `getDate()`, `getDouble()`, `getFloat()`, `getInt()`, `getLong()`, `getObject()`, `getString()`, `getTime()`
  - All methods take as a parameter the column index in the `ResultSet` (***indexed from 1***) or the column name and return the requested type.
  - Java will attempt to perform casting if the type you request is not the type returned by the database.

Scrollable ResultSets

It is also possible to request ResultSets that allow you to navigate backwards as well as forwards.

- Request ResultSet type during createStatement.

```
// rs will be scrollable and read-only
Statement stmt = con.createStatement(
                                ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("SELECT eno, ename FROM emp");
```

Scrollable ResultSets allow you to navigate in any direction through it, and these methods can now be used:

- absolute(int row) - set cursor to point to the given row (starting at 1)
- afterLast(), beforeFirst(), first(), last(), next(), previous()
- Scrollable ResultSets were introduced in JDBC 2.0 API and may be less efficient than forward-only ResultSets.

Updatable `ResultSets`

Updatable `ResultSets` allow you to update fields in the query result and update entire rows.

Updating an existing row:

```
// rs will be scrollable and updatable record set
Statement stmt = con.createStatement(
                            ResultSet.TYPE_SCROLL_INSENSITIVE,
                            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT eno, ename FROM emp");

rs.absolute(2);                     // Go to the 2nd row
rs.updateString(2,"Joe");           // Change name of employee
rs.updateRow();                     // Update data source
```

Updatable `ResultSets` (2)

Adding a new row to a `ResultSet`:

```
// rs will be scrollable and updatable record set
Statement stmt = con.createStatement(
                          ResultSet.TYPE_SCROLL_INSENSITIVE,
                          ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT eno, ename FROM emp");

rs.moveToInsertRow();                    // Move cursor to insert row
rs.updateString(1,"E9");
rs.updateString("ename","Joe Smith");
rs.insertRow();                          // Insert new row in DB
rs.moveToCurrentRow();                   // Move cursor to row you were on
                                         //  before insert
```

JDBC API Question

Question: Which one is not a JDBC API interface?

A) `Driver`

B) `Connection`

C) `Statement`

D) `ResultSet`

E) `HashMap`

JDBC API Question

Question: Select a **true** statement.

A) When a `ResultSet` is first opened, the current row is 0.

B) When a `ResultSet` is first opened, the current row is 1.

C) When asking for columns, the first column is index 0.

D) The method call `first()` is allowed for a forward-only `ResultSet`.

Custom APIs

Many databases have custom APIs for various languages.

- For example, MySQL has APIs for C/C++ and PHP in addition to ODBC and JDBC drivers.

When to use a custom API?

- Custom APIs may be useful when the ODBC/JDBC standard does not provide sufficient functionality.
- Also useful if *know* that application will only access one database. (Be careful with this .. think of the future).

Custom APIs may have improved performance and increased functionality. The disadvantage is that your code is written for a specific DBMS which makes changes difficult.

- If you use a custom API, always isolate the database access code to a few general classes and methods!

Object-Relational Mapping
Java Persistence Architecture (JPA)

A huge challenge with database programming is converting the database results to and from Java objects. This is called object-relational mapping, and it is tedious and error-prone.

- ***Impedance mismatch*** - Database returns values in tables and rows and Java code manipulates objects, classes, and methods.

Various vendors (e.g. Hibernate) have produced object-relational mapping technologies that help the programmer convert database results into Java objects.

The Java Persistence Architecture (JPA) has been developed as a standard interface. Vendors can then implement the interface in their products.

JDBC Question

Create a JDBC program that:

- Connects to `WorksOn` database on `cosc304.ok.ubc.ca`.
- Prints on the console each department and its list of projects.

Variant:

- Output in reverse order by department number. Two versions:
  - Change SQL
  - Use scrollable ResultSets (hint: previous() method).

Challenge:

- Improve your code so that it prints the department number, name, and how many projects in that department THEN the list of projects.

Conclusion

Querying databases can be done from any language with many different APIs. Querying has these steps:

1) Load the database access library
2) Create a connection to the database
3) Execute a SQL command
4) Retrieve database results produced by the command
5) Close the database connection

Queried MySQL and Microsoft SQL Server with Java.

JDBC is an API for accessing databases using Java. Steps:

- Load a `Driver` for the database, Make a `Connection`, Create a `Statement`, Execute a query to produce a `ResultSet`, Navigate the `ResultSet` to display results or update the DB

Objectives

- Explain the common steps in querying a database using a programming language.
- Draw and explain the database-program architecture and the key components.
- List the main JDBC classes (`Driver`, `Connection`, `Statement`, `ResultSet`) and explain the role of each.
- Discuss the different types of `ResultSets` including scrollable and updatable `ResultSets`.
- Write a simple JDBC program (given methods of the JDBC API).
- Discuss and explain the advantages and disadvantages of using a standard API versus a vendor-based APIs.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Database Programming with Python and R

