CS 61:
Database Systems

ER models

Adapted from Silberschatz, Korth, and Sundarshan unless otherwise noted
Augmented by Claude Opus

Today we begin to focus on developing our own database schemas

Database design goals:

- Well structured tables with minimal redundancy
- Store data facts one time

Pierson's simplified two-step view of database design:

Today 1. Identify entities, attributes, and relations -> leads to database tables and keys

Next class 2. Normalize tables -> check that your tables are well designed to prevent data anomalies (store data once!)

(There is also physical database design (e.g., how are database files organized), but we will discuss that soon)

Done well, Step 1 makes Step 2 easy!

Agenda

1. Entity Relationship (ER) models
2. Relationships
3. How to build an ER model
4. Reverse and forward engineering

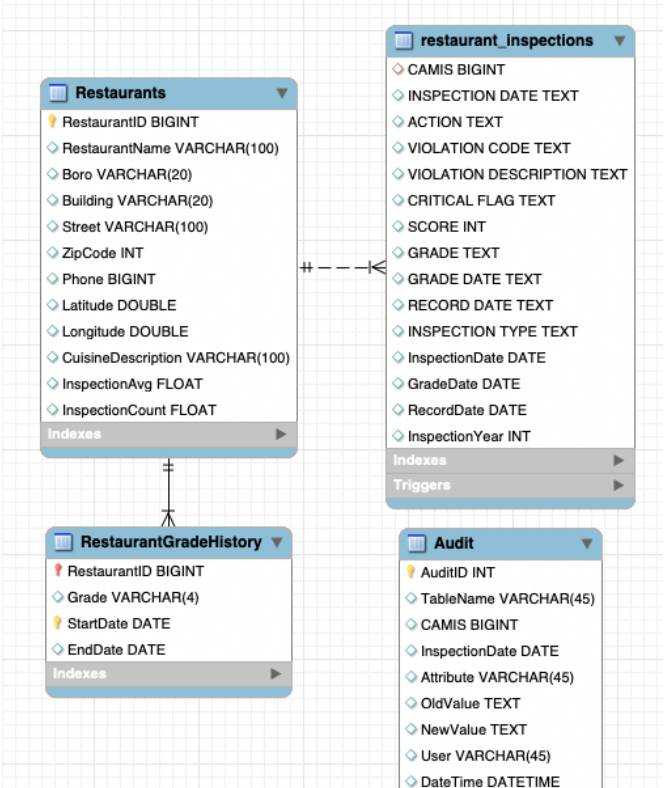
ER models use three basic concepts: Entities, Relationships, and Attributes

Entity Relationship (ER) models

ER model (ERM) uses three basic concepts:

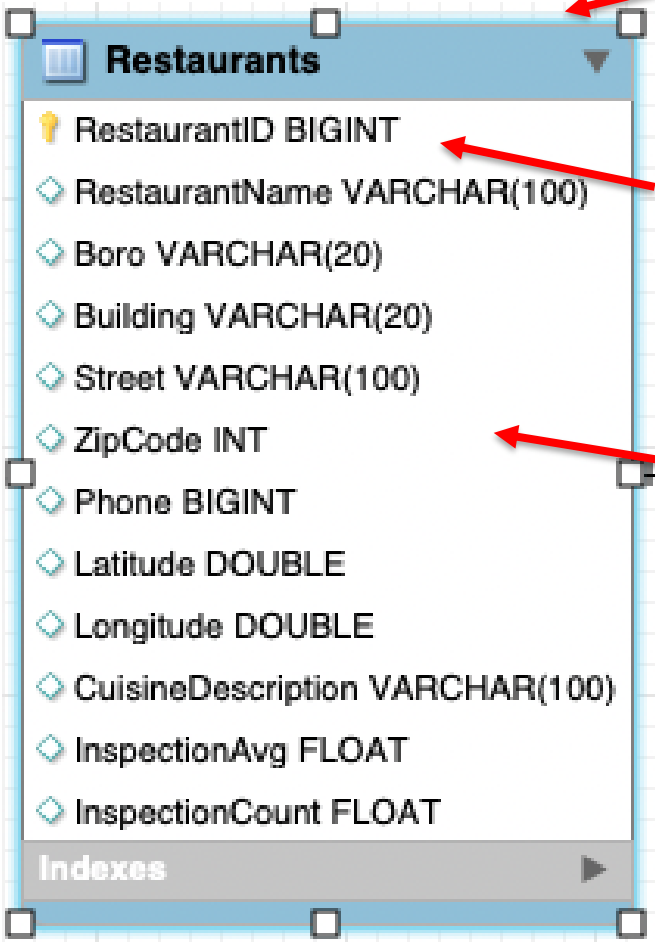
1. Entities: what are the nouns involved?
2. Relationships: how are the entities related
3. Attributes: what characteristics do entities have?

ER diagram (ERD)
expresses the overall model graphically



Entities are nouns, each represents people, places, things, concepts, or events

Entity Relationship Diagram (ERD)



Entities are represented as rectangles
Entity set is set of entity instances
Entity set is materialized as a table

Primary key uniquely identifies entity instance
Can be composite key (made up of several attributes)

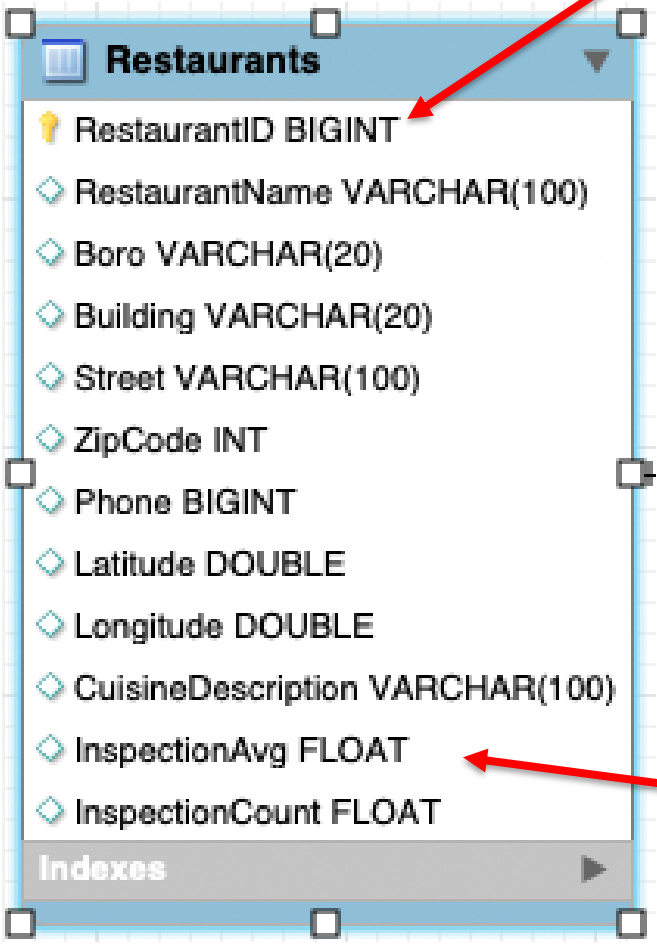
Entities are made up of attributes

Avoid storing same information in multiple tables (avoid data redundancy) unless:

1. Need speed: joining multiple tables is too slow for business need
2. Historical documentation: want to store the state at the time of a transaction (e.g., what was the price of an item when it was sold, store current price and also store price when sold)

Attributes describe an entity and have data type

Entity Relationship Diagram (ERD)



Attribute name and data type

MySQL does not support composite attributes

If FullName is composite of

- **First name**
- **Last name**

Just promote all composite components to simple attributes

How can we automatically compute full name from first and last names?

Use generated column

FullName VARCHAR(100) GENERATED ALWAYS AS CONCAT(LastName, ', ', FirstName) VIRTUAL

Some attributes can be derived from other attributes (possibly in other tables as we did using triggers in last class)

Value of derived attributes can be stored or computed on demand

Entity Relationship Diagram (ERD)

Derived attribute: store value or compute on demand

	Store computed value	Compute on demand
Advantages	• Faster read • Can be used to keep track of historical data	• Faster write • Less space • Computation always yields current value
Disadvantages	• Slower write • Requires constant maintenance keep value current	• Slower read • Adds coding complexity to queries

Entities can be strong or weak

Strong entity

Exists **independently** and has its own primary key that uniquely identifies each instance

Characteristics

- Exists on its own – does not depend on another entity
- Has a primary key made from its own attributes

Examples:

Entity	Primary Key
Student	StudentID
Department	DeptID
Product	ProductID
Customer	CustomerID

Weak entity

Cannot exist without a related strong entity (called the owner/parent entity). It does not have a sufficient primary key on its own

Characteristics

- Depends on a strong (owner) entity for its existence
- Has a partial key (also called a discriminator) — an attribute that only uniquely identifies it *within the context* of its owner entity
- Its full unique identification = Owner's Primary Key + Partial Key

Examples:

Entity	Primary Key
Building room	Building ID, Room Number
Product order	Order ID, Item ID

Example: Hotel (strong) and hotel room (weak)

```
CREATE TABLE Hotels (
    HotelID INT PRIMARY KEY AUTO_INCREMENT,
    HotelName VARCHAR(100) NOT NULL,
    Location VARCHAR(200)
);
 -- Insert hotels
 INSERT INTO Hotels (HotelID, HotelName, Location)
 VALUES
     (1, 'Grand Plaza', 'New York'),
     (2, 'Ocean View', 'Miami');
```

