Entity-Relationship Data Model

SWEN304 / SWEN435

Trimester 1, 2024

Lecturer: Kevin Shedlock

Engineering and Computer Science

Database Design

<u>We will study:</u>

1. Schema Naming Conventions
2. Higher Order Relationship Types
    - Composition
    - Aggregation
    - Generalization
3. Mapping Conceptual into Logical Design
    - Primary Keys
    - Completeness
    - Disjoint
    - Super and Subtypes
    - Inheritance

SWEN304/ Swen435 Course Noticeboard

Tutorials for SWEN304 and SWhEN435.

Course Tutors are available at help desk labs in Room CO246. The days and times are:

- Monday, 2-3pm
- Friday 2-3pm

Review: Extended Entity-Relationship Model

- An Extended Entity-Relationship (EER) model is an enhancement to the original Entity-Relationship (ER) model proposed by Chen.

- An Extended Entity-Relationship provides additional constructs to represent complex relationships and constraints more accurately.

- The Extended Entity-Relationship model is suitable for capturing complex data requirements and constraints in modern database systems.

- Extended Entity Relationships often involve Higher-order relationship types

Review: Extended Entity-Relationship Model

Extensions enrich the power of the ER model by giving database modellers features that add more information to the model. Some of these higher order relationship types include:

- Specialization and Generalization
- Aggregation
- Composition

Review: Higher-order Relationship Types

Higher-order relationship types refer to relationships between entities that involve other participant relationships. These higher-order relationships are complex but necessary to accurately model real-world scenarios

Let's consider the database for TTUPA to depict a higher-order relationship type. The focus is on the relationship between **departments**, **courses**, and **lecturers**.

The example demonstrate how higher-order relationship types can be applied to the TTUPA database that model the complex relationships that may exist amongst the entities.

Higher-order Relationship Types – by Aggregation

Entities: Department, Course, Lecturer

Relationships: A department offers multiple courses and, a lecturer teaches multiple courses.

Higher-order Relationship Type: Aggregation
Each department (e.g., Computer Science) aggregates multiple courses offerings (Introduction to Database Engineering) that it offers. Similarly, a lecturer (Professor Meng) could teach multiple courses. Both departments and instructors are entities, and the relationships they have with courses constitute an aggregation higher-order relationship.

Higher-order Relationship Types by Composition

Entities: Department, Course, Lecturer

Relationships: A department consists of multiple courses and, a lecturer is assigned to teach on courses within a department.

Higher-order Relationship Type: Composition
A department owns the courses it offers. For example, if the department of Mathematics ceases to exist, all its associated courses (e.g., Algebra, Calculus) would also cease to exist. Similarly, a lecturer is assigned to teach courses within a department, and their assignment may be terminated if they leave or are reassigned from that department.

Higher-order Relationship Types by Generalization

Entities: Person, Student, Faculty

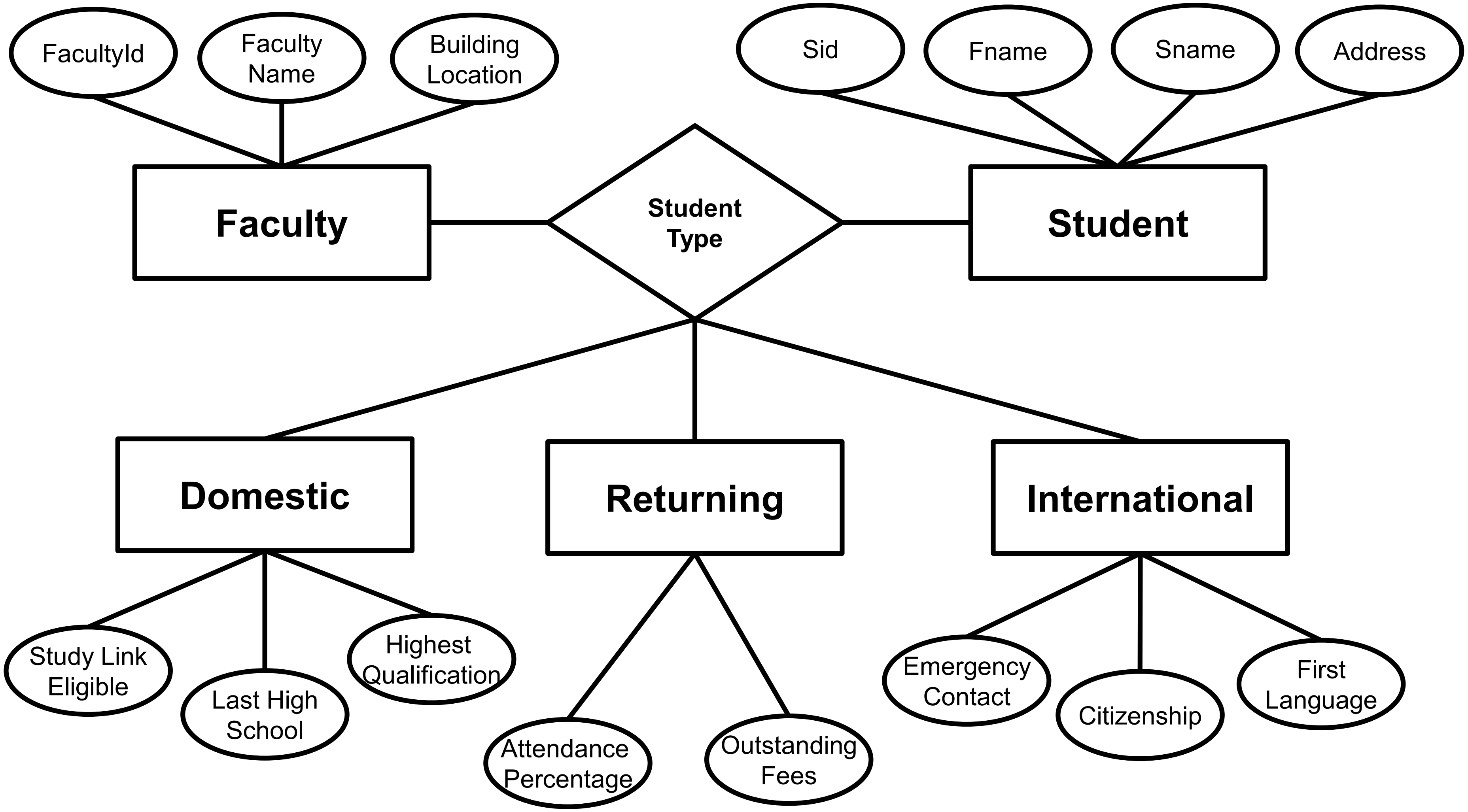
Relationships: Both students and faculty are persons and, both students and faculty may have relationships with courses.