COSC 304 – Introduction to Database Systems

SQL Programming Overview

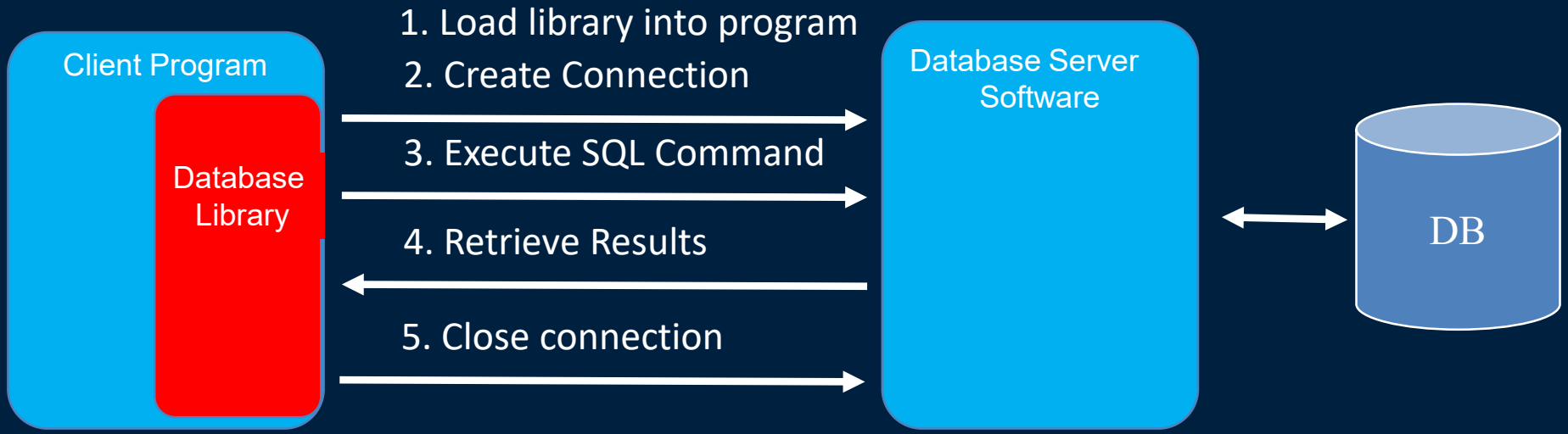
Interaction with a database is often through programs. Programming with a database requires:

- A database server and its connection information
- A programming language for writing the code to query the database
- A library or driver for connecting to the particular database system

General process for programming with a database:

1) Load the database access library
2) Create a connection to the database
3) Execute a SQL command
4) Retrieve database results produced by the command
5) Close the database connection

SQL Programming Architecture



1) Load Library Error:
- Library not found in path
- Wrong library

2) Create Connection Errors:
- Invalid server URL
- Incorrect user/password
- Network issues
- Wrong library

3) Execute SQL Errors:
- Incorrect SQL
- Wrong database/table
- Improper library use

4) Retrieve Results Errors:
- Wrong column name
- Wrong column index
- Off-by-one
- Improper library use

5) Close Connection:
- FORGET TO DO IT!

SQL Programming in Python

Python can connect to many SQL databases by loading the proper library. We will use MySQL, Microsoft SQL Server, and SQLite.

- Python has a standard API called DB-API 2.0.

Step #1 – Install Libraries:

- MySQL: `conda install mysql-connector-python`
- SQL Server:
  - https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server
  - Install ODBC driver for SQL Server then: `pip install pyodbc`
- SQLite: `pip install pysqlite3`
- Note: May have to find `conda.bat` or `conda.sh` or `pip`. Check `Anaconda3/Library/bin`.

Python and MySQL

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='rlawrenc', password='test',
                                  host='cosc304.ok.ubc.ca', database='workson')
    cursor = cnx.cursor()
    query = "SELECT eno, ename, salary FROM emp WHERE salary < 50000"
    cursor.execute(query)
    for (eno, ename, salary) in cursor:
        print(eno, ename, salary)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```

Import MySQL API

DB Connection Info

Execute query and process results

Close connection

Python and MySQL Query with Parameters

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='rlawrenc', password='test',
                  host='cosc304.ok.ubc.ca', database='workson')
    cursor = cnx.cursor()
    query = "SELECT pno, count(*) FROM emp NATURAL JOIN workson "
            "WHERE salary < %s and ename > %s GROUP BY pno"
    cursor.execute(query, (45000, 'L'))
    for (pno, cnt) in cursor:
        print(pno, cnt)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```

Python and MySQL INSERT Statement

```
try:
    cnx = mysql.connector.connect(user='rlawrenc', password='test',
                host='cosc304.ok.ubc.ca', database='db_rlawrenc')
    cursor = cnx.cursor()
    sql = "INSERT INTO emp (eno, ename, salary) VALUES (%s, %s, %s)"
    cursor.execute(sql, ('E10', 'Test Person', 100000))
    cnx.commit() # Update data by committing transaction on connection

    sql = "DELETE FROM emp WHERE eno = 'E10'"
    cursor.execute(sql) # Remove employee just added
    cursor.close()
    cnx.commit()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```

Try it: Python and MySQL

Question: Write a program that queries the WorksOn database and returns the employees grouped by title where the employee name is after `'J'`. The output should display their title and the average salary for that title. Connection info:

- `cnx = mysql.connector.connect(user='rlawrenc', password='test', host='cosc304.ok.ubc.ca', database='workson')`

Output:

```
EE 30000.000000
ME 40000.000000
PR 20000.000000
SA 50000.000000
```

ODBC Overview

ODBC (Open Database Connectivity) is a standard database ***application programming interface*** (***API***) developed by Microsoft for connecting to databases.

- Java has a similar standard API called JDBC.

The ODBC API contains methods that allow user programs to connect to a database, execute queries, and retrieve results.

For each database system, the vendor writes an **ODBC driver** that implements the API. Application programs can access different systems simply by changing the driver used in their program.

Python and Microsoft SQL Server

ODBC is the underlying technology used to connect to Microsoft data sources (Excel, Access, SQL Server, etc.) from Python.

ODBC is language independent (developed first in C/C++) and has been implemented in Python and packaged as library `pyodbc`.

Python and Microsoft SQL Server