Hotels stand on their own

Example: Hotel (strong) and hotel room (weak)

```
CREATE TABLE Hotels (
   HotelID INT PRIMARY KEY AUTO_INCREMENT,
   HotelName VARCHAR(100) NOT NULL,
   Location VARCHAR(200)
);
 -- Insert hotels
 INSERT INTO Hotels (HotelID, HotelName, Location)
 VALUES
    (1, 'Grand Plaza', 'New York'),
    (2, 'Ocean View', 'Miami');
```

Hotels stand on their own

```
CREATE TABLE Rooms (
   HotelID INT,
   RoomNumber INT,
   RoomType VARCHAR(50),
   Price DECIMAL(10,2),
   PRIMARY KEY (HotelID, RoomNumber),
   FOREIGN KEY (HotelID)
       REFERENCES Hotel(HotelID)
     ON DELETE CASCADE
);
```

Rooms can't exist without a hotel (weak entity)

```
-- Insert rooms
INSERT INTO Rooms (HotelID, RoomNumber, RoomType, Price)
VALUES
   (1, 101, 'Suite', 299.99),
   (1, 102, 'Standard', 149.99),
   (2, 101, 'Deluxe', 199.99),    -- Same room #s, different hotel!
   (2, 102, 'Standard', 129.99);
```

Weak entity characteristics

```
CREATE TABLE Hotels (
   HotelID INT PRIMARY KEY AUTO_INCREMENT,
   HotelName VARCHAR(100) NOT NULL,
   Location VARCHAR(200)
);
 -- Insert hotels
 INSERT INTO Hotels (HotelID, HotelName, Location)
 VALUES
    (1, 'Grand Plaza', 'New York'),
    (2, 'Ocean View', 'Miami');
```

Compound primary key consisting of strong entity PK and partial key (cannot exist on its own)

```
CREATE TABLE Rooms (
   HotelID INT,
   RoomNumber INT,
   RoomType VARCHAR(50),
   Price DECIMAL(10,2),
   PRIMARY KEY (HotelID, RoomNumber),
   FOREIGN KEY (HotelID)
      REFERENCES Hotel(HotelID)
    ON DELETE CASCADE
);
```

```
-- Insert rooms
INSERT INTO Rooms (HotelID, RoomNumber, RoomType, Price)
VALUES
   (1, 101, 'Suite', 299.99),
   (1, 102, 'Standard', 149.99),
   (2, 101, 'Deluxe', 199.99),    -- Same room #s, different hotel!
   (2, 102, 'Standard', 129.99);
```

Foreign key constraint that deletes rows from weak entity if strong entity disappears

Theory: weak entities use composite keys

```
CREATE TABLE Hotels (
   HotelID INT PRIMARY KEY AUTO_INCREMENT,
   HotelName VARCHAR(100) NOT NULL,
   Location VARCHAR(200)
);
 -- Insert hotels
 INSERT INTO Hotels (HotelID, HotelName, Location)
 VALUES
    (1, 'Grand Plaza', 'New York'),
    (2, 'Ocean View', 'Miami');
```

Theory

Weak entities use composite PK with strong entity PK

No surrogate keys on weak entities

Follows formal ER modeling rules

```
CREATE TABLE Rooms (
   HotelID INT,
   RoomNumber INT,
   RoomType VARCHAR(50),
   Price DECIMAL(10,2),
   PRIMARY KEY (HotelID, RoomNumber),
   FOREIGN KEY (HotelID)
      REFERENCES Hotel(HotelID)
     ON DELETE CASCADE
);
```

```
-- Insert rooms
INSERT INTO Rooms (HotelID, RoomNumber, RoomType, Price)
VALUES
   (1, 101, 'Suite', 299.99),
   (1, 102, 'Standard', 149.99),
   (2, 101, 'Deluxe', 199.99),    -- Same room #s, different hotel!
   (2, 102, 'Standard', 129.99);
```

Practice: Everything gets a surrogate key

```
CREATE TABLE Hotels (
   HotelID INT PRIMARY KEY AUTO_INCREMENT,
   HotelName VARCHAR(100) NOT NULL,
   Location VARCHAR(200)
);
 -- Insert hotels
 INSERT INTO Hotels (HotelID, HotelName, Location)
 VALUES
    (1, 'Grand Plaza', 'New York'),
    (2, 'Ocean View', 'Miami');

CREATE TABLE Rooms (
   HotelID INT,
   RoomNumber INT,
   RoomType VARCHAR(50),
   Price DECIMAL(10,2),
   PRIMARY KEY (HotelID, RoomNumber),
   FOREIGN KEY (HotelID)
       REFERENCES Hotel(HotelID)
     ON DELETE CASCADE
);
```

Practice

Almost EVERYTHING gets surrogate key

Surrogate keys on ALL tables (strong and weak)

Natural/composite keys enforced with UNIQUE

```
CREATE TABLE Rooms (
   RoomID INT AUTO_INCREMENT,
   HotelID INT,
   RoomNumber INT,
   RoomType VARCHAR(50),
   Price DECIMAL(10,2),
   PRIMARY KEY (RoomID),
   FOREIGN KEY (HotelID)
       REFERENCES Hotel(HotelID)
     ON DELETE CASCADE
);
```

Claude's guidance

Are you...

 In a database theory class?

→ Use composite keys, no surrogates on weak entities

→ Follow the textbook rules

 Building a real application?

→ Surrogate keys on everything

→ UNIQUE constraints on natural keys

→ ON DELETE CASCADE where needed

 On an exam?

→ Ask your professor!

→ Most will expect composite keys for weak entities

Agenda

1. Entity Relationship (ER) models

2. Relationships
   - One-to-many (1:M)
   - One-to-one (1:1)
   - Many-to-many (M:N)

3. How to build an ER model

4. Reverse and forward engineering

Examples of 1:Many relationships

Coach ⇔ Players

- Each coach has many players on their team, each player has one coach

Department ⇔ Employees

- Each department has many employees, each employee belongs to one department

Landlord ⇔ Properties

- Each landlord may have many properties, each property belongs to one landlord

Ask:

Can A have multiple B's? **Yes**

Can B belong to multiple A's? **No**

If both answers match: **1:M**

(If the answer to **both** is **Yes**, then it's likely a **M:M** relationship instead)

Handle:

Add a FK column in the many side table

One-to-many relationships are the most common

One-to-many (1:M)

Many side

One side

Notice how links "fan out"

Date: Jan 2, 2026
Score: 7
Grade: A

Date: Feb 4, 2026
Score: 15
Grade: B

Date: Apr 20, 2026
Score: 4
Grade: A

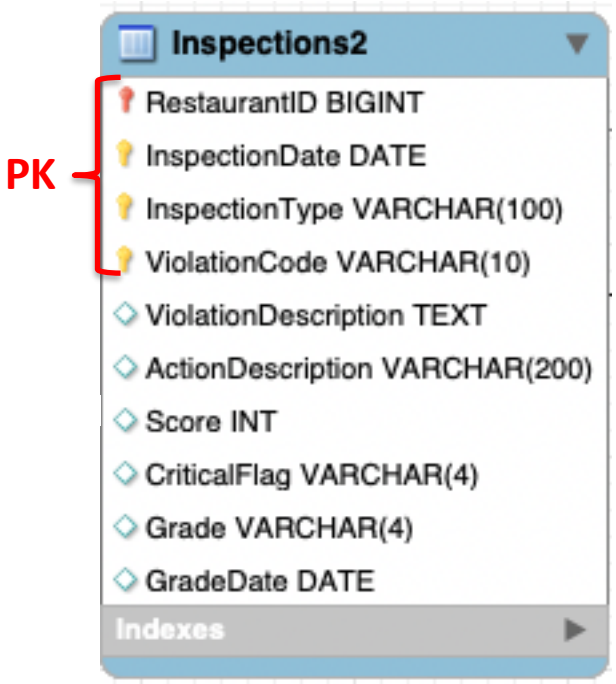
One restaurant can be inspected many times

One inspection is for one restaurant

There is a 1:M relationship between restaurants and inspections

Can Restaurants (A) have multiple Inspections (B)? **Yes**
Can Inspections (B) belong to multiple Restaurants (A)? **No**