Higher-order Relationship Type: Generalization
Students and faculty are specialized types of persons within a university context. They share common attributes and behaviors with the superclass "Person," such as name, address, and contact information. However, they also have specific attributes and relationships associated with their roles as students or faculty members.

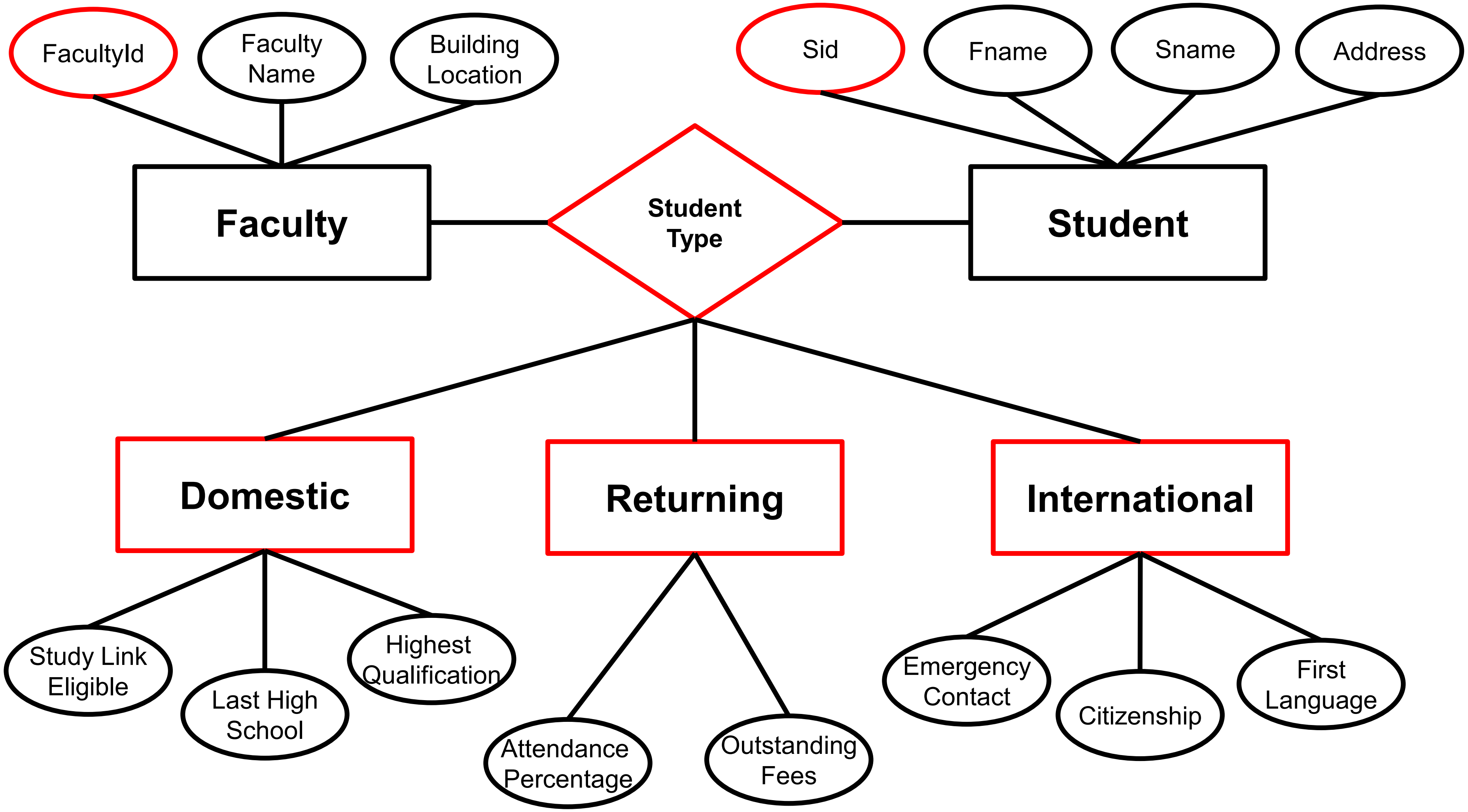
Mapping Conceptual into Logical Design

- Conceptual Design
- Transferring to the logical model
- Applying a set of higher order constraints
  - Assigning Primary and Foreign keys
  - Completeness
  - Disjoint Relationship or Overlapping
  - Applying discriminators to sub-types (checklist)
  - Inheritance

ERD: TTUPA Conceptual Model



ERD: TTUPA Conceptual Model

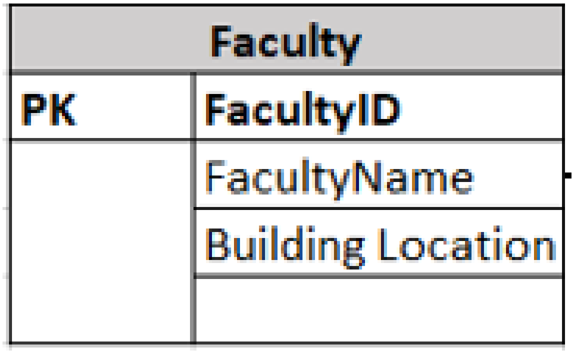


EER Diagram: TTUPA Logical Model

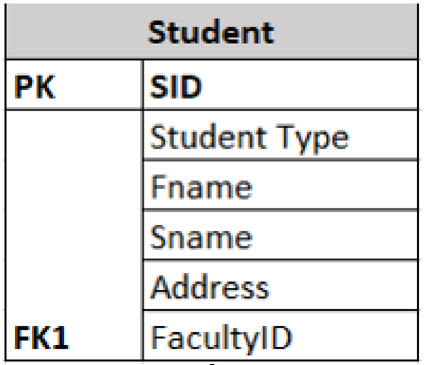
Primary Key and Entity Integrity

- A primary key, also called a primary keyword, is a column in a relational database table that's distinctive for each record.
- It's a unique identifier, such as a student number, index number, department number.
- Examples:
  - STUDENT(Sid, Student Type, Fname, Sname, Address)
  - FACULTY(FacultyId, FacultyName, BuildingLocation)

  (primary key underlined)