```
import pyodbc
try:
    cnx = pyodbc.connect("""DRIVER={ODBC Driver 17 for SQL Server};
                            SERVER=sql04.ok.ubc.ca;
                            DATABASE=workson;UID=rlawrenc;PWD=test""")
    cursor = cnx.cursor()
    cursor.execute("SELECT * FROM emp WHERE salary < ?", [50000])
    for row in cursor:
        print(row[0],row[1], row.salary)
except pyodbc.Error as err:
    print(err)
finally:
    cnx.close()
```

Try it: Python and Microsoft SQL Server

Question: Write a program that asks the user for a total salary value. Write a query on the workson database that returns the projects where the total salary of employees working on the project is greater than the value the user entered. Connection info:

- `cnx = pyodbc.connect("""DRIVER={ODBC Driver 17 for SQL Server}; SERVER=sql04.ok.ubc.ca; DATABASE=workson;UID=rlawrenc;PWD=test""")`

Output:

```
Enter total salary:75000
('P1   ', Decimal('80000.00'))
('P2   ', Decimal('120000.00'))
('P3   ', Decimal('80000.00'))
```

Python and SQLite

SQLite is an embedded database designed to be used within other programs.

The amount of data stored in a SQLite database is often considerably smaller than other systems. However, the system is easy to use and transport data between systems.

Python and SQLite

```
import sqlite3
try:
    cnx = sqlite3.connect("test.db")
    cursor = cnx.cursor()
    cursor.execute("CREATE TABLE test (id int, name text)")
    cursor.execute("INSERT INTO test VALUES(1, 'Joe')")
    cursor.execute("INSERT INTO test VALUES(2, 'Jen')")
    cursor.execute("INSERT INTO test VALUES(3, 'Jeff')")
    cnx.commit()
    cursor.execute("SELECT * FROM test WHERE name > 'Je'")
    for row in cursor:
        print(row)
    cursor.execute("DROP TABLE test")
except sqlite3.Error as err:
    print(err)
finally:
    cnx.close()
```

SQL Programming in R

R can connect to many SQL databases by loading the proper library. We will use MySQL, Microsoft SQL Server, and SQLite.

- R has `DBI` standard API for querying databases.
  - https://www.rdocumentation.org/packages/DBI

Step #1 – Install Libraries:

- MySQL: `install.packages("RMariaDB")`
- SQL Server: `install.packages("odbc")`
- SQLite: `install.packages("RSQLite")`

R and MySQL

```
library("RMariaDB")
con <- dbConnect(RMariaDB::MariaDB(), user='rlawrenc', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

# List database tables
dbListTables(con)

# Execute a query
res <- dbGetQuery(con, "SELECT * FROM emp")

# Disconnect
dbDisconnect(con)
```

R and Microsoft SQLServer

```
library("DBI")
library("odbc")

db <- dbConnect(odbc::odbc(),
                Driver = 'ODBC Driver 17 for SQL Server',
                Server = 'sql04.ok.ubc.ca', Database = "workson",
                UID='rlawrenc', PWD='test')

data <- dbGetQuery(db,"SELECT * FROM emp")
data

# Disconnect
dbDisconnect(con)
```

R and SQLite

```
library("RSQLite")
library("DBI")

con <- dbConnect(RSQLite::SQLite(), ":memory:")

# Write data frame to table
testData <- data.frame(id = 1:5,
                       name=c("Abe", "Ben", "Cindy", "Dana", "Emma"))

dbWriteTable(con, "test", testData)
data <- dbGetQuery(con,"SELECT * FROM test")
data

# Disconnect
dbDisconnect(con)
```

R Database Querying with dplyr

```
library(dplyr)
library(dbplyr)

con <- dbConnect(RMariaDB::MariaDB(), user='rlawrenc', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

# Execute a query
data <- tbl(con, "emp") %>%
  group_by(title) %>%
  summarise(
    totalSalary = sum(salary),
    totalEmp = n())
show_query(data)
data # Executes query

# Disconnect
dbDisconnect(con)
```

SQL Programming: R and Python Question

Question: **True or False:** Although the API methods may be slightly different, the general approach to querying a database is very similar in R and Python.

A) true

B) false

R SQL Programming Question

Question: What is the API used to query Microsoft SQL Server with R?

A) JDBC

B) RMariaDB

C) pyodbc

D) ODBC

Try it: R and SQLite

Question: Write a program that creates a data frame with 5 rows consisting of id (integer) and message (string). Save the data frame to SQLite. Query SQLite to only return values with id < 12 and message has an `"e"` in it. Print the results.

Data:

