Lecture 2

# Relational model & keys

# Learning Objectives

- Understand relational model fundamentals
- Master different types of keys
- Learn ER modeling techniques
- Apply normalization (1NF-3NF, BCNF)

# What is the Relational Model?

**A data model based on relations (tables) where data is organized in rows and columns**

Key Components:

- Relation (Table): Collection of related data entries
- Tuple (Row): Single data record
- Attribute (Column): Data field with specific meaning
- Domain: Set of valid values for an attribute

# Example: University Database

- Student Table

| StudentID | Name | Email | Major | GPA |
|---|---|---|---|---|
| 12345 | Alice Johnson | alice@edu.com | Computer Science | 3.8 |
| 12346 | Bob Smith | bob@edu.com | Mathematics | 3.6 |
| 12347 | Carol Davis | carol@edu.com | Physics | 3.9 |

# Relational Model Properties

**1. Unique Relation Names**

Each table must have a distinct name in the database

**2. Atomic Attribute Values**

Each cell contains single, indivisible values

❌ Phone: "123-456-7890, 987-654-3210»

✅ Separate PhoneNumber table

**3. No Duplicate Tuples**

Each row must be unique

**4. Order Independence**

Row order doesn't matter: (Alice, Bob, Carol) = (Bob, Carol, Alice)

Column order doesn't matter in logical sense

# Example of Violations

❌ **BAD DESIGN:**

Student(ID, Name, Contacts)

(1, "John", "john@email.com, 555-1234")

✅ **GOOD DESIGN:**

Student(ID, Name)

Contact(StudentID, ContactType, ContactValue)

# Understanding Keys - Superkey

## Superkey: Any combination of attributes that uniquely identifies each tuple

Example:

Student Relation

Student(StudentID, SSN, Email, Name, Major, GPA)

**Superkeys:**

- {StudentID} ← Unique student identifier
- {SSN} ← Social Security Number
- {Email} ← University email is unique
- {StudentID, Name} ← More than needed but still unique
- {SSN, Major} ← More than needed but still unique
- {StudentID, SSN, Email} ← Definitely more than needed!

Key Point: If you can uniquely identify every student using these attributes, it's a superkey!

# Understanding Keys - Candidate Key

## Candidate Key: Minimal superkey (no proper subset is also a superkey)

From Previous Example:

Student(StudentID, SSN, Email, Name, Major, GPA)

**Candidate Keys (minimal superkeys):**

- {StudentID} ✓ Minimal - removing it breaks uniqueness
- {SSN} ✓ Minimal - removing it breaks uniqueness
- {Email} ✓ Minimal - removing it breaks uniqueness

**Not Candidate Keys:**

- {StudentID, Name} ✗ Not minimal
- Name is unnecessary - {SSN, Major} ✗ Not minimal
- Major is unnecessary

# Understanding Keys - Primary & Foreign

**Primary Key: Selected candidate key for the relation**

- Only ONE per relation
- Cannot be NULL
- Should be stable (rarely changes)

**Foreign Key:**

**Attribute that references primary key of another relation**

# Understanding Keys - Primary & Foreign

**Example:**

`Student(StudentID, Name, Email, Major)` – **Primary key**

`Course(CourseID, Title, Credits, Department)` – **Primary key**

`Enrollment(StudentID, CourseID, Semester, Grade)` – **Foreign keys**

`StudentID references Student(StudentID)`

`CourseID references Course(CourseID)`

# Key Constraints in Action

## Real University Database Example:

```
-- Primary Keys ensure uniqueness
Student(StudentID, Name, Email,
Major)
  PK: StudentID

Course(CourseID, Title, Credits,
Department)
  PK: CourseID

Professor(ProfID, Name, Department,
Salary)
  PK: ProfID
```

```
Enrollment(StudentID, CourseID,
Semester, Grade)

Teaching(ProfID, CourseID, Semester,
Classroom)
```