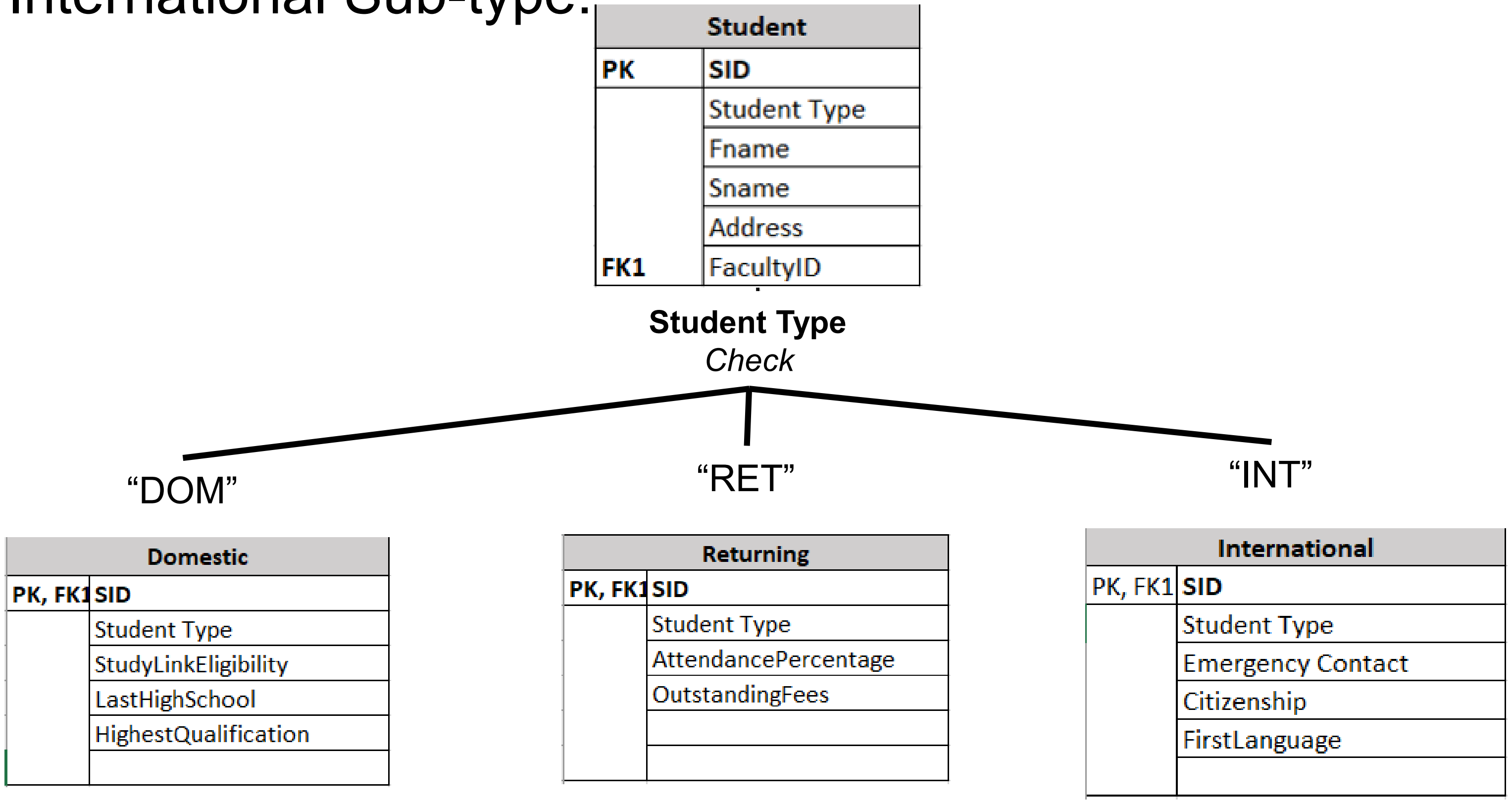
Faculty	
PK	FacultyID
	FacultyName
	Building Location



Student	
PK	SID
	Student Type
	Fname
	Sname
	Address
FK1	FacultyID

EER Diagram: TTUPA Logical Model

Complete: a database constraint that ensures every occurrence in the Super-type has a corresponding occurrence in one of the Sub-types. *Each and every