```
  id message
1 10   Hello
2 11 Goodbye
3 12  Friend
4 13     Ant
5 14     Out
```

Conclusion

Querying databases can be done from any language with many different APIs. Querying has these steps:

1) Load the database access library
2) Create a connection to the database
3) Execute a SQL command
4) Retrieve database results produced by the command
5) Close the database connection

We have seen how to query MySQL, Microsoft SQL Server, and SQLite using both Python and R.

Objectives

- Explain the common steps in querying a database using a programming language.
- Draw and explain the database-program architecture and the key components.
- Write a simple program to query a database in both Python and R.
- Use a variety of databases including MySQL, Microsoft SQL Server, and SQLite.
- Explain the purpose of ODBC.
- List the components of a database connection URL.
- Use dplyr to query a database.
- Debug and resolve database connection and usage issues in Python and R code.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Database Web Programming

COSC 304 – Introduction to Database Systems

Database Web Programming Overview

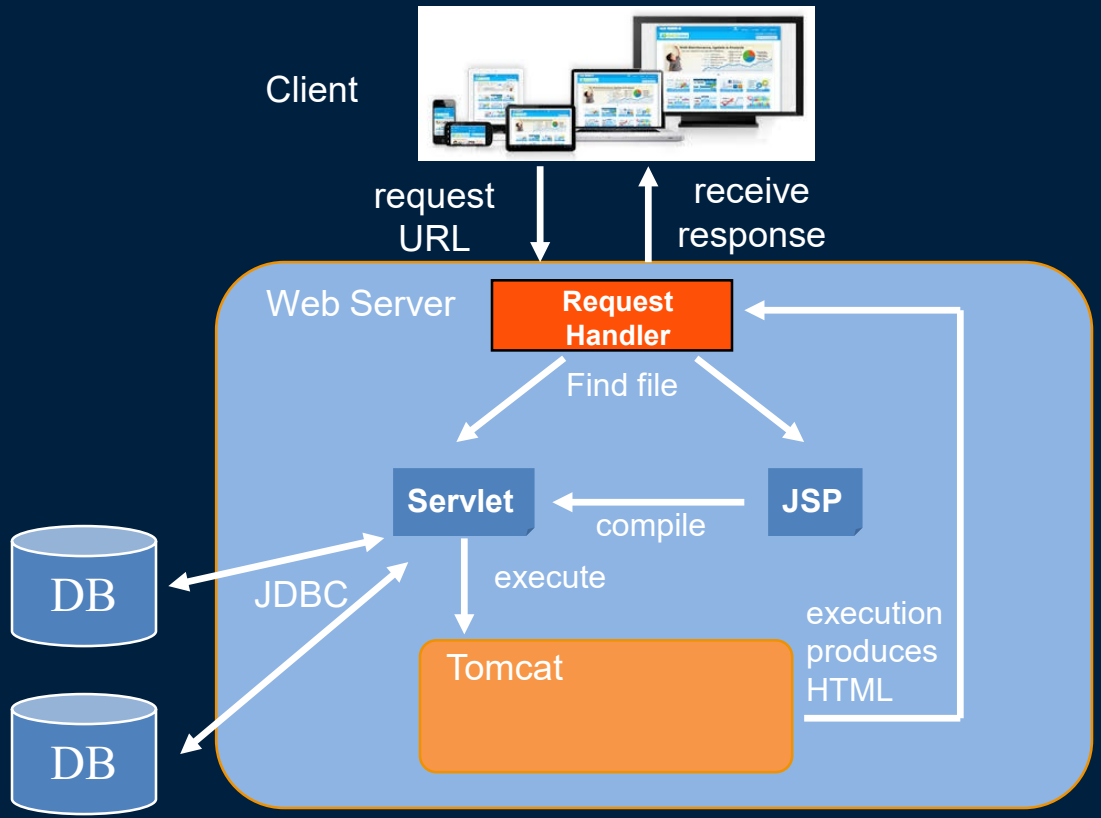
Web applications consist of a client interface and a server component.

The client interface does not access the database directly. The server code accesses the database and provides data as needed to the client.

- The server code may be JSP/Servlets, PHP, Python, etc.
- The client code is HTML/JavaScript.

HTTP is a stateless protocol which requires special handling to remember client state.

Dynamic Web Server Architecture



Hypertext Markup Language (HTML)

Hypertext Markup Language (***HTML***) describes the layout of a web page including structure, style, images, and links.

An HTML document has ***tags*** in angle brackets **<** and **>** as markup.

- In HTML 5, tags are not case-sensitive. We will use lower case as convention.

Example:

```
<html>
  <head>
    <title>Example HTML with Hyperlink</title>
  </head>
  <body>  <!-- This is a comment-->
    <p>Link: <a href="https://www.apple.com">Apple</a></p>
  </body>
</html>
```

Tables in HTML

Data can be displayed in tables using the **<table></table>** tags.

- Rows are enclosed in table row **<tr></tr>** tags, and each cell is denoted using table data **<td></td>** tags. A header cell uses the **<th></th>** tags.
- A table may have a caption centered at the top of the table by using the **<caption></caption>** tags.

```
<table border="1" style="background-color:green">
  <caption>This is my table caption.</caption>
  <tr><th>Tag</th><th>Purpose</th></tr>
  <tr><td>table</td><td>Start and end entire table</td></tr>
  <tr><td>tr</td><td>Start and end each row</td></tr>
  <tr><td>th</td><td>Special column header formatting</td></tr>
  <tr><td>td</td><td>Start and end each data cell</td></tr>
  <tr><td colspan="2">COLSPAN has cell span multiple columns</td></tr>
   <tr><td rowspan="2">ROWSPAN</td><td>ROWSPAN spans rows</td></tr>
</table>
```

Dynamic HTML and JDBC

Dynamic HTML involves constructing HTML output dynamically on the server then sending it to the client.

By building the content dynamically, it can have different content for each user. Typically, this dynamic content is retrieved from databases and then inserted into the HTML.

The two standard ways for generating dynamic content using Java are ***Java Server Pages*** (***JSPs***) and ***Servlets***. Since both use Java, they can use JDBC to retrieve dynamic, database content.

JSP/Servlet Question

Question: **True or False:** Java JSP/Servlet code runs in the browser.

A) true

B) false

JSP/Servlet Question #2

Question: **True or False:** JavaScript running in the browser can connect to a database directly using JDBC.

A) true

B) false

Java Server Pages (JSPs)

Java Server Pages (***JSPs***) provide a layer above `Servlets` that allow you to mix HTML and calls to Java code.

JSPs are converted into `Servlets`, and are easier to work with for those familiar with HTML.

Java Server Pages (JSPs) (2)

How to create JSPs:

- 1) Create a file with a `.jsp` extension that contains HTML.
- 2) Include JSP Scripting Elements in the file such as:
  - Expressions `<%= expression %>`
    - Note that an expression returns a string into the HTML.
  - Scriptlets `<% code %>`
  - Declarations `<%! Code %>`
  - Directives `<%@ variable directive %>`
  - Comments `<%-- JSP Comment -->`
- 3) Can use pre-defined variables:
  - `request` `HttpServletRequest`
  - `response` `HttpServletResponse`
  - `session` `HttpSession`
  - `out` `PrintWriter`
    - Prints to HTML document.

JSP "Hello World!"