# Key Constraints in Action

## Real University Database Example:

```
-- Primary Keys ensure uniqueness

Student(StudentID, Name, Email,
Major)

  PK: StudentID


Course(CourseID, Title, Credits,
Department)

  PK: CourseID


Professor(ProfID, Name, Department,
Salary)

  PK: ProfID
```

```
-- Composite Primary Key

Enrollment(StudentID, CourseID, Semester,
Grade)

PK: (StudentID, CourseID, Semester)

FK: StudentID → Student(StudentID)

FK: CourseID → Course(CourseID)


Teaching(ProfID, CourseID, Semester,
Classroom)

PK: (ProfID, CourseID, Semester)

FK: ProfID → Professor(ProfID)

FK: CourseID → Course(CourseID)
```

# ER Model Introduction

## Entity-Relationship Model: Conceptual design tool for database structure

Main Components:

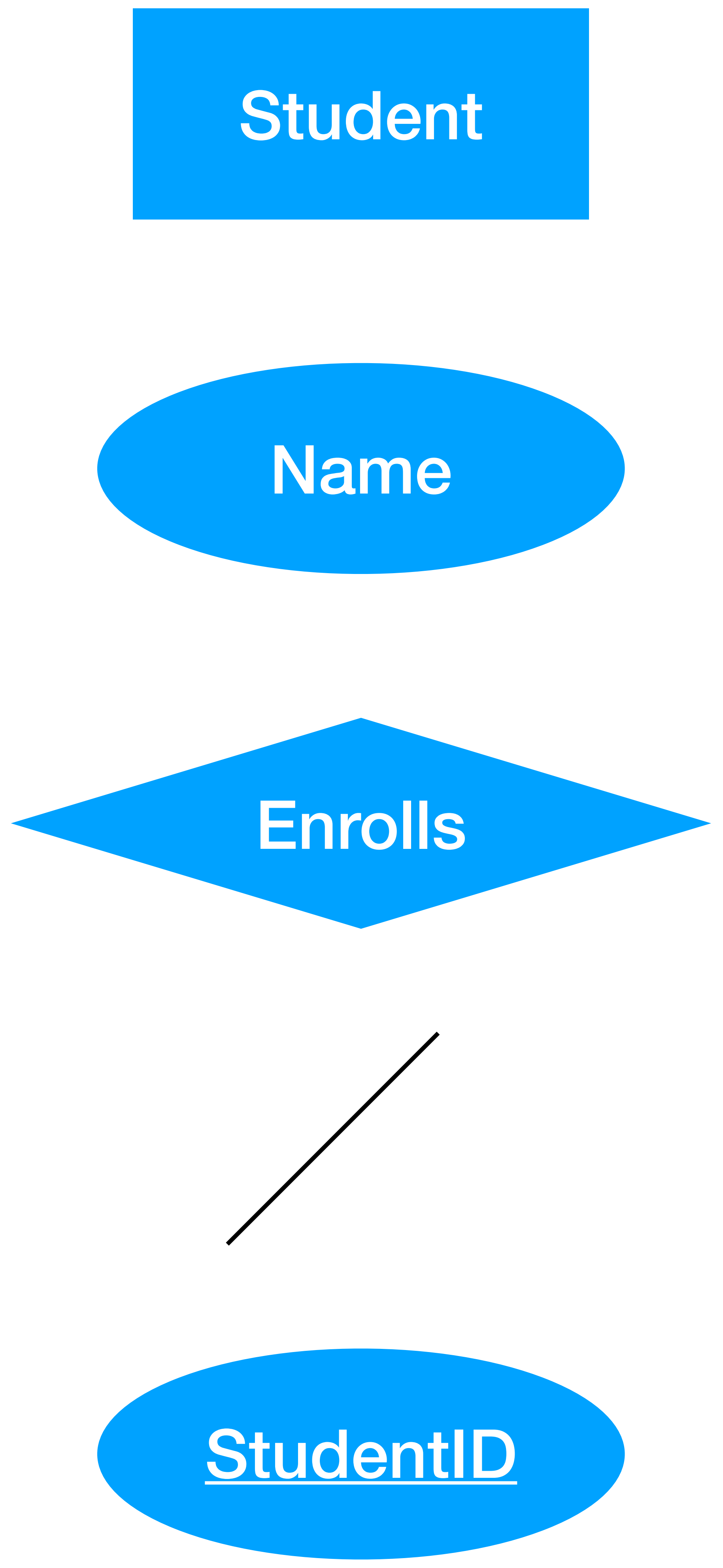
1. Entities - Things in the real world
2. Attributes - Properties of entities
3. Relationships - Associations between entities