student recorded in the student table* **MUST** *also be represented in either the* Domestic, Returning or the International Sub-type.



Student	
PK	**SID**
	Student Type
	Fname
	Sname
	Address
FK1	FacultyID

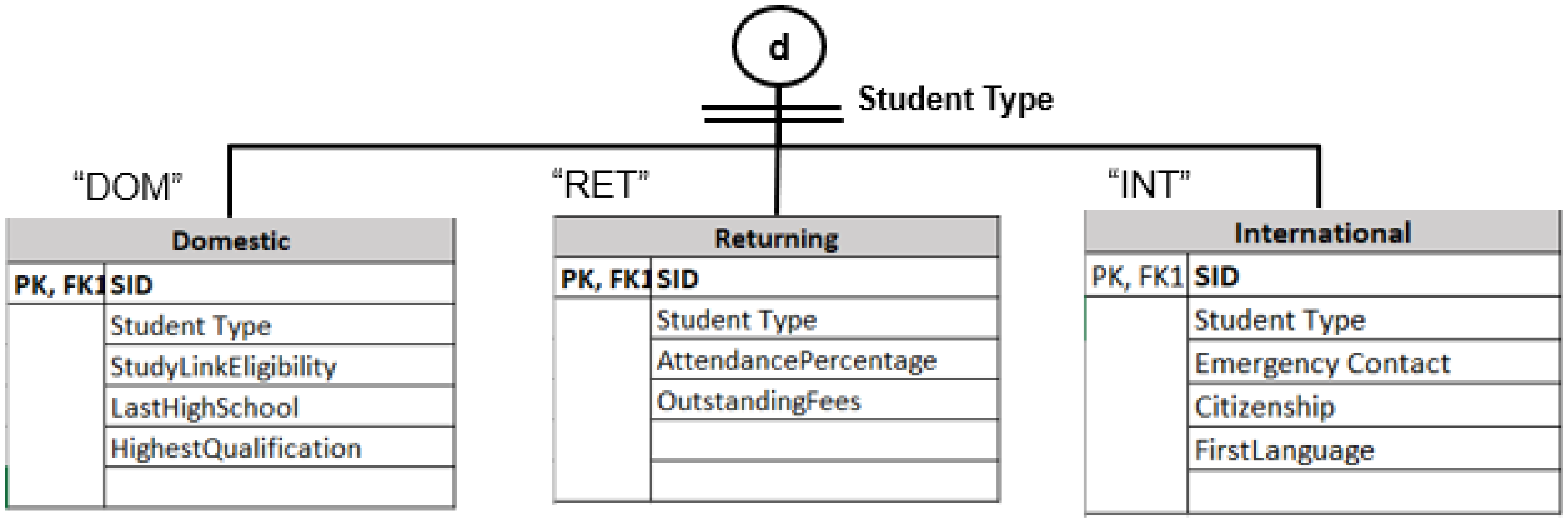
Domestic	
PK, FK1	**SID**
	Student Type
	StudyLinkEligibility
	LastHighSchool
	HighestQualification