```
<html>
<head>
<title>Hello World in JSP</title>
</head>
<body>

<% out.println("Hello World!"); %>

</body>
</html>
```

← Java code embedded in HTML file.

Useful JSP Code

Include another file:

```
<%@ include file="commontop.html" %>
```

Import from the Java library:

```
<%@ page import="java.util.*" %>
```

Declare and use a variable:

```
<%! private int accessCount = 0 %>
<h2>Accesses to page since server reboot:
<%= ++accessCount %></h2>
```

Access session and request information:

```
<h2>Current time: <%= new java.util.Date() %></h2>
<h2>Remote Host: <%= request.getRemoteHost()%></h2>
<h2>Session ID: <%= session.getID() %></h2>
```

JSP and JDBC

```
<%@ page import="java.sql.*" %>
<html><head><title>Query Results Using JSP</title></head><body>
<% // Note: Need to define url, uid, pw
  try
  ( Connection con = DriverManager.getConnection(url, uid, pw);
    Statement stmt = con.createStatement(); )
  {
    String sql = "SELECT ename, salary FROM emp";
    ResultSet rst = stmt.executeQuery(sql);
    out.println("<table><tr><th>Name</th><th>Salary</th></tr>");
    while (rst.next()) {
    {  out.println("<tr><td>"+rst.getString(1)+"</td>"
                  +"<td>"+rst.getDouble(2)+"</td></tr>"); }
    out.println("</table>");
  } catch (SQLException ex)       { out.println(ex); }
%></body></html>
```

Answering Queries using HTML Forms

One of the common uses of dynamic web pages is to construct answers to user queries expressed using HTML forms.

The HTML code will contain a `FORM` and the `FORM ACTION` will indicate the server code to call to process the request.

In our example, the JSP or Servlet gets the HTML request (and parameters in the URL), queries the database, and returns the answers in a table.

HTTP GET and POST Methods

GET and POST are two ways a HTTP client can communicate with a web server. GET is used for getting data from the server, and POST is used for sending data there.

- GET appends the form data (called a query string) to an URL, in the form of key/value pairs, for example, name=John.
  - In the query string, key/value pairs are separated by & characters, spaces are converted to + characters, and special characters are converted to their hexadecimal equivalents.
  - Since the query string is in the URL, the page can be bookmarked. The query string is usually limited to a relatively small amount of data.
- POST passes data of unlimited length as an HTTP request body to the server. The user working in the client Web browser cannot see the data that is being sent, so POST requests are ideal for sending confidential or large amounts of data to the server.

GET vs. POST Question

Question: Select a **true** statement.

A) Data sent using `POST` is passed as part of the URL.

B) A `GET` request with all parameters cannot be bookmarked.

C) Your JSP/servlet can do the same action for both `GET` and `POST` requests.

D) A `GET` request is used to send a large amount of data.

HTML Form

```
<html>
<head><title>Querying Using JSP and Forms</title></head>
<body>

<h1>Enter the name and/or department to search for:</h1>

<form method="get"
   action="https://cosc304.ok.ubc.ca/tomcat/EmpQuery.jsp">
Name:<input type="text" name="empname" size="25">
Dept:<input type="text" name="deptnum" size="5">
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

</body></html>
```

FORM action and type

FORM buttons and input fields

Reading Parameters

Parameters passed into the JSP are read using the `request` object:

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
```

Reading Parameters
Badly Building a Query – SQL Injection!

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
try
( Connection con = DriverManager.getConnection(url, uid, pw);
  Statement stmt = con.createStatement(); )
{
  String SQLSt="SELECT ename, salary, dno FROM emp WHERE 1=1";
  if (empName != null && !empName.equals(""))
    SQLSt = SQLSt + " and ename LIKE '%"+empName+"%'";

  if (deptNum != null && !deptNum.equals(""))
    SQLSt = SQLSt + " and dno ='"+deptNum+"'";

  ResultSet rst = stmt.executeQuery(SQLSt);
  ...
```

Building a Query using a PreparedStatement

```
String empName = request.getParameter("empname");
String deptNum = request.getParameter("deptnum");
try
( Connection con = DriverManager.getConnection(url, uid, pw);)
{
  String sql = "SELECT ename, salary, dno FROM emp";
  boolean hasEmp = empName != null && !empName.equals("");
  boolean hasDept= deptNum != null && !deptNum.equals("");
  PreparedStatement pstmt=null;
  ResultSet rst = null;
  if (!hasEmp && !hasDept)
  {
    pstmt = con.prepareStatement(sql);
    rst = pstmt.executeQuery();
  }
```

Building a Query using a PreparedStatement

```
else if (hasEmp)
{ empName = "%"+empName+"%";
  sql += " WHERE ename LIKE ?";
  if (hasDept)  sql += " AND dno = ?";
  pstmt = con.prepareStatement(sql);
  pstmt.setString(1, empName);
  if (hasDept) pstmt.setString(2, deptNum);
  rst = pstmt.executeQuery();
}
else if (hasDept)
{ sql += " WHERE dno = ?";
  pstmt = con.prepareStatement(sql);
  pstmt.setString(1, deptNum);
  rst = pstmt.executeQuery();
}
```

JSP Examples
Implementing Login and Security

How do you password protect files in your web site?

The basic idea is to:

- Create a login page like `login.jsp`.
- Create a page to validate the user login (often by connecting to the database to determine if given valid user id/password).
- Create a file containing JSP code that is included in every one of your protected pages. This code:
  - Examines if a session variable flag is set indicating if logged in.
  - If user is logged in, show page. Otherwise redirect to login page.

Implementing Login and Security

login.jsp

```
<html>
<head><title>Login Screen</title></head>
<body>
<center>
<h3>Please Login to System</h3>

<%
// Print prior error login message if present
if  (session.getAttribute("loginMessage") != null)
    out.println("<p>"+
        session.getAttribute("loginMessage").toString()+"</p>");
%>