Why Use ER Models?

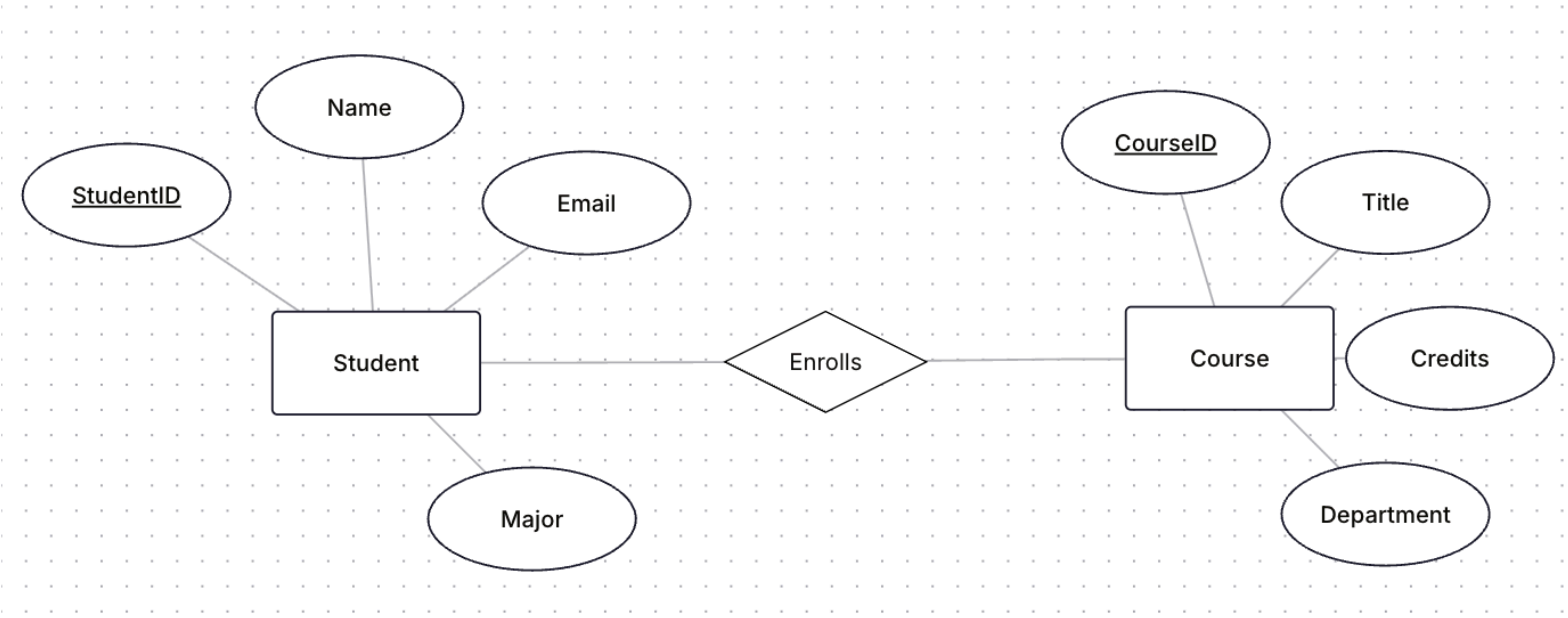
- Visual representation of database structure
- Communication tool between stakeholders
- Foundation for relational database design - Helps identify business rules and constraints