We can identify each row in restaurant_inspections with four attributes



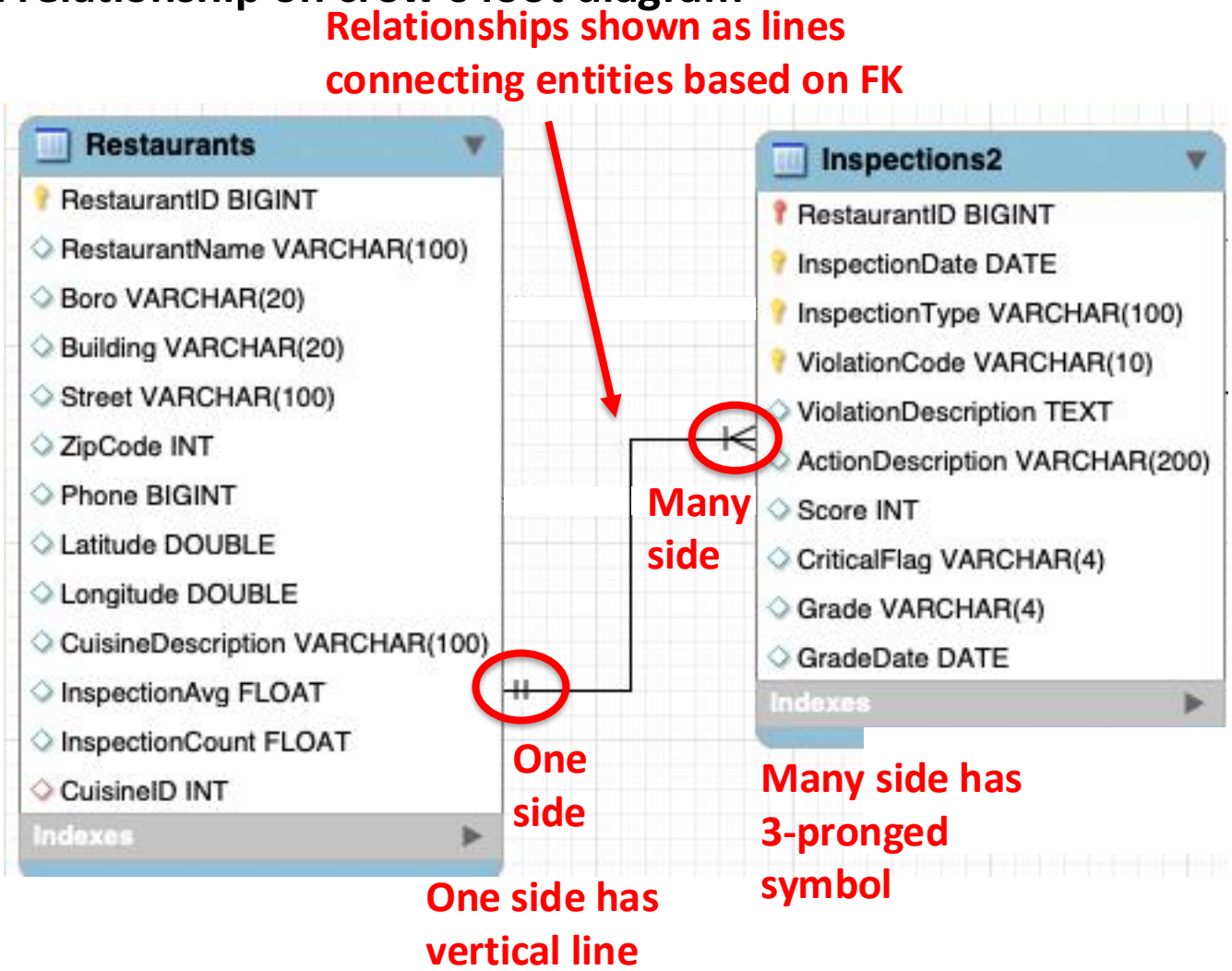
Each row in restaurant_inspections can be identified by

1. **RestaurantID**
2. **InspectionDate**
3. **InspectionType (e.g., initial, re-opening)**
4. **ViolationCode (e.g., 10A)**

These four attributes *could* be used to create a compound primary key

Crow's foot diagram shows one-to-many using a 3-pronged symbol on the many side

1:M relationship on crow's foot diagram



Relationship based on RestaurantID

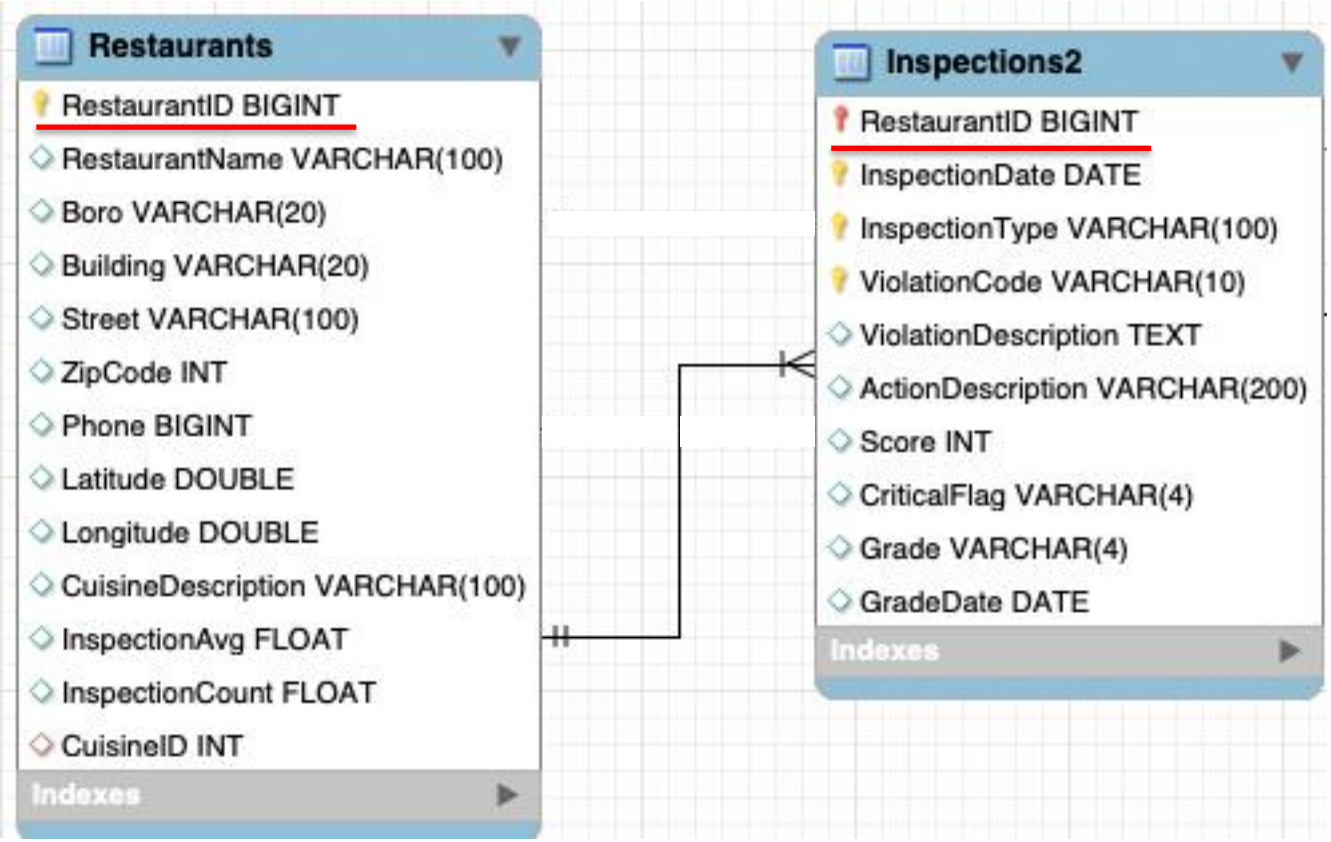
RestaurantID FK on Inspections2 constraint relates an Inspection to a Restaurant

One Restaurant can be inspected many times (1:M)

Decide:
1:M FK not unique
1:1 FK unique

Solid line indicates a strong (identifying) relationship between entities

1:M relationship on crow's foot diagram



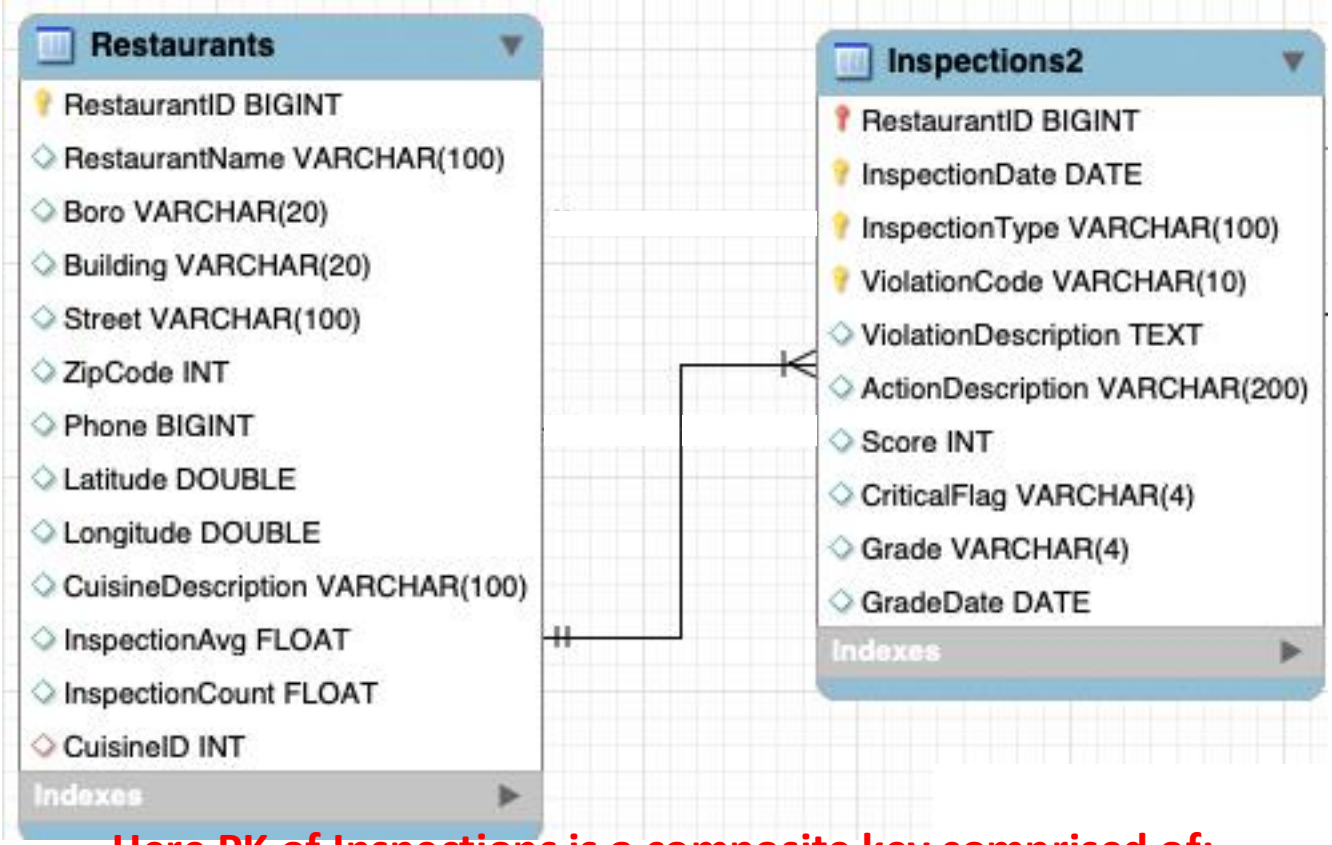
Strong (identifying) relationships

- **Primary key of the related entity contains a primary key component of the parent entity**
- **Shown as solid line**