<br>
```

Implementing Login and Security

login.jsp (2)

```
<form name="MyForm" method=post action="validateLogin.jsp">
<table width="40%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td><div align="right">Username:</div></td>
<td><input type="text" name="username" size=8 maxlength=8></td>
</tr>
<tr>
<td><div align="right">Password:</div></td>
<td><input type="password" name="password" size=8
maxlength=8"></td>
</tr>
</table>
<input class="submit" type="submit" name="Sub" value="Log In">
</form>
```

Implementing Login and Security

validateLogin.jsp

```
<%@ page language="java" import="java.io.*" %>
<%
    String authenticatedUser = null;
    session = request.getSession(true);// May create new session
    try
    {
        authenticatedUser = validateLogin(out,request,session);
    }
    catch(IOException e) { out.println(e); }
    if (authenticatedUser != null)
        response.sendRedirect("protectedPage.jsp"); // Success
    else
        response.sendRedirect("login.jsp"); // Failed login
        // Redirect back to login page with a message
%>
```

Implementing Login and Security

validateLogin.jsp (2)

```
<%!
String validateLogin(JspWriter out,HttpServletRequest request,
                     HttpSession session) throws IOException
{
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String retStr = null;
    if(username == null || password == null)
        return null;
    if((username.length() == 0) || (password.length() == 0))
        return null;
    // Should make a database connection here and check password
    // Here just hard-coding password
    if (username.equals("test") && password.equals("test"))
        retStr = username;
```

Implementing Login and Security

validateLogin.jsp (3)

```
    if(retStr != null)
    {   session.removeAttribute("loginMessage");
        session.setAttribute("authenticatedUser",username);
    }
    else
        session.setAttribute("loginMessage","Failed login.");
    return retStr;
}
%>
```

Implementing Login and Security

protectedPage.jsp

```
<html>
<head><title>Password Protected Page</title></head>
<body>

<%@ include file="auth.jsp"%>

<%
String user = (String)session.getAttribute("authenticatedUser");
out.println("<h1>You have access to this page: "+user+"</h1>");
%>

</body>
</html>
```

Implementing Login and Security

auth.jsp

```
<%
    Object authUser = session.getAttribute("authenticatedUser");
    boolean authenticated = authUser == null ? false : true;

    if (!authenticated)
    {
        String loginMessage = "You are not authorized to "+
            "access the URL "+request.getRequestURL().toString();
        session.setAttribute("loginMessage",loginMessage);
        response.sendRedirect("login.jsp");
        return;
    }
%>
```

JSP Examples
Passing Objects Between Pages

How do you pass information between pages?

One way is to encode that information in the URL as parameters. These parameters can be extracted from the `request` object.

If you have a lot of information, it is better to use the `session` object. The session object allows you to store any number of objects that are later looked up by name. Since they remain on the server, performance/security is improved.

Passing Objects Between Pages

sendingPage.jsp

```
<%@ page import="java.util.ArrayList,java.util.Random" %>
<html><head><title>Sending Data Page</title></head><body>
<%
    // Generate and print array
    ArrayList<Integer> ar = new ArrayList<Integer>(20);
    Random generator = new Random();
    out.println("<h2>Created the following array:</h2>");
    for (int i = 0; i < 20; i++)
    {   ar.add(new Integer(generator.nextInt(10)));
        out.println(ar.get(i)+"<br>");
    }
    // Store arraylist in a session variable
    session.setAttribute("arraydata",ar);
%>
</body></html>
```

Passing Objects Between Pages

receivingPage.jsp

```
<%@ page import="java.util.ArrayList" %>
<html><head><title>Receiving Data Page</title> </head><body>

<%
    ArrayList ar = (ArrayList)session.getAttribute("arraydata");
    if (ar == null)
        out.println("<h2>No data sent to page.</h2>");
    else
    {   out.println("<h2>Received the following array:</h2>");
        for (int i = 0; i < 20; i++)
            out.println(ar.get(i)+"<br>");
    }
%>
</body>
</html>
```

Session Question

Question: **True or False:** Data associated with a session remains on the server.

A) true

B) false

Servlets

`Servlets` are Java programs that run inside a web server that can perform server-side processing such as interacting with a database or another application. `Servlets` can generate dynamic html and return this to the client.

How to create a `Servlet`:

- 1) create an HTML file that invokes a `Servlet` (usually through the `FORM ACTION=…`).
- 2) create a Java program that does the following:
  - `import javax.servlet.*;`
  - `import javax.servlet.http.*;`
  - inherit from `HttpServlet`
  - override the `doGet` and `doPost` methods
  - write the response HTML file using `java.io.Printwriter`

Servlets Notes

There is one instance of `Servlet` for each `Servlet` name, and the same instance serves all requests to that name.

Instance members persist across all requests to that name.

Local variables in `doPost` and `doGet` are unique to each request.

Servlets "Hello World!"

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Import Servlet API

```
public class HelloServlet extends HttpServlet {
     public void init(ServletConfig cfg) throws ServletException {
          super.init(cfg);  // First time Servlet is invoked
     }
     public void doGet(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, java.io.IOException {
        doHello(request, response);     }

     public void doPost(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, java.io.IOException {
        doHello(request, response); }

     private void doHello(HttpServletRequest request,HttpServletResponse response)
               throws ServletException, java.io.IOException {
          response.setContentType("text/html");
          java.io.PrintWriter out = response.getWriter();
          out.println("<html><head><title>Hello World</title></head>");
          out.println("<body><h1>Hello World</h1></body></html>");
          out.close();
} }
```

Get & Post

Write out HTML for client

Servlets and JDBC

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class JdbcServlet extends HttpServlet {

    private Connection con;

    public void init(ServletConfig cfg) throws ServletException {
        super.init(cfg);
        String url = "<fill-in>";
        con = null;
        try
        {
            con = DriverManager.getConnection(url);
        }
        catch (SQLException e)
        {   throw new ServletException("SQLException: "+e); }
    }
```