# ER Notation

- Rectangles: Entities
- Ovals: Attributes (double oval = multi-valued, dashed = derived)
- Diamonds: Relationships
- Lines: Connect components
- Underlined: Primary key attributes



# ER Notation


Name
StudentID
Email
Student
Enrolls
Major
CourseID
Title
Course
Credits
Department

# ER Model - Entities

## Entity: Real-world object with independent existence

Types of Entities:

1. **Strong Entity**: Independent existence

Examples: Student, Course, Professor, Department

2. **Weak Entity**: Depends on another entity for identification

Examples:

- Dependent (depends on Employee)
- Room (depends on Building)
- OrderItem (depends on Order)

ER Notation:

Strong Entity: Rectangle


Weak Entity: Double Rectangle

# ER Model - Attributes

## Entity: Real-world object with independent existence

Attribute Types:

1. **Simple Attributes**: Atomic, indivisible Examples: Name, Age, Salary, GPA

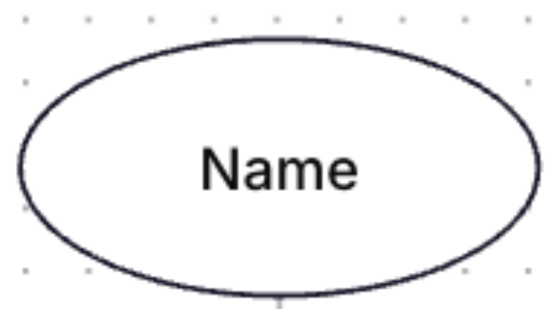
2. **Composite Attributes**: Can be divided into subparts Address → Street, City, State, ZipCode FullName → FirstName, MiddleName, LastName

3. **Multi-valued Attributes**: Multiple values allowed PhoneNumbers: {555-1234, 555-5678, 555-9012} Languages: {English, Spanish, French}

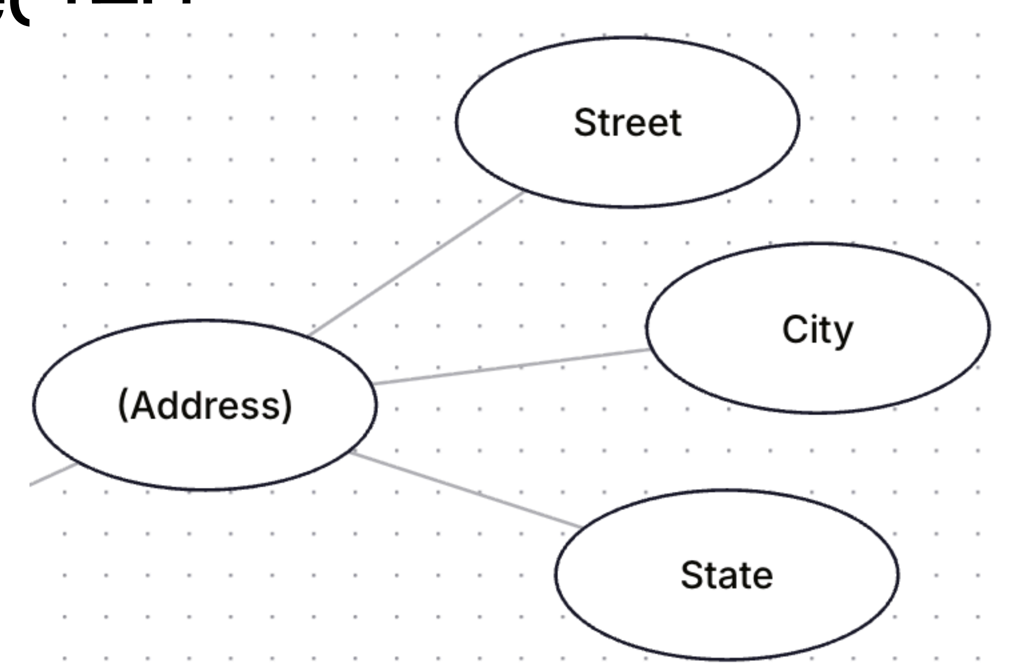
4. **Derived Attributes**: Calculated from other attributes Age (derived from BirthDate and current date) TotalSalary (derived from BaseSalary + Bonus)