Here RestaurantID from Restaurants is part of Inspections2 table's Primary Key

Solid line indicates a strong (identifying) relationship between entities

1:M relationship on crow's foot diagram



Solid line indicates a strong (identifying) relationship between entities

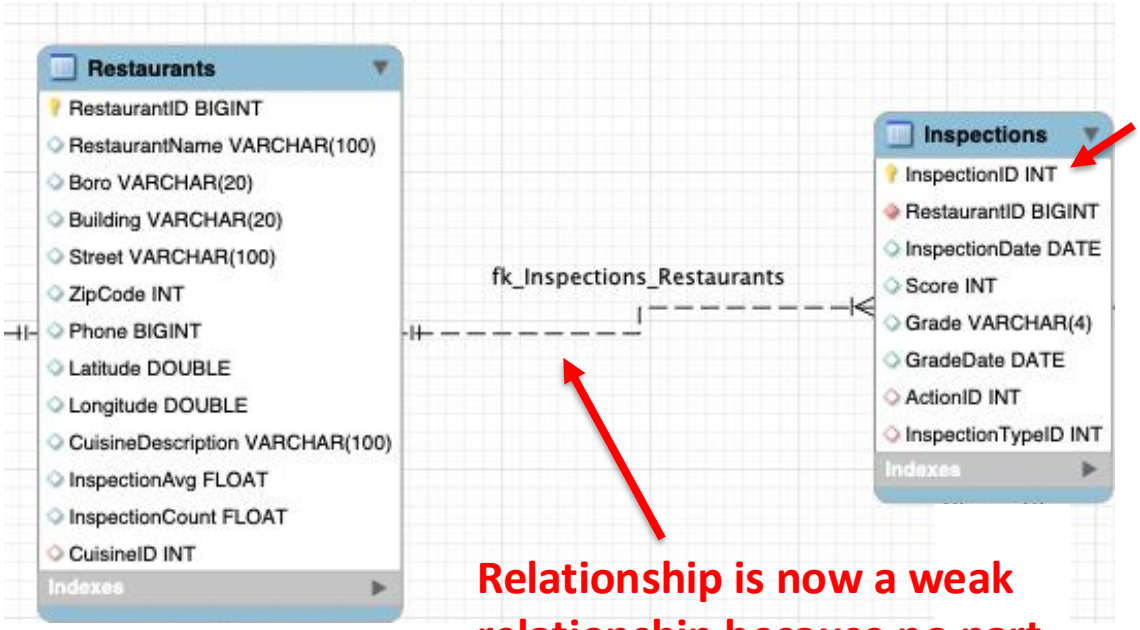
The related table is existence-dependent on the parent table (an Inspection cannot exist without the parent Restaurant)

Related table PK contains part of PK of parent table

Here PK of Inspections is a composite key comprised of: RestauantID, InspectionDate, InspectionType, ViolationCode

- **Inspections PK contains part of PK of Restaurants table**
- **Cannot have entry in Inspections without entry in Restaurants**
- **Inspections are existence-dependent on restaurants**

Giving Inspections a surrogate key makes for a weak relationship



Inspections PK is now a surrogate key

Relationship is now a weak relationship because no part of Restaurants PK (RestaurantID) is part of Inspections PK

Weak relationship is shown as a dashed line

Dashed line indicates a weak (non-identifying) relationship between entities

1:M relationship on crow's foot diagram

1 Restaurant can have 1 Cuisine type

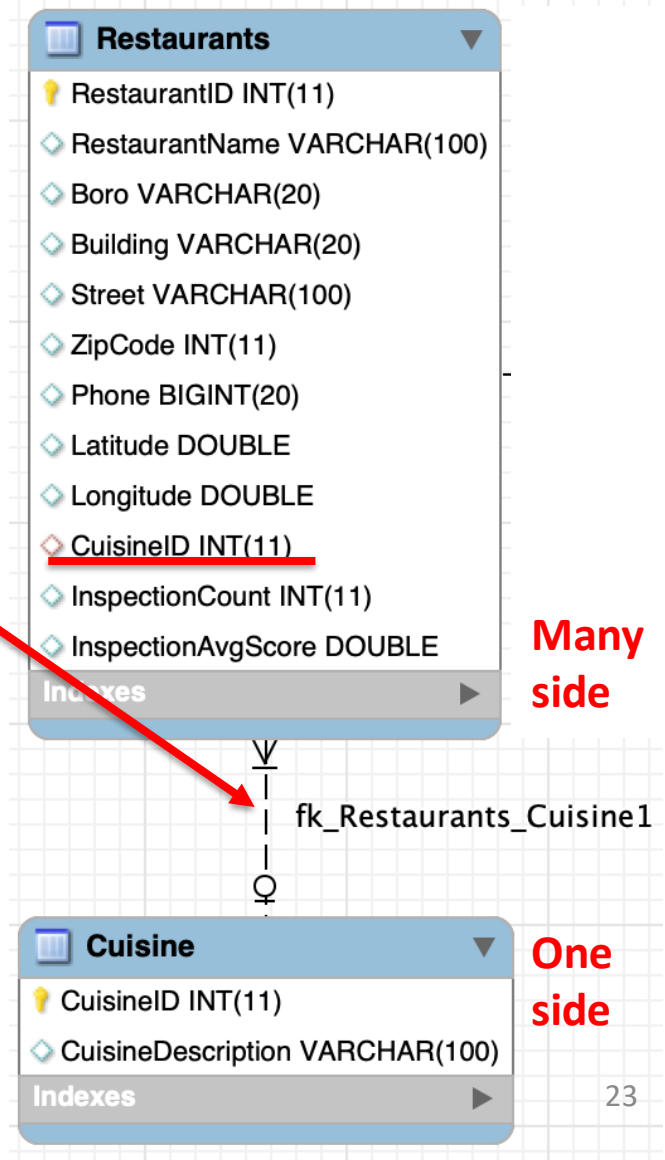
1 Cuisine type can have many restaurants

Design choice: make a look up table

- Consistency: Italian/italian/ITALIAN
- Easy updates: Italian -> Italian/Mediterranean
- Prevents invalid cuisines
- Smaller storage

Dashed line indicates a weak (non-identifying) relationship between entities

- PK of related table does not contain part of PK of parent table
- Here CuisineID of Cuisine table is not part of the PK of Restaurants
- Surrogate key makes Cuisine a strong entity

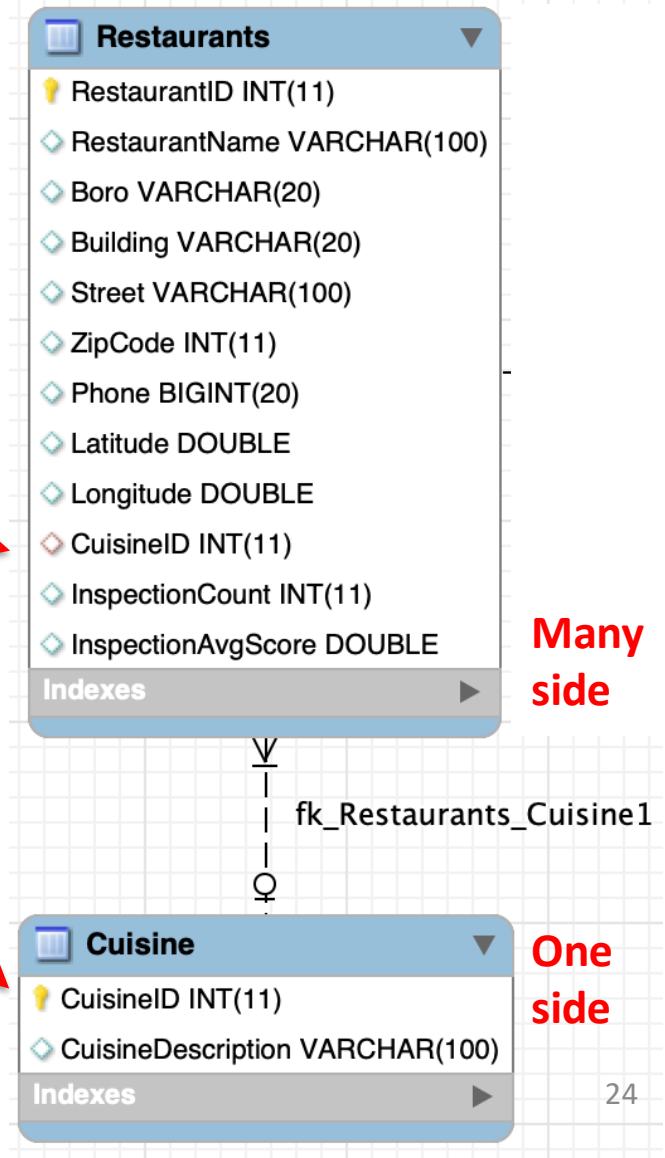


Implement 1:M relationship by including common attribute as foreign key in table

1:M relationship on crow's foot diagram

Implement 1:M by using a foreign key on the many side

Foreign key on the many side is primary key on the one side



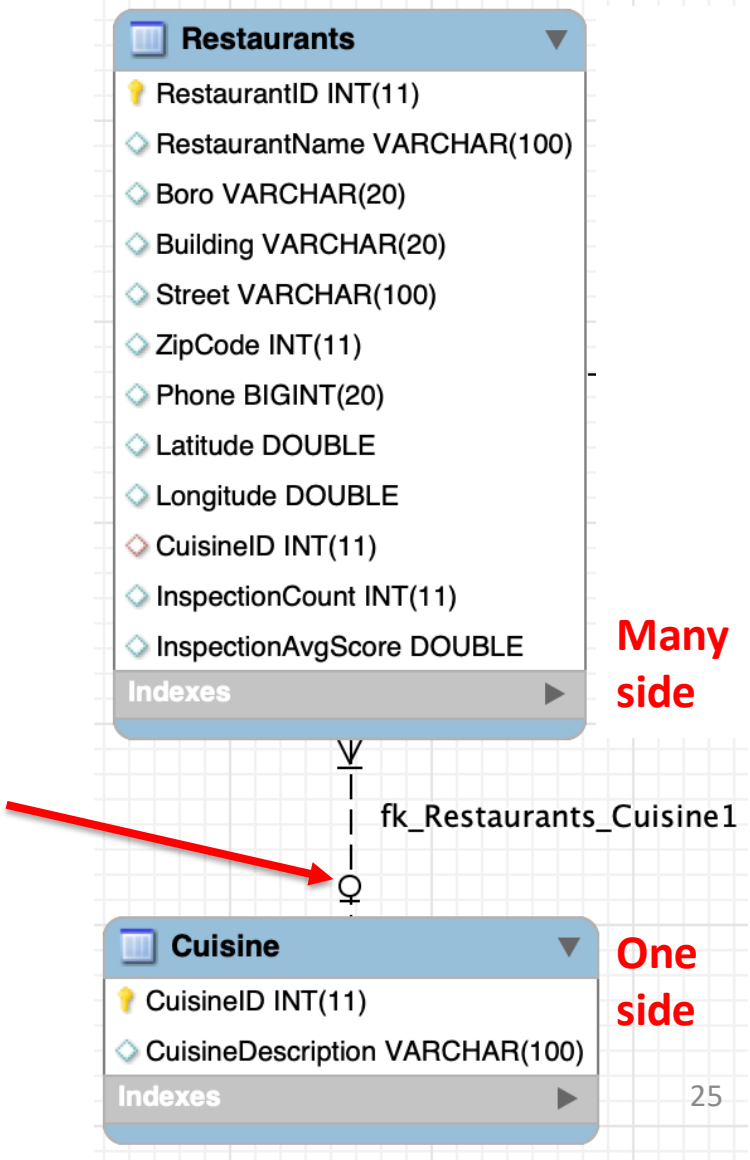
Implement 1:M relationship by including common attribute as foreign key in table

1:M relationship on crow's foot diagram

- **Optional participation**
  - **One entity occurrence does not require a corresponding entity occurrence in a particular relationship (FK can be NULL)**
- **Mandatory participation**
  - **One entity occurrence requires a corresponding entity occurrence in a particular relationship (FK cannot be NULL)**

Circle indicates CuisineID is optional in Restaurants

The "participation" is optional (foreign key can be NULL)



Agenda

1. Entity Relationship (ER) models

2. Relationships
   - One-to-many (1:M)
   - One-to-one (1:1)
   - Many-to-many (M:N)

3. How to build an ER model

4. Reverse and forward engineering

Examples of 1:1 relationships

Country ⇔ Capital
- Each country has one capital, each capital belongs to one country

Employee ⇔ Company car
- Each employee has at most one company car, each company car belongs at one employee

Order ⇔ Invoice
- Each order has one invoice, each invoice belongs to one order

Ask:
Can A have multiple B's? **No**
Can B belong to multiple A's? **No**
If both answers No: **1:1**

Handle:
Consider combining two tables into one

May be able to add a FK to one table

There are several reason to use a 1:1 relationship instead of one table

Security
Sensitive data (passwords, SSNs) in a separate, restricted table

Performance
Split large/rarely-used columns to reduce table size

Modularity
Optional data that not every record will have

Clarity
Logical separation of concerns

Compliance
Isolating data subject to regulations (HIPAA, PCI, GDPR)

One-to-one relationships are somewhat uncommon

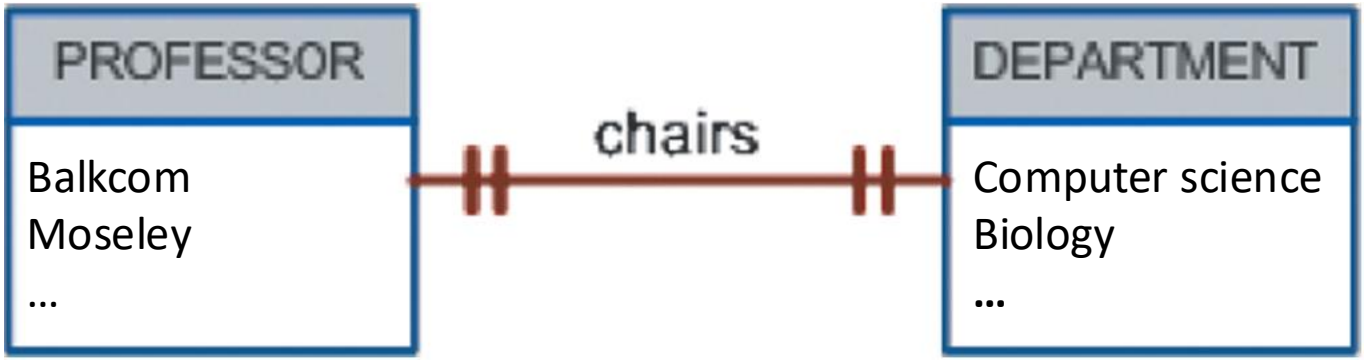
One-to-one (1:1)

A professor chairs at most one department

One department is chaired by one professor

Sometimes you cannot avoid them

One-to-one (1:1)

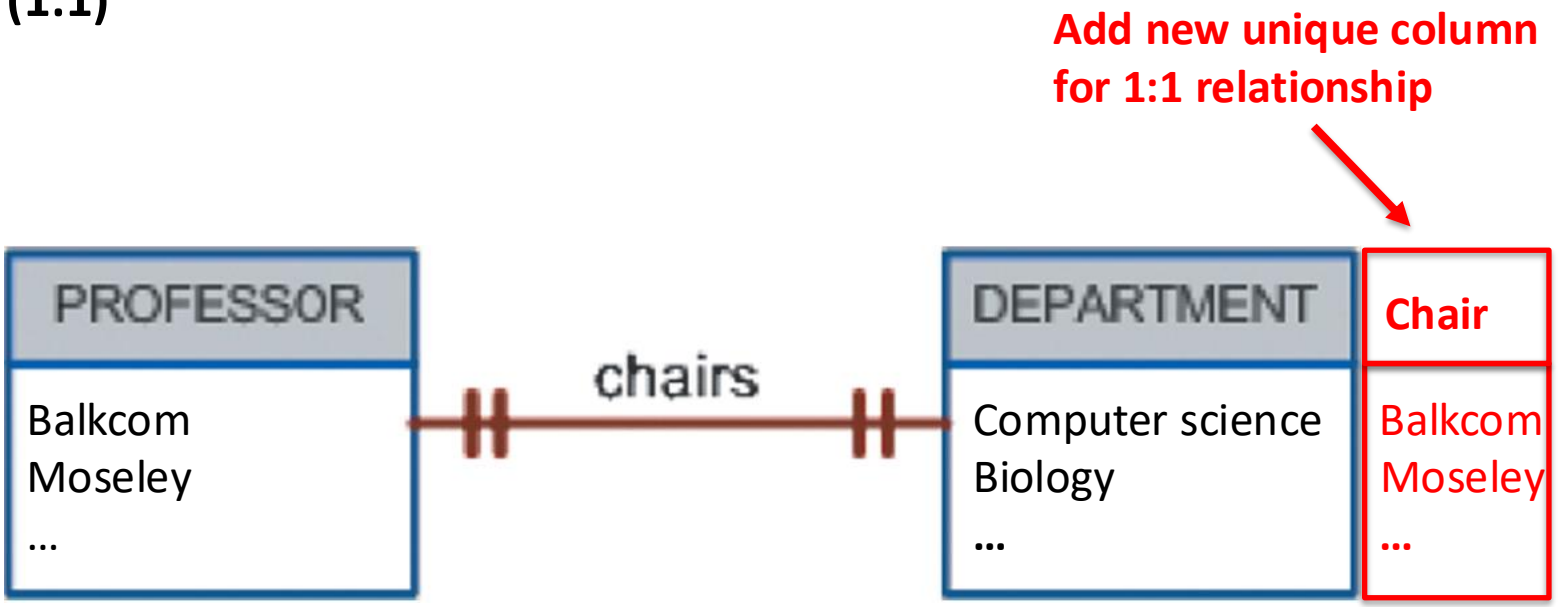


One entity can only be related to only one other entity in another table and vice versa

- Often you would just combine the attributes of both tables into one table (look for two tables with the same PK)
- Sometimes you can't do that

Implement using a column in one table and with a unique constraint

One-to-one (1:1)



To implement here:

- Add a column in Department for the Chair
- Make Chair column *unique* (no duplicates allowed, otherwise 1:M)
- Fill column with PK of Professor that chairs a department (e.g., Balkcom for CS)
- One department now has one chair (due to one attribute)
- One professor can only chair one department (due to unique constraint on Chair)

We will look at another variant next class

Agenda

1. Entity Relationship (ER) models

2. Relationships
   - One-to-many (1:M)
   - One-to-one (1:1)
   - Many-to-many (M:N)

3. How to build an ER model

4. Reverse and forward engineering

Examples of M:N relationships

Student ⇔ Courses
- Each student can take many courses, each course has many students

Actor ⇔ Movie
- Each actor can be in many movies, each movie can have many actors (CS10 PS-4)

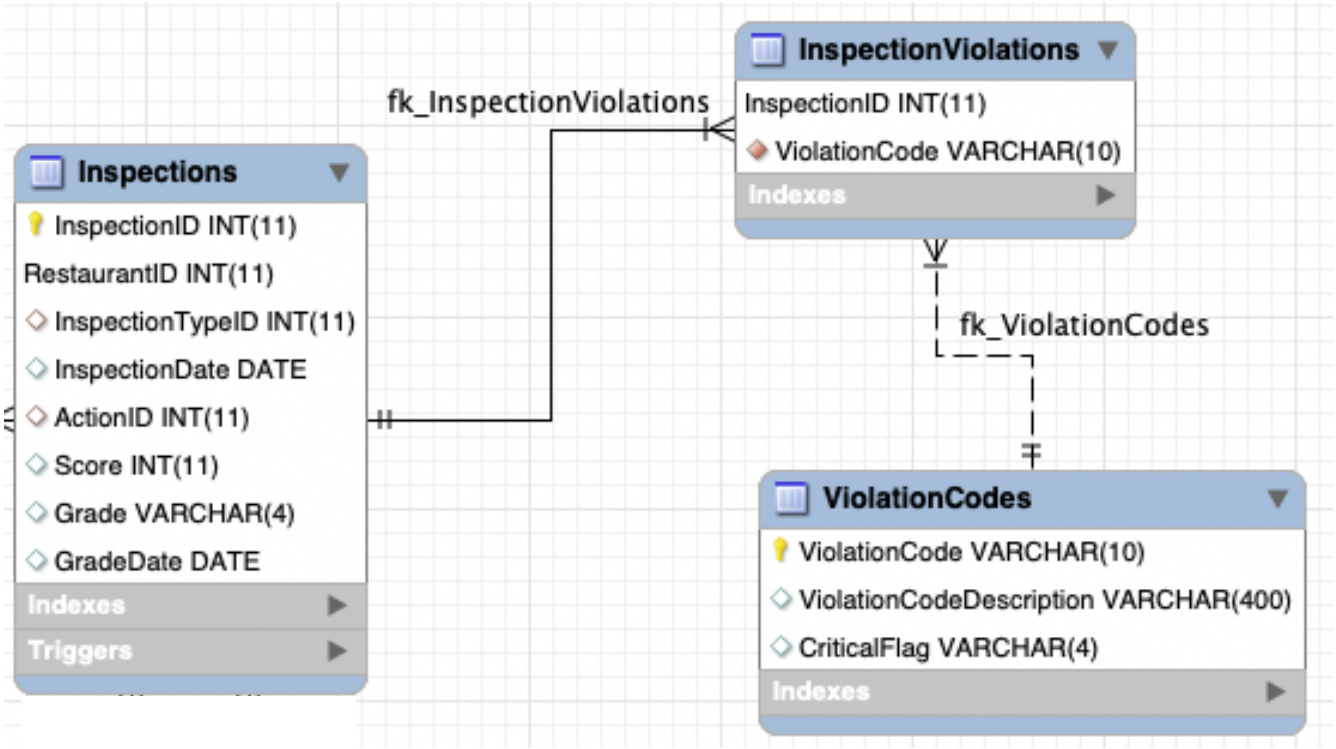
Employee ⇔ Project
- Each employee can be part of many projects, each project can have many employees

Ask:
Can A have multiple B's? **Yes**
Can B belong to multiple A's? **Yes**
If both answers Yes: **M:M**

Handle:
Use a joining (bridging table) to create two 1:M relationshiop

We have no direct way to model many-to-many relationships

Many-to-many (M:N)

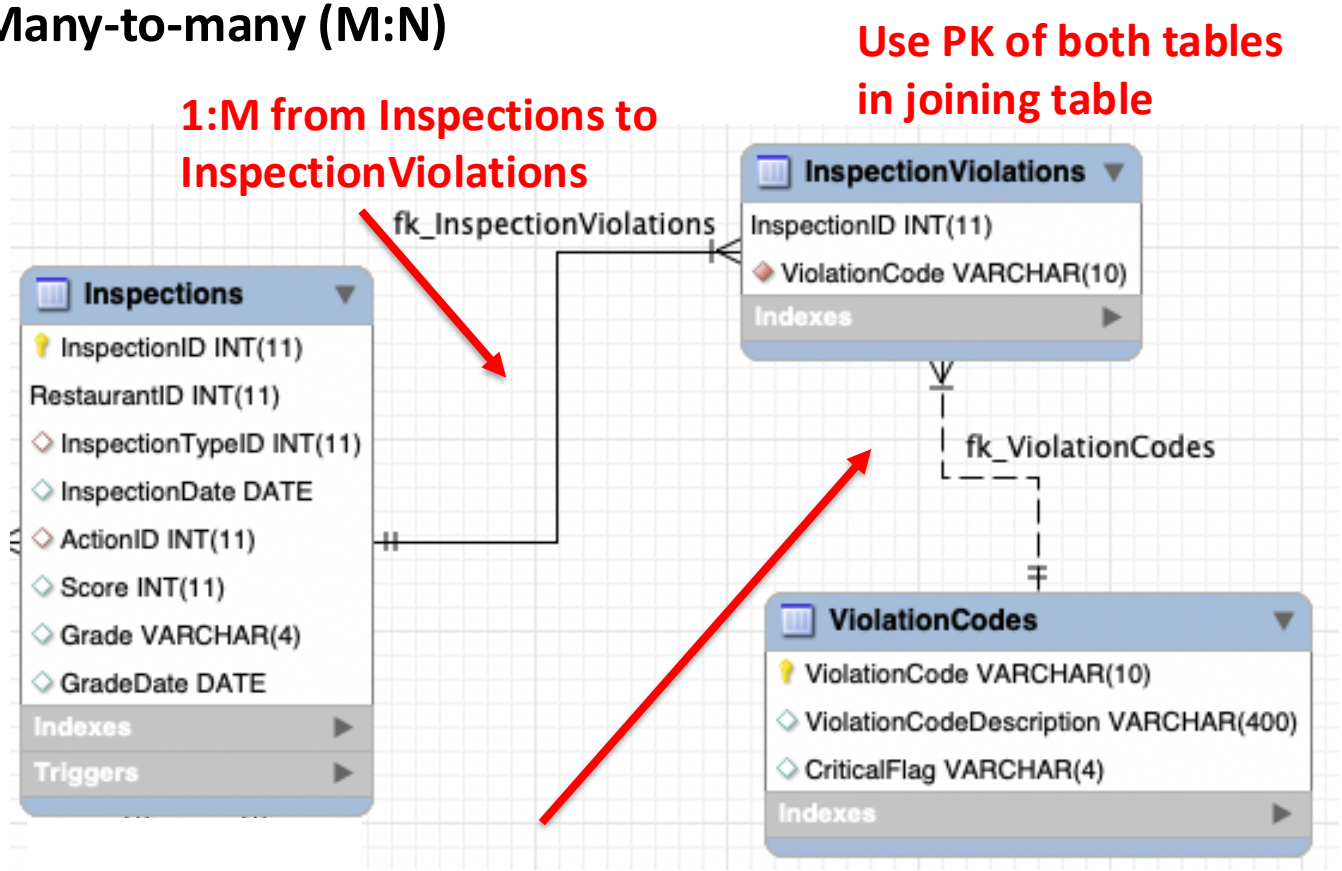


Problem:

- **One inspection can have many violation codes**
- **One violation code may occur in many inspections**
- **Many-to-many relationship**
- **We have no direct way to model M:N relationships**

Implement M:N with a joining table, create two 1:M relationships

Many-to-many (M:N)



NOTE: using InspectionID in Inspections table (instead of 4 attribute PK)

Problem:

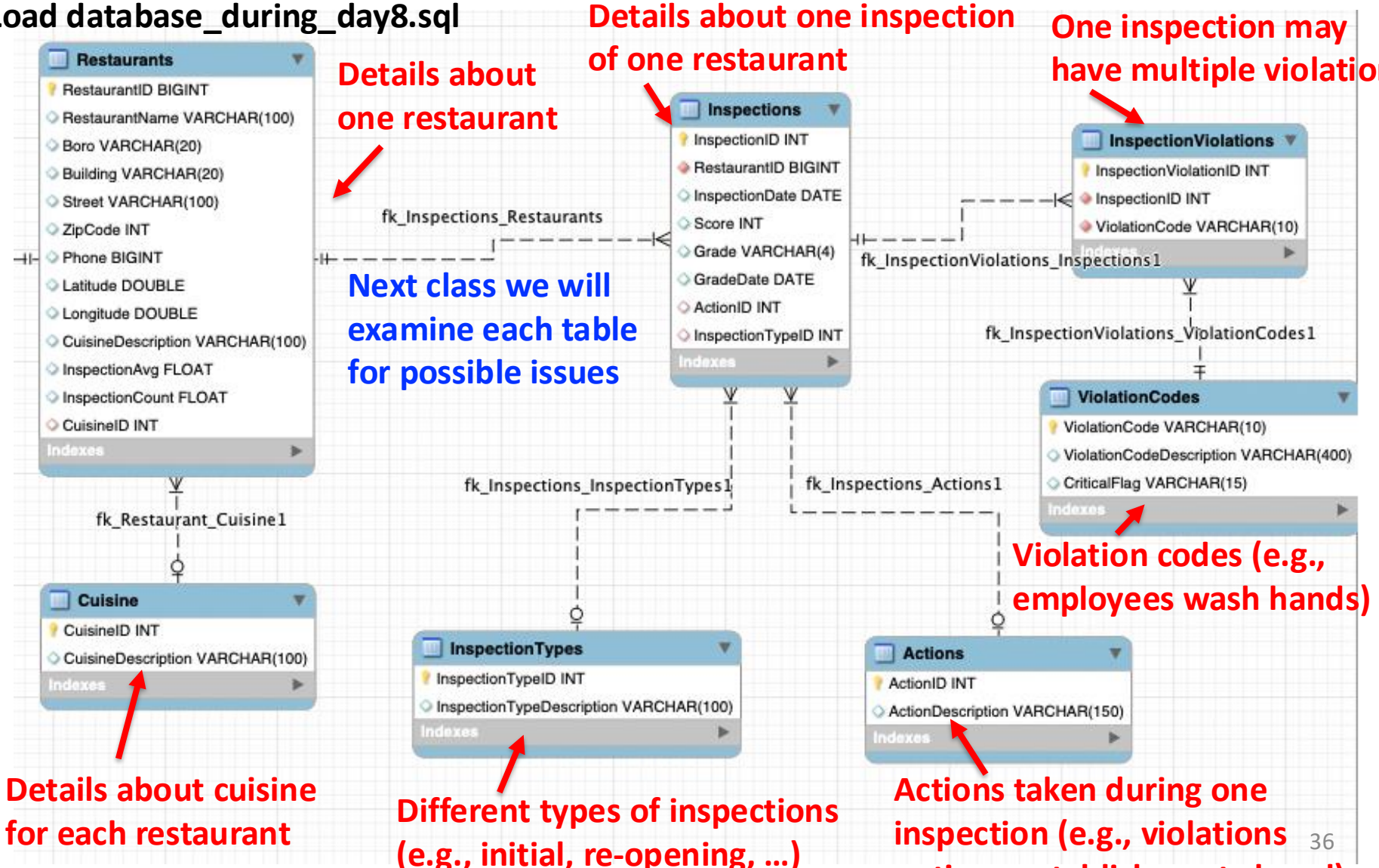
- One inspection can have many violation codes
- One violation code may occur in many inspections
- Many-to-many relationship
- We have no direct way to model M:N relationships

Solution:

- Use a joining (bridging) table (InspectionViolations here)
- Create two 1:M relationships

Entities and relationships for our health inspections database ERD

Load database_during_day8.sql

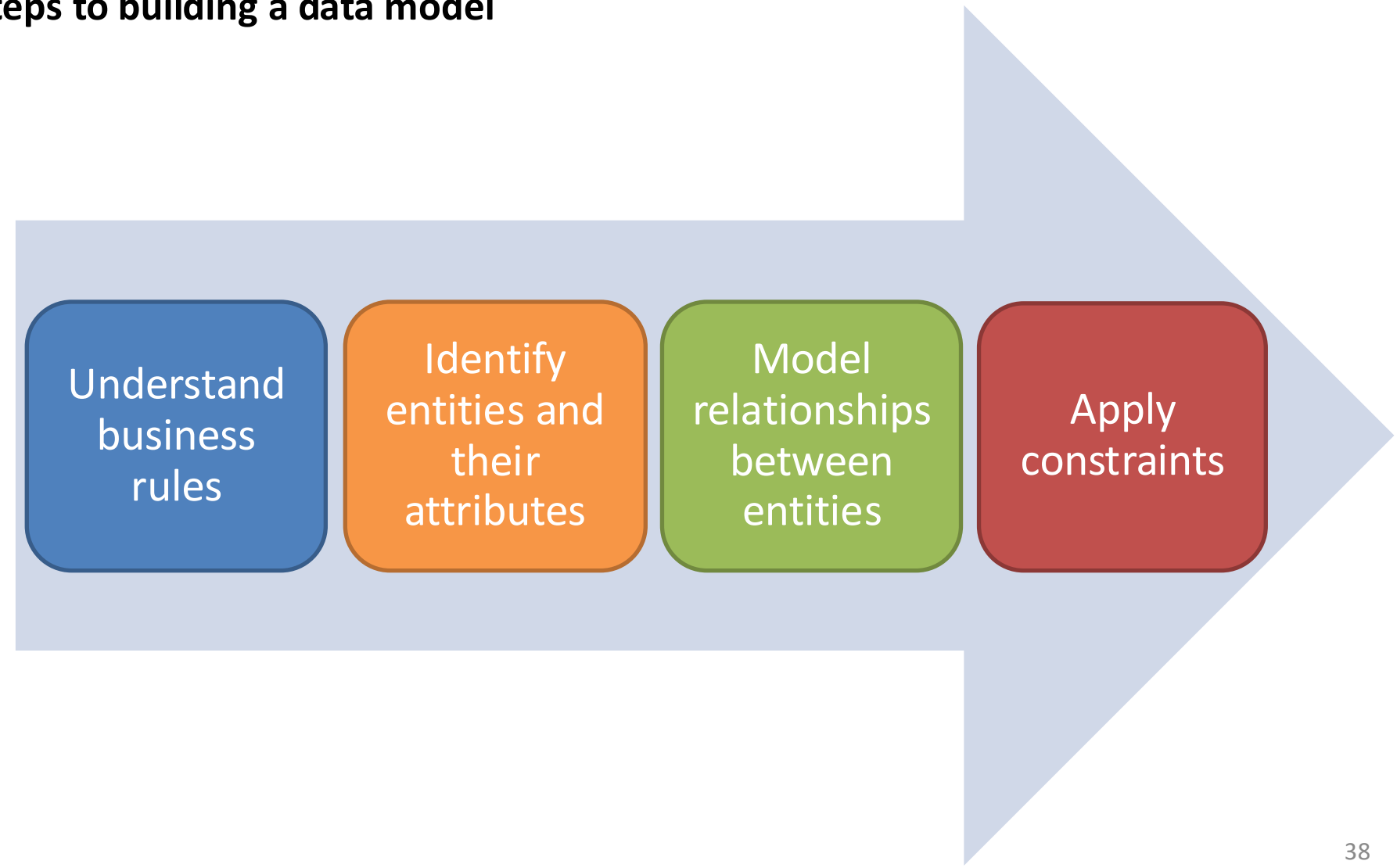


Agenda

1. Entity Relationship (ER) models
2. Relationships
3. How to build an ER model
4. Reverse and forward engineering

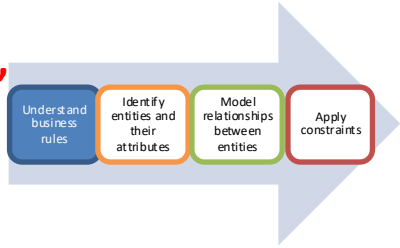
Data models are a (relatively) simple expression of the real world; build in steps

Steps to building a data model



First understand business rules so you know how the system should behave

Understand business rules **Output of this work is sometimes called a "specification of functional requirements"**



What are business rules?

- "Business rules" really means organization's rules
- "Brief, precise, and unambiguous *written* description of a policy procedure, or principle within a specific organization"
- Important to get this right!

Example:

- The college has many departments
- Each department belongs to one college (e.g., Arts & Sciences, Tuck, Thayer, Geisel, …)

How to I learn about the business rules?

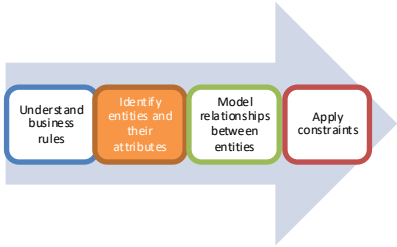
- Review written procedures – tells you how things <u>*should*</u> be done
- Talk to people to find out how it <u>*does*</u> work:
  - C-level – Have view of large portions of the organization, think they understand details, but frequently do not
  - Mid-level managers – know their part of the organization, but may not have big picture of how pieces work together
  - Users – might tell you how it really works

- Written business rules often help organization understand itself better
- Can lead to "business process engineering" to make organizational changes
- Consultants make lots of money doing this!

Next identify the entities (the nouns) involved and create them in the database

Identify entities and their attributes

Once entities and attributes established, create tables with DDL commands



Entities

- Person, place, thing, or event (noun)
- Normally become tables in the database
- Examples: Employee, Customer, Product
- Entities instances (rows) should be "distinguishable" from other entities based on keys

Attributes

- Characteristic of an entity
- Example: First name, Last Name, SSN

Some advice about naming

- I like to prefix attribute names with the entity name
- Example:, CustomerName CustomerAddress
- I think of this like a namespace
- Helps prevent confusion later (e.g., does Name mean customer name or product name?)

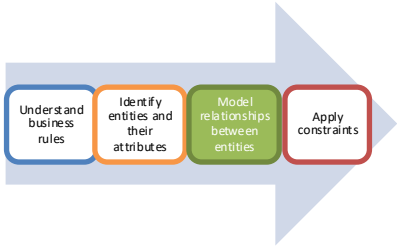
```
#create new database
CREATE SCHEMA 'new_schema';

#create student entity as table with
attributes and their types
CREATE TABLE STUDENT (
        STU_NUM int,
        STU_LNAME varchar(15),
        STU_FNAME varchar(15),
        STU_INIT varchar(1),
        STU_DOB datetime,
        STU_GPA numeric(4,2)
);
```

I prefer to spell out Student (not STU) and LastName (not LName)

Then model relationships between entities (the verbs) using 1:M, M:N, or 1:1

Three types of relationships between entities



One to many (1:M or 1..*) **1:M Use a foreign key**

- Associations among two or more entities where one entity is associated with two or more other entities
- Example
  - A painter can paint many paintings
  - Each painting is only painted by one painter
- Ask question in both directions:
  - How many instances of B (paintings) are related to one instance of A (painter)?
  - And how many instances of A (painter) are related to one instance of B (painting)
- Other examples?

Many to many (M:N or M:M or *..*) **M:N Use a joining table**

- Employee may learn many skills
- More than one employee can learn the same skill
- We must model these relationships using a joining table

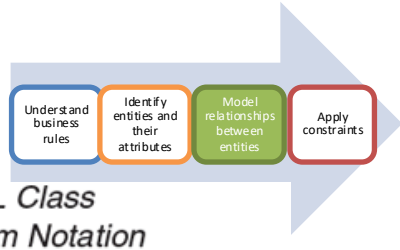
One to one (1:1 or 1..1)

Combine tables or add column to one table and set as unique

- A store is managed by one employee
- An employee can only manage one store

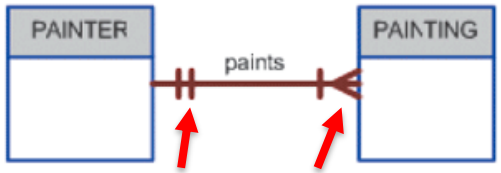
Draw relationships on an Entity Relationship Diagram using 1 of 3 formats

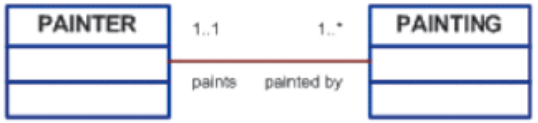
Three types of Entity Relationship Diagrams (ERD)



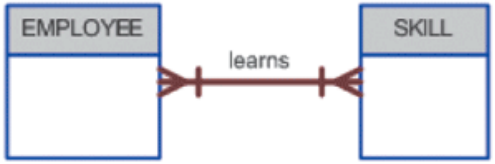
Chen Notation | *Crow's Foot Notation* | *UML Class Diagram Notation*

A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGs; each PAINTING is painted by one PAINTER.





A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLs; each SKILL can be learned by many EMPLOYEEs.

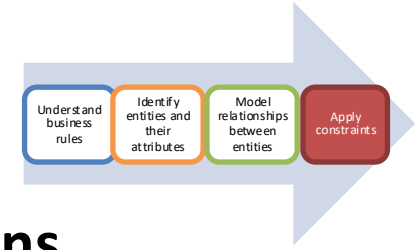


A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.

Source: Coronel and Morris

Finally apply any attribute constraints

Apply constraints



Attributes are sometimes limited to particular domains

- GPA must be between 0 and 4.0
- Employee's salary must be between $10K and $1M

Add CHECK constraint when defining table (e.g., GPA double CHECK (GPA >=0 and GPA <=4)

Once everything is set up, Data Manipulation Language (DML) allows us alter the database contents

- Perform CRUD (create, read, update, delete)
- SQL is both DML and DDL

Apply these steps in a phased approach

Design phases

- Models a subset of the total problem
- Work with end users to get this right
- Hardware and software independent

External schema

- Combine external models into global view of the entire database
- Work with managers to get this right
- Hardware and software independent

Conceptual schema

- Database's view
- Considers the specific DBMS used (e.g., Oracle vs MySQL vs Mongo)
- Hardware independent
- **Software dependent**

Internal schema

- How the database will be deployed on actual hardware
- **Hardware dependent**
- **Software dependent**

Physical schema

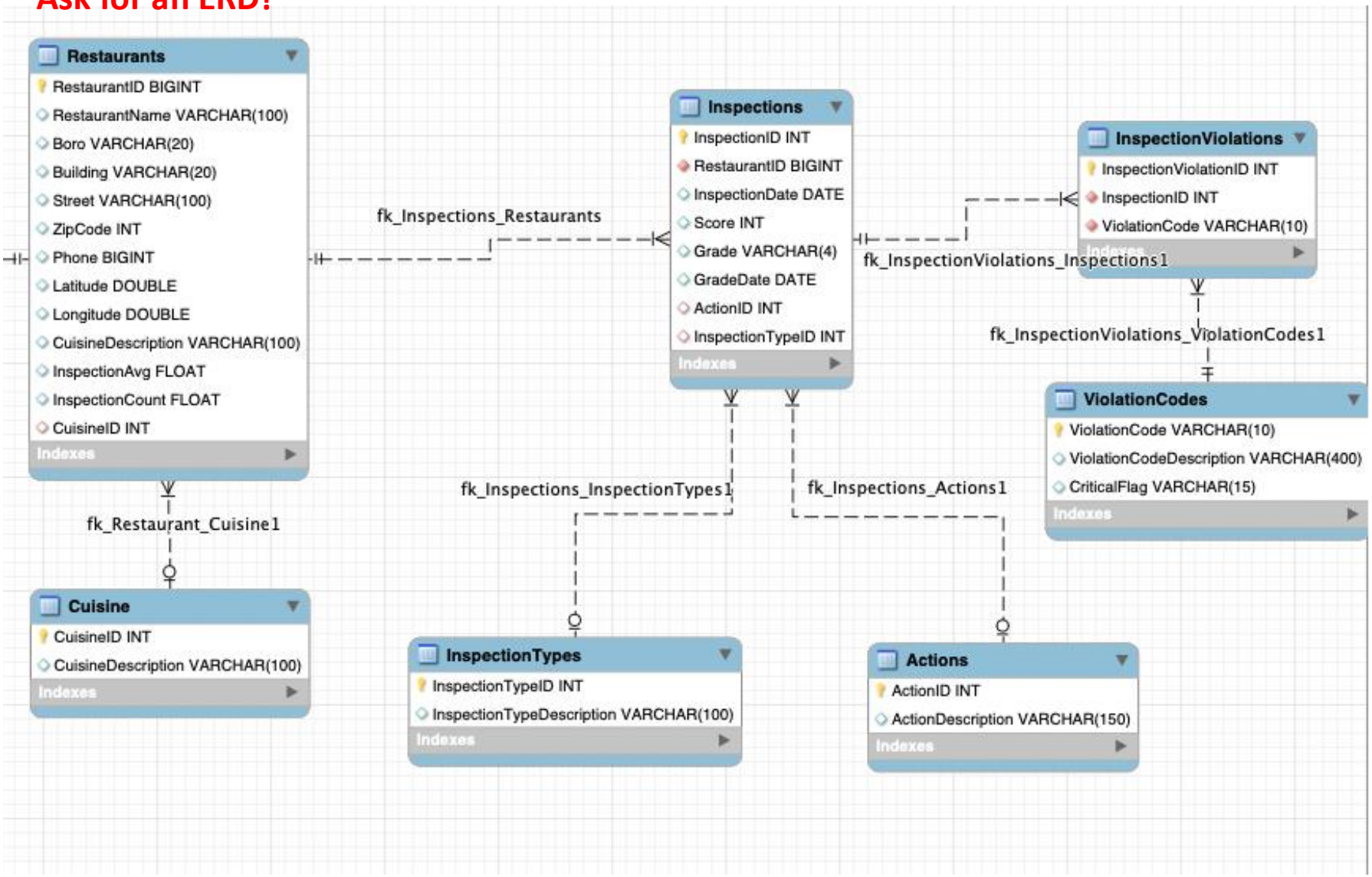
Adapted from Coronel and Morris

Agenda

1. Entity Relationship (ER) models
2. Relationships
3. How to build an ER model
4. Reverse and forward engineering

What do you do when you first encounter a new database?

Ask for an ERD!



DEMO: Reverse engineer an existing database to get an ERD

Reverse engineering a database is a quick way to get an ERD of an existing database

Reverse engineer nyc_data

- Download “database_during_day8.sql” from course web page
- From MySQL Workbench choose Database->Reverse engineer
- Make connection to database
- Select nyc_data
- Re-arrange tables into logical order

DEMO: Forward engineer a new schema based on a new ERD

Forward engineering is a way to quickly layout an entire database

Forward engineer a new schema

1. Create new ERD
   - From MySQL Workbench choose File->New model
   - Change schema (double click on mydb, use lowercase!)
   - Add diagram
     - Add tables
     - Add relationships (start from many side, then connect one side!)

2. Create schema
   - Database->Forward engineer to create new schema based on ERD

Practice

Forward engineer a database according to the following rules to track painters, paintings, and galleries for a famous art museum:

- Entities
  - Artists
  - Paintings
  - Galleries
- Relationships
  - A painting is painted by a one artist
  - An artist may create zero or more paintings
  - A gallery can exhibit many paintings, but each painting can be exhibited in only one gallery
  - Similarly, a painting is painted by a single painter, but each painter can paint many paintings
- Execute with MySQL Workbench->Database->Forward engineer

Note: use only lower case letters for your schema name (but tables names can be TitleCase)

More practice: allow multiple artists per painting