Servlets and JDBC (2)

```
public void destroy() {
    try {
        if (con != null)
            con.close();
    }
    catch (SQLException e)
    {   System.err.println("SQLException: "+e); }
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
    doTable(request, response);
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
    doTable(request, response);
}
```

Servlets and JDBC (3)

```
private void doTable(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, java.io.IOException {
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    out.println("<html><head><title></title></head>");
    if (con == null)
        out.println("<body><h1>Unable to connect to DB</h1></body></html>");
    else
    {   try {
          Statement stmt = con.createStatement();
          ResultSet rst = stmt.executeQuery("SELECT ename,salary FROM emp");
          out.print("<table><tr><th>Name</th><th>Salary</th></tr>");
          while (rst.next())
          { out.println("<tr><td>"+rst.getString(1)+"</td>"
                              +"<td>"+rst.getDouble(2)+"</td></tr>");
          }
          out.println("</table></body></html>");
          out.close();
        } catch (SQLException ex) { out.println(ex); }
    } } }
```

JSP and Servlets Discussion

Java Server Pages (JSPs) are HTML files with embedded Java code. They are easier to produce because the HTML file is the main product and the Java code is secondary.

When a JSP file is actually used, it is converted to a `Servlet` and run. Apache Tomcat handles the conversion and execution of the `Servlet`.

The advantage of JSP over `Servlets` is that the HTML page can be edited by users not familiar with the Java language using standard HTML editors and the Java code added separately by programmers.

JavaScript

JavaScript is a *scripting* language used primarily for web pages.

- JavaScript was developed in 1995 and released in the Netscape web browser (since renamed to Mozilla Firefox).

Despite the name, JavaScript is not related to Java, although its syntax is similar to other languages like C, C++, and Java.

- There are some major differences between JavaScript and Java.

From the database perspective, JavaScript is used to make HTML forms more interactive and to validate input client-side before sending it to the server.

Hello World Example - JavaScript Code

helloWorld.html

```
<html>
<head><title>HelloWorld using JavaScript</title></head>
<body>

<h1>
    <script type="text/javascript">
        document.write("Hello, world!");
    </script>
</h1>

</body>
</html>
```

<script> tag indicating code

JavaScript code

document is HTML document
document.write() puts that text into the document at this location

JavaScript and JSP

Your JSP code can either include JavaScript code:

- 1) Directly in the HTML code
- 2) By outputting it using out.println()
  - With servlets, you only have option #2.

Remember, the JSP/Servlet code is run on the server, and the JavaScript code is run on the client.

- The JavaScript code cannot access the database directly.
- The JSP/Servlet code cannot interact with the user (unless you use AJAX or some other method to send server requests).

Hello World written by JSP Code

helloWorld_JS.jsp

```
<html>
<head><title>HelloWorld using JavaScript</title></head>
<body>

<h1>
    <%
        out.println("<script>");
        out.println("document.write(\"Hello, world!\")");
        out.println("</script>");
    %>
</h1>

</body>
</html>
```

JavaScript for Data Validation

JavaScript is commonly used to validate form input on the client-side without sending it to the server. This reduces the load on the server, and more importantly, improves the browsing experience for the user.

JavaScript Data Validation Example

validateNum.html

```
<html><head><title>Number Field Validation</title></head><body>
<script>
function validate(num){
    if (parseInt(num))
        alert("You entered number: " + parseInt(num));
    else
        alert("Invalid number entered: " + num);
}
</script>

Enter a number:
<form name="input"> <input type="text" name="numField"
                           onchange="validate(this.value)">
</form>
</body></html>
```

Validation code

Validate when field is changed. Can also use:
- `onblur` – triggered when field loses focus
- `onsubmit` – triggered when form is submitted

AJAX

AJAX allows client-side JavaScript to request and receive data from the server without refreshing the page.

AJAX (Asynchronous JavaScript+XML) was named by Jesse Garret in 2005. However, XML is not required for data communication between client and server (JSON is more common).

AJAX uses the **`XMLHttpRequest Object`** to communicate requests and receive results from the server.

Communication can either be synchronous or asynchronous.

AJAX Validate Form Fields Example

validateUserEmail.html

```
<html><head><title>Validate User Name and Email</title></head><body>
<script>
function checkUserName(){
     var name = document.getElementById("username").value;
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.open("GET","validateUserEmail.jsp?name="+name,false);
     xmlhttp.send(null); // Synchronous version, no additional payload
     if (xmlhttp.responseText != "VALID")
          alert(xmlhttp.responseText);
}
function checkEmail(){
     var email = document.getElementById("email").value;
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.open("GET","validateUserEmail.jsp?email="+email,false);
     xmlhttp.send(null); // Synchronous version, no additional payload
     if (xmlhttp.responseText != "VALID")
          alert(xmlhttp.responseText);
}
</script>
<form name="input">
     Name: <input type="text" id="username">
          <a href="javascript: checkUserName()">Check</a> <br>
     Email: <input type="text" id="email">
          <a href="javascript: checkEmail()">Check</a>
</form></body></html>
```

AJAX Validate Form Fields Example

validateUserEmail.jsp

```
<%  String name = request.getParameter("name");
    if (name != null)
    {   // This is simple validation code.
        if (name.length() < 4)
            out.println("Name too short. Must be at least 4 chars.");
        else
            out.println("VALID");
    }
    else
    {
        String email = request.getParameter("email");
        if (email != null)
        {   if (!email.contains("@"))
                out.println("INVALID");
            else
                out.println("VALID");
        }
        else
            out.println("INVALID INPUT");
    }
%>
```