ER Notation:

Simple: Oval




Composite: Oval with connected sub-ovals



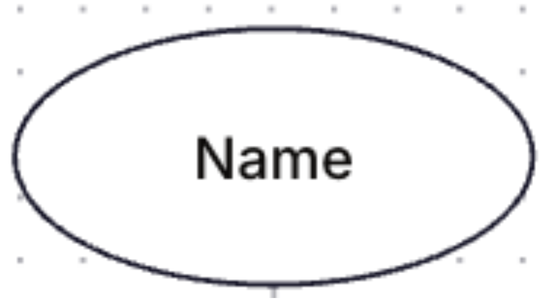

Multi-valued: Double Oval

Derived: Dashed Oval

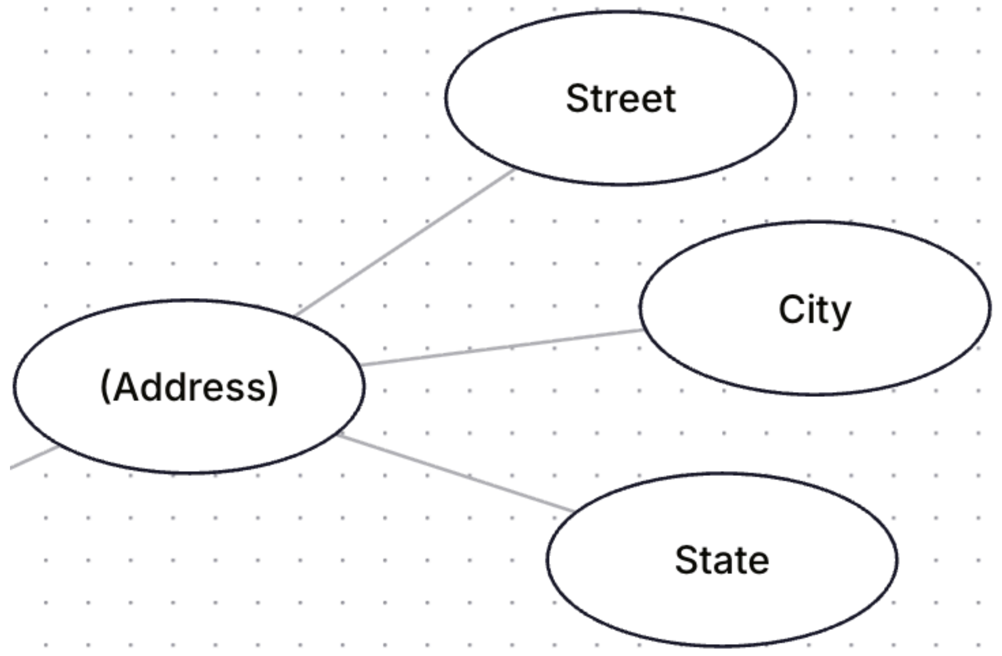
# ER Model - Attributes

**ER Notation:**

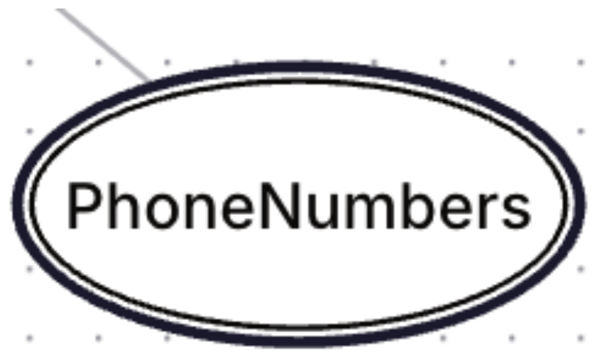
Simple: Oval



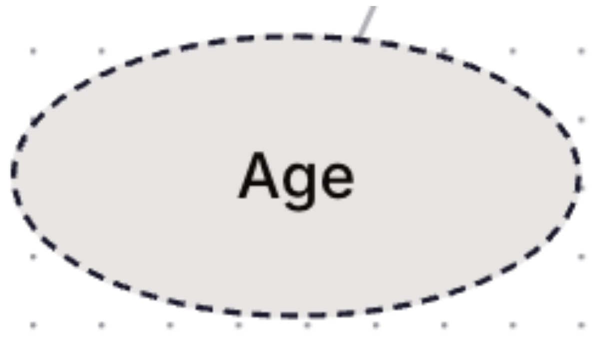

Composite: Oval with connected sub-ovals




Multi-valued: Double Oval




Derived: Dashed Oval



# ER Model - Relationships

## Relationship: Association between entities

**Types by Degree:**

- Binary: Between 2 entities (most common)

- Ternary: Between 3 entities (Student TAKES Course FROM Professor)

- n-ary: Between n entities

**Cardinality Constraints:**

- 1:1 (One-to-One)

- 1:N (One-to-Many)

- M:N (Many-to-Many)

```
Student ──── ENROLLS ──── Course
(0,N)                   (0,N)
↑                       ↑
min,max participation

Professor ──── TEACHES ──── Course
(1,N)                     (0,1)

Department ──── HAS ──── Professor
(1,1)                 (5,50)
```

# Normalization

# Normalization - Why Do We Need It?

## Problems with Unnormalized Tables:

Example: StudentCourseInfo Table

| StudentID | Name | Major | CourseID | Title | Credits | Grade |
|---|---|---|---|---|---|---|
| 1001 | John | Computer Science | CS101 | Intro | 3 | A |
| 1001 | John | Computer Science | CS102 | OOP | 4 | B |
| 1002 | Mary | Mathematics | CS101 | Intro | 3 | A |
| 1002 | Mary | Mathematics | MATH201 | Calculus | 4 | A |

Problems:

1. **Redundancy**: John's info repeated for each course
2. **Update Anomaly**: Change John's major → update multiple rows
3. **Insert Anomaly**: Can't add new student without enrollment
4. **Delete Anomaly**: Delete last enrollment → lose student info

# First Normal Form (1NF)

## Rule: All attributes must contain atomic (indivisible) values

Violation Examples:

❌ Multi-valued Attributes:

`Student(ID, Name, PhoneNumbers, Skills)`

`(1, "John", "555-1234, 555-5678", "Java, Python, SQL")`

❌ Composite Attributes:

`Employee(ID, Name, Address, Department)`

`(101, "Alice", "123 Main St, Boston, MA 02101", "CS")`

1NF Solutions:

✅ Separate Multi-valued:

`Student(ID, Name)`

`StudentPhone(StudentID, PhoneNumber)`

`StudentSkill(StudentID, Skill)`

✅ Break Composite:

`Employee(ID, Name, Street, City, State, Zip, Department)`

# Second Normal Form (2NF)

## Rule: Must be in 1NF + No partial dependencies

## Partial Dependency: Non-key attribute depends on part of composite primary key

Violation Example:

❌ `Enrollment(StudentID, CourseID,` **`StudentName, CourseName`**`, Grade)`

`Primary Key: (StudentID, CourseID)`

Partial Dependencies:

- StudentName depends only on StudentID
- CourseName depends only on CourseID
- Grade depends on full key ✓

2NF Solution:

✅ `Student(StudentID, StudentName)`

`Course(CourseID, CourseName)`

`Enrollment(StudentID, CourseID, Grade)`

**Key Point: If primary key is single attribute, you automatically satisfy 2NF!**

# Third Normal Form (3NF)

## Rule: Must be in 2NF + No transitive dependencies

### Transitive Dependency: A → B → C (where A is key, C is non-prime)

Violation Example:

❌ `Employee(EmpID, Name, Department, DeptHead, DeptBudget)`

Dependencies:

`EmpID → Department (direct dependency ✓)`

`Department → DeptHead (transitive dependency ❌)`

`Department → DeptBudget (transitive dependency ❌)`

**`Chain: EmpID → Department → DeptHead`**

3NF Solution:

✅ `Employee(EmpID, Name, Department)`
`Department(Department, DeptHead, DeptBudget)`

Another Example:

❌ `Order(OrderID, CustomerID, CustomerName, CustomerCity)`

`OrderID → CustomerID → CustomerName`

`OrderID → CustomerID → CustomerCity`

✅ `Order(OrderID, CustomerID)`
`Customer(CustomerID, CustomerName, CustomerCity)`

# Boyce-Codd Normal Form (BCNF)

## Rule: For every functional dependency A → B, A must be a superkey

### When 3NF ≠ BCNF: Overlapping candidate keys

**`StudentAdvisor(StudentID, Major, Advisor)`**

Business Rules:

- Each student has one major
- Each advisor works in only one major -
- Students can have multiple advisors in their major

Functional Dependencies:

- (StudentID, Advisor) → Major
- (StudentID, Major) → Advisor
- Advisor → Major ← This violates BCNF!

Candidate Keys: {StudentID, Advisor}, {StudentID, Major}

BCNF Solution:

```
StudentMajor(StudentID, Major)
AdvisorMajor(Advisor, Major)
StudentAdvisor(StudentID, Advisor)
```

# Normalization Practice - Library System

**Given Unnormalized Table:**

```
LibraryRecord(BookID, Title, AuthorName,
AuthorCountry, PublisherName, PublisherCity,
StudentID, StudentName, StudentMajor,
CheckoutDate, DueDate, FineAmount)
```

**Sample Data:**

```
(B1, "Database Systems", "Elmasri", "USA",
"Pearson", "Boston", S1, "John", "CS",
"2024-01-15", "2024-02-15", 5.00)
```

Step 1: Identify Functional Dependencies

```
BookID → Title, AuthorName,
AuthorCountry, PublisherName,
PublisherCity
```

```
StudentID → StudentName, StudentMajor
```

```
(StudentID, BookID, CheckoutDate) →
DueDate, FineAmount
```

```
AuthorName → AuthorCountry (assuming
each author from one country)
```

```
PublisherName → PublisherCity
```

# Normalization Practice - Solution

**2NF Decomposition:**

```
BookInfo(BookID, Title, AuthorName, AuthorCountry, PublisherName, PublisherCity)

StudentInfo(StudentID, StudentName, StudentMajor)

CheckoutRecord(StudentID, BookID, CheckoutDate, DueDate, FineAmount)
```

**3NF Decomposition:**

```
Book(BookID, Title, AuthorName, PublisherName)

Author(AuthorName, AuthorCountry)

Publisher(PublisherName, PublisherCity)

Student(StudentID, StudentName, StudentMajor)

Checkout(StudentID, BookID, CheckoutDate, DueDate, FineAmount)
```

# Summary & Key Takeaways

**Relational Model:**

- Tables (relations) with rows (tuples) and columns (attributes)
- Various types of keys ensure data integrity

**ER Modeling:**

- Visual design tool for database structure
- Entities, attributes, and relationships
- Foundation for relational database design

Normalization Benefits:

- 1NF: Atomic values
- 2NF: No partial dependencies
- 3NF: No transitive dependencies
- BCNF: All dependencies from superkeys

Key Skills Developed:

✓ Identify different types of keys

✓ Create ER diagrams from requirements

✓ Map ER diagrams to relational schemas

✓ Apply normalization techniques

✓ Recognize and fix data anomalies