Returning	
PK, FK1	**SID**
	Student Type
	AttendancePercentage
	OutstandingFees

International	
PK, FK1	**SID**
	Student Type
	Emergency Contact
	Citizenship
	FirstLanguage

EER Diagram: TTUPA Logical Model

Disjoint: a database constraint that **prevents an occurrence** within one subtype from appearing in another subtype. *Each student record within each type can only belong to one sub-type* e.g. student record appearing in the **Domestic** subtype cannot be recorded in the **International** subtype.



Domestic	
PK, FK1	SID
	Student Type
	StudyLinkEligibility
	LastHighSchool
	HighestQualification

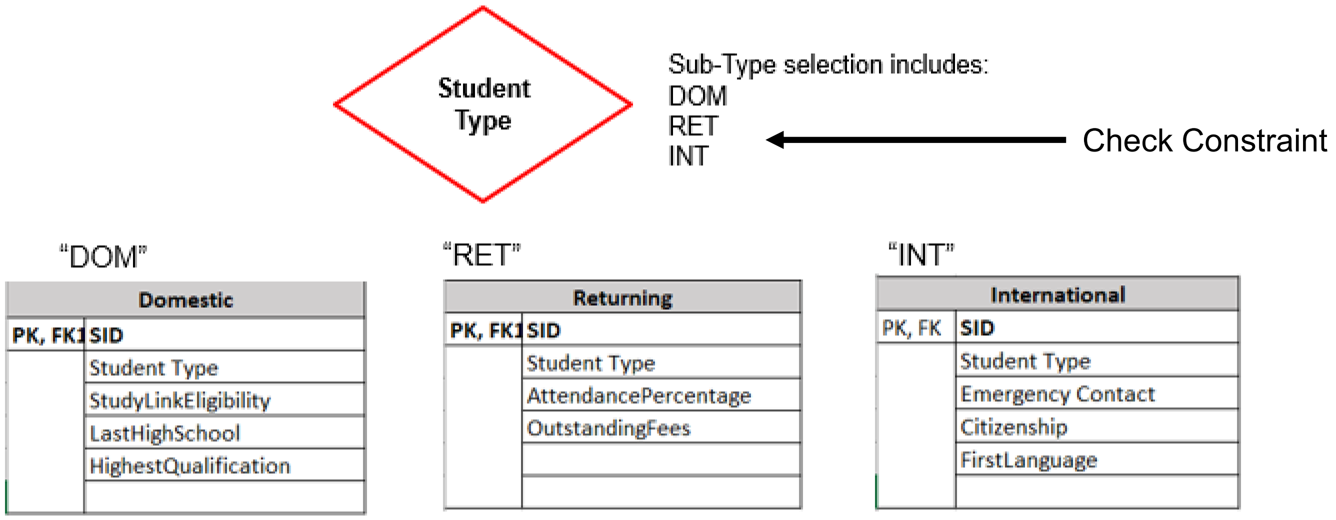
Returning	
PK, FK1	SID
	Student Type
	AttendancePercentage
	OutstandingFees

International	
PK, FK1	SID
	Student Type
	Emergency Contact
	Citizenship
	FirstLanguage

EER Diagram: TTUPA Logical Model

Sub-type Discriminator: a column used to discriminate one type of occurrence from another **Student Type** is used to achieve this task in the above model.

The values taken by the discriminator determines which of the Sub-Type occurrence in the Super type is represented. If an occurrence in the Super-Type has a value of "*RET*" in the subtype discriminator column Supertype, then there will also be a corresponding record for that student in the Returning Type



Domestic	
PK, FK1	SID
	Student Type
	StudyLinkEligibility
	LastHighSchool
	HighestQualification

Returning	
PK, FK1	SID
	Student Type
	AttendancePercentage
	OutstandingFees

International	
PK, FK	SID
	Student Type
	Emergency Contact
	Citizenship
	FirstLanguage

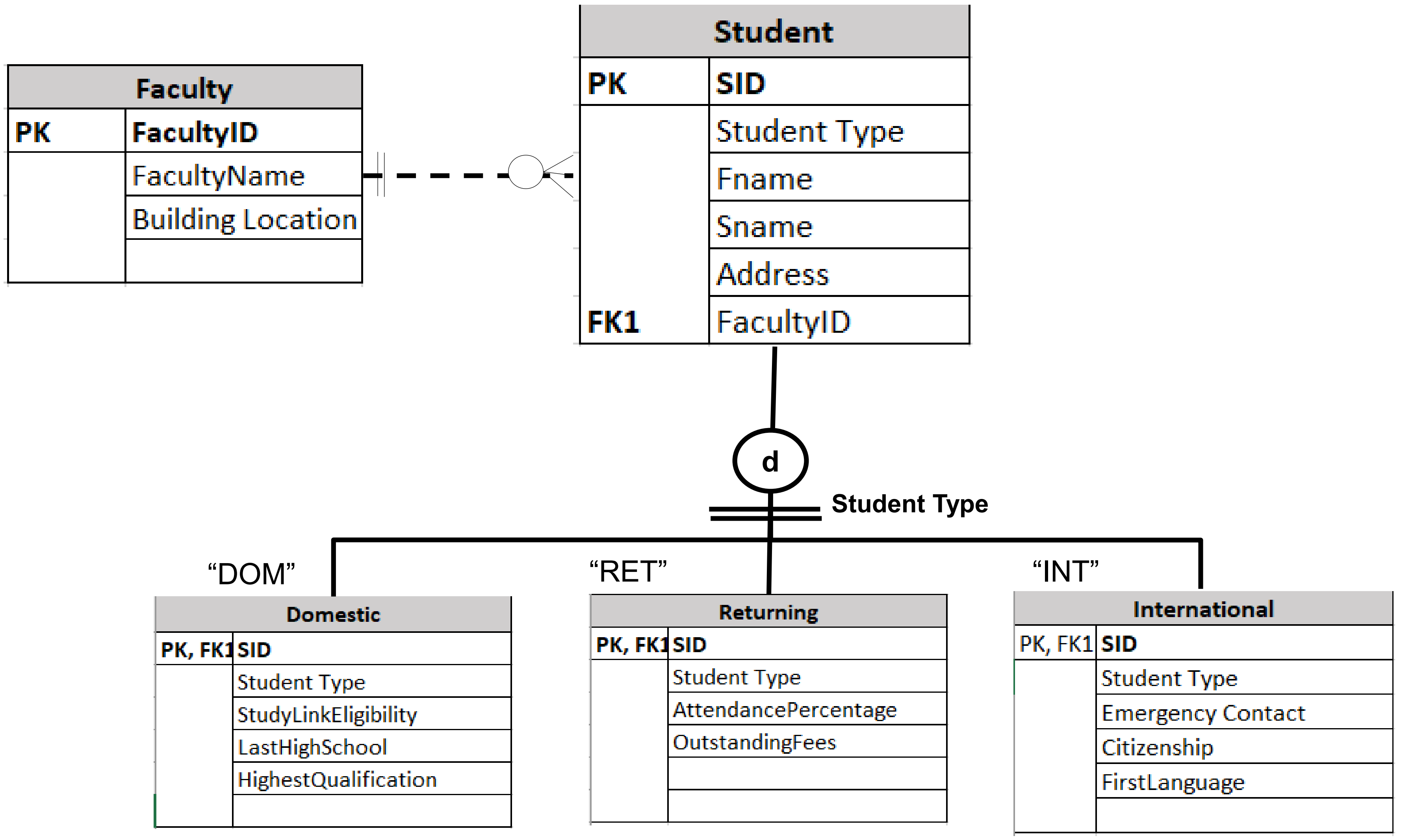
EER Diagram: TTUPA Logical Model

Mapping concepts in logic design As a general rule of interpretation:

- A **disjoint rule** states and entity instance of a super type can only be a member of one sub type. Example, A student opens a bank account (Superclass Account) with two sub-classes Savings Account and Cheque Account. The student can only access an instance for one account at a time.

- A **overlapping rule** states and entity instance of a super type can be a member of many sub types. Example, a Person Superclass within TTUPA can belong to the sub-class as an '*Employee*' or a '*Student*' This relationship would be an overlapping link because a Person can be an Employee or a Student.

EER Diagram: TTUPA Logical Model



EER Diagram: TTUPA Logical Model

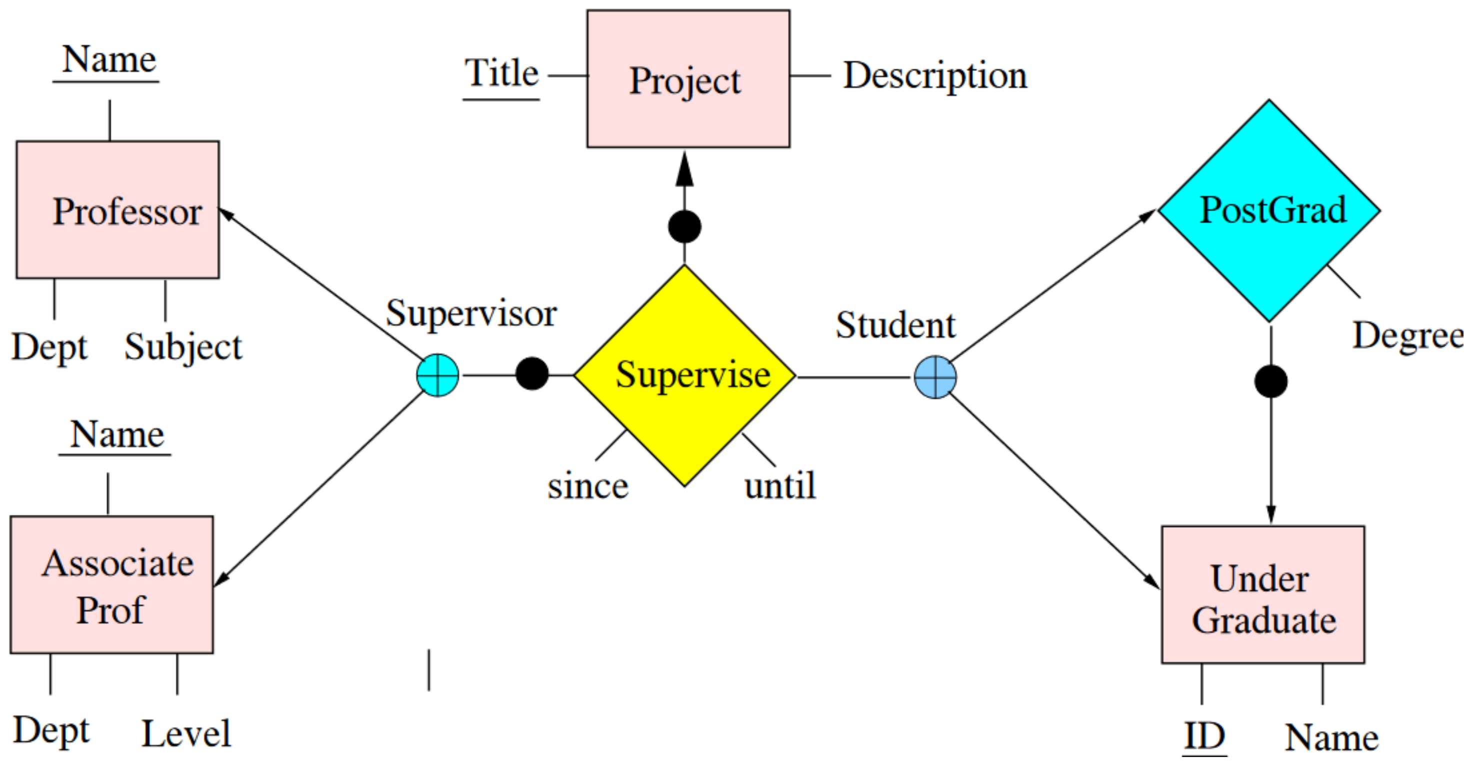
Inheritance: describes the ability of subtypes to inherit properties of the supertype.

Because the cardinality of the relationship between each subtype and its Super-Type is 1:1, then any attributes contained in the Super-Type are implicitly attributes of each of its subtypes. Therefore student contact details recorded in the Student Super-Type are inherited by all of its sub types.

Relationships that the Super-Type engages with are also inherited by the subtypes. In the EER Logical Model, the relationship with Faculty is also a relationship that each super-type is also implicitly engaged.

Wrap Up: EER and Transforming Clusters

- Consider the following EER diagram:



SUPERVISE=({SUPERVISOR, STUDENT, PROJECT},{since, until},{SUPERVISOR, PROJECT}) with

- cluster SUPERVISOR = PROFESSOR $\oplus$ ASSOCIATEPROF
- cluster STUDENT = POSTGRAD $\oplus$ UNDERGRADUATE

Wrap Up: Symbolic Conventions

- An extended entity-relationship diagram (EER diagram) is a directed graph with a node for every entity type or relationship type. By convention symbol are draw as entity types as *rectangles*, draw relationship types as *diamonds*, draw clusters as $\oplus$

- An extended entity-relationship schema (EER schema) is a finite set S of object types (entity types, relationship types, clusters), such that for every object type O in S all its components belong to S, too

- A state/instance S^t of an EER schema S assigns each object type O *as* an object set O^t, so that for each object o in the object set O^t and for every component O' of O the object $o(O')$ belongs to the object set O'^t

Higher-order Relationship Types

A relationship type R is of order 1 if all its components are entity types. *In a (simple) ER diagram all relationship types are of order 1*

R is of order k if its components have maximum order $k - 1$

- This extension is possible in an extended ER diagram
- It gives the designer more flexibility, e.g. to model aggregation

- For convenience, entity relationship types are all called object types
  - Entities and relationships are called objects
  - Entity sets and relationship sets are called object sets
  - Entity types may be regarded as object types of order 0
  - Object types of order k are just relationship types of order k

Examples

Consider schema STUDENT(Sid, Lname, Fname, Adress)

1. Suppose each attribute (e.g. Lname) can have 100 different values
   a) How many different individual records of the STUDENT schema construct can be made?

      e.g. (007007, Bond, James, Wellington), or

      (010101, Wong, Sue, Nelson),

      100^4

   b) How many different student records can be created if we create a constraint that each record must have a unique *Id* value?

      2^(2^4)=2^16

   c) Suppose, instead, each attribute (e.g. Lname) can have only 2 different values, and there is no restriction on *Sid* values

   - How many different sets of records (instances) can be made?

      2^16=65536

Example: EER Schema of the MovieDB

Cluster (order 2):

- Recipient $= \text{Movie} \oplus \text{Crew} \oplus \text{Director} \oplus \text{Writer} \oplus \text{Role}$

Relationship Types (order 2):

- Appearance
  - *components:* Role, Scene
  - *attributes:*
  - *primary key:* Role, Scene

Relationship Types (order 3):

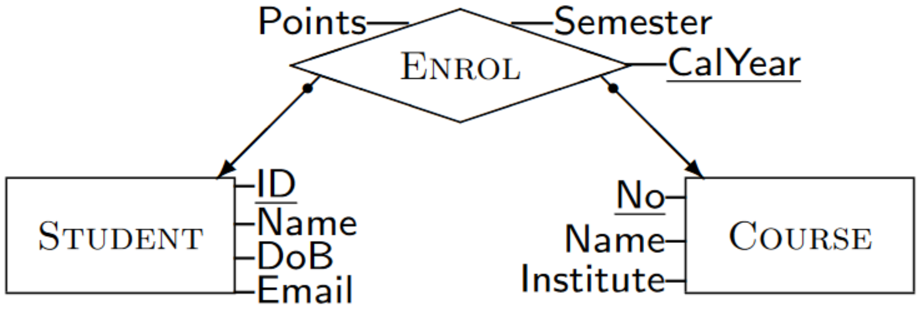
- Awarded
  - *components:* Award, Recipient
  - *attributes:* year, category, result
  - *primary key:* Award, Recipient, year, category

Transformation Entity-Relationship to Relational Data Model

Consider a relationship type $R = (comp(R), attr(R), key(R))$. For each component E of R we choose a set:

$$keyAttr(E) = \{E\_A : \text{where } A \text{ is a primary key attribute of } E^{\text{trans}}\}$$

where E_A are pairwise disjoint new attribute names that do not occur in $attr(R)$ and E^{trans} originates from a prior transformation of component E



Example:

Given a relationship type ENROL with component set {STUDENT, COURSE}, attribute set {CalYear, Semester, Points}, and primary key {STUDENT, COURSE, CalYear}

. . . we choose the following sets:
$keyAttr(\text{STUDENT}) = \{\text{Student\_Id}\}$ and $keyAttr(\text{COURSE}) = \{\text{Course\_No}\}$

Evaluation

The philosophy of a relational data model is based on several key principles that guide the design and implementation of relational databases.

These principles evolve around:

- Simplicity,
- Data Integrity
- Flexibility,
- Normalization
- Data Independence.
- Mathematical Principles

Review: Success Factors in Database Design

Simplicity - The relational model emphasizes simplicity in data representation and manipulation. Data is organized into tables (relations) consisting of rows (tuples) and columns (attributes). This tabular format provides a straightforward way to store and retrieve information, making it easy for users to understand and work with the data.

Review: Success Factors in Database Design

Data Integrity - The relational model promotes data integrity by enforcing constraints and relationships between tables. Constraints such as primary keys, foreign keys, and referential integrity ensure that data remains consistent and accurate. This helps to prevent anomalies and maintain the quality of the data stored in the database.

Review: Success Factors in Database Design

Flexibility - Relational databases offer flexibility in data access and manipulation. Users can perform various operations such as querying, updating, and deleting data using Structured Query Language (SQL).

The relational model provides a powerful and standardized way to interact with the database, allowing users to retrieve specific information based on their requirements.

Review: Success Factors in Database Design

Normalization - One of the fundamental concepts in the relational model is normalization, which involves organizing data into multiple tables to eliminate redundancy and dependency.

By decomposing tables into smaller, more manageable entities and establishing relationships between them, normalization helps to reduce data duplication and improve database efficiency.

Review: Success Factors in Database Design

Data Independence - The relational model promotes data independence, allowing users to separate the logical and physical aspects of the database.

Changes to the database schema or underlying storage structures should not affect the applications that interact with the data. This independence enables easier maintenance, scalability, and portability of the database system.

Review: Success Factors in Database Design

Adherence to Mathematical Principles - The relational model is grounded in mathematical principles, particularly set theory and predicate logic.

This mathematical foundation provides a rigorous framework for defining the structure and operations of relational databases, ensuring consistency and coherence in database design and implementation.