AJAX Example - Province/State

provinceState.html

Canada Version

US Version


```
<html><head><title>Province/State</title></head>
<body bgcolor="white">
<form name="input">
  Country: <select id="country" onchange="changeCountry()">
       <option value="CA">Canada</option>
       <option value="US">United States</option>
    </select><br>
 <p id="stateText">State:</p><select id="state"></select>
</form>
```

Province/State for Canada/US

provinceState.html (2)

```
<script>var xmlhttp = new XMLHttpRequest();
function changeCountry_callBack()
{    if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
     {    var text = xmlhttp.responseText;
          if (text != "INVALID")
          {    var stateList = document.getElementById("state");
               // Remove all current items in list
               for (var i = stateList.length-1; i >= 0; i--)
                 stateList.remove(i);
               // Add elements to list
               var values = text.split(',');
               for (var i=0; i < values.length; i++)
               { var stateCode = values[i];
                 var newOpt = document.createElement('option');
                 newOpt.text = stateCode;
                 newOpt.value = stateCode;
                 stateList.add(newOpt, null);
               }
          }
     }
}
```

Province/State for Canada/US

provinceState.html (3)

```
function changeCountry(){
    var countryList = document.getElementById("country");
    var stateText = document.getElementById("stateText");

    if (countryList.selectedIndex == 1)
        stateText.innerHTML = "State: ";
    else
        stateText.innerHTML = "Province:";

    xmlhttp.open("GET","provinceState.jsp?country="+countryList.value,true);
    xmlhttp.onreadystatechange = changeCountry_callBack;
    xmlhttp.send(null);   // Asynchronous version
}

changeCountry();      // Initialize list with default as Canada
</script>
</body>
</html>
```

Province/State for Canada/US – JSP

provinceState.jsp

```
<%@ page import="java.sql.*" %> <%
String country = request.getParameter("country");
if (country == null)
    out.println("INVALID");
else
{   try
    ( Connection con = DriverManager.getConnection(url,uid,pw); )
    {   String sql = "SELECT stateCode FROM states where countrycode=?";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, country);
        ResultSet rst = pstmt.executeQuery();
        StringBuffer buf = new StringBuffer();
        while (rst.next()) {
            buf.append(rst.getString(1));
            buf.append(',');
        }
      if (buf.length() > 0)
         buf.setLength(buf.length()-1);
      out.println(buf.toString());
    }
    catch (SQLException ex){  out.println("INVALID"); }
    }
%>
```

Connection Pools

A ***connection pool*** is a group of database connections managed by a (web) server.

- All connections in the pool are open. Clients request a connection from the pool and return it when done.
- This results in improved performance as connections are shared across clients and do not pay an open/close penalty every time a connection is used.

Using a connection pool in Tomcat with JNDI:

```
Context root = new InitialContext();                    // Get root of JNDI tree
String path = "java:comp/env/jdbc/workson";             // Name key
DataSource ds = (DataSource) root.lookup(path);         // JNDI lookup
con = ds.getConnection();                               // Get connection from pool
...
con.close();                                            // Return connection to pool
```

Connection Pools Question

Question: **True or False:** A connection pool will speed up the execution time of a query (not considering connection time).

A) true

B) false

Configuring your Web Application

Each web application (*webapp*) has its own directory that stores HTML/JSP files and has a subdirectory `WEB-INF` that contains:

- `classes` directory – stores all Java class files
- `lib` directory – store all jars used by your Servlets
- `web.xml` – is a ***deployment descriptor*** that provides mapping from URL path to Servlet class name. Example:

```
<web-app>
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/servlet/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Internal name used in mapping (you decide)

Servlet Java class name

Internal name (same as line #1)

URL to get to Servlet (from user's perspective)

PHP

PHP (www.php.net) is a general-purpose scripting language used extensively in web development.

PHP supports several different ways of connecting to databases including a custom MySQL connector as well as support for ODBC and PHP Data Objects (PDO).

Unlike JDBC, each database has its own database extension which has different features and methods.

PHP MySQLi Example
Procedural Version

```
<?php
// Connecting, selecting database
$mysqli = mysqli_connect("cosc304.ok.ubc.ca", "rlawrenc",
                         "<pw>", "db_rlawrenc")
          or die("Could not connect: " . mysql_error());

// Performing SQL query
$query = "SELECT * FROM emp";
$result = mysqli_query($mysqli, $query);
if (mysqli_connect_errno($mysqli))
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
```

PHP MySQLi Example Procedural Version (2)

```
// Printing results in HTML
echo "<table>\n";
while ($line = mysqli_fetch_assoc($result)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
// Free resultset
mysqli_free_result($result);
// Closing connection
mysqli_close($mysqli);
?>
```

PHP MySQLi Example
Object-Oriented Version

```
<?php
$mysqli = new mysqli("cosc304.ok.ubc.ca", "<user>", "<pw>", "workson");
if ($mysqli->connect_errno)
  echo "Failed to connect to MySQL: " . $mysqli->connect_error;
$result = $mysqli->query($query);
echo "<table>\n";
while ($line = $result->fetch_assoc()) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
$result->free();
$mysqli->close();
?>
```

Conclusion

JSP and `Servlets` can use JDBC for constructing dynamic web content and answering user queries via HTML.

JSP and `Servlets` are used with HTML forms to allows user to enter queries or information to the database.

- JSP/ `Servlets` can be run using Apache Tomcat which supports connection pooling.

JavaScript is used for browser-based validation and interactivity. AJAX supports requests to server for data.

Connecting and querying a database with PHP (and other languages) is very similar to using Java/JDBC.

Objectives

- Write a simple JSP page that uses JDBC.
- Explain the relationship between JSP and `Servlets` and how dynamic web pages are created.
- Create client-side code in JavaScript for data validation.
- Explain the general idea with AJAX.
- Explain what a connection pool is and why it is beneficial.
- Write database access code using PHP.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA