Data Modeling

Data Boot Camp
Lesson 9.3

Class Objectives

By the end of today's class you will be able to:

Apply data modeling techniques to database design.

Normalize data.

Identify data relationships.

Create visual representations of a database through entity relationship diagrams.

Instructor Demonstration
Data Normalization

Data normalization is the process of restructuring data to a set of defined "normal forms."

The process of data normalization eliminates data redundancy and inconsistencies.

Data Normalization

Three main forms of normalization

Process of restructure data to a set of "normal forms."

Reduce and eliminate data redundancy and inconsistencies

Three most common forms:

First normal form (1NF)

Second normal form (2NF)

Third normal form (3NF)

There are even more levels!

Data Normalization

Each field in a table row should contain a single value

Each row is unique

Rows can have fields that repeat

But whole rows do not fully match

Data Normalization

Raw Data

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Patricia Smith	Mrs.
1FAFP33Z24W147740	Oil Change, Alignment	Patricia Smith	Ms.
KNDJD736875757954	Oil Change, Brake Replacement	Mikhail Ivanov	Mr.
3N1AB7AP7EL611028	Transmission Rebuild	Mikhail Ivanov	Mr.

Normalization

→

First Normal Form

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Patricia Smith	Mrs.
1FAFP33Z24W147740	Oil Change	Patricia Smith	Ms.
1FAFP33Z24W147740	Alignment	Patricia Smith	Ms.
KNDJD736875757954	Oil Change	Mikhail Ivanov	Mr.
KNDJD736875757954	Brake Replacement	Mikhail Ivanov	Mr.
3N1AB7AP7EL611028	Transmission Rebuild	Mikhail Ivanov	Mr.

Data Normalization

Raw Data

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Patricia Smith	Mrs.
1FAFP33Z24W147740	Oil Change, Alignment	Patricia Smith	Ms.
KNDJD736875757954	Oil Change, Brake Replacement	Mikhail Ivanov	Mr.
3N1AB7AP7EL611028	Transmission Rebuild	Mikhail Ivanov	Mr.

Normalization

First Normal Form

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Patricia Smith	Mrs.
1FAFP33Z24W147740	Oil Change	Patricia Smith	Ms.
1FAFP33Z24W147740	Alignment	Patricia Smith	Ms.
KNDJD736875757954	Oil Change	Mikhail Ivanov	Mr.
KNDJD736875757954	Brake Replacement	Mikhail Ivanov	Mr.
3N1AB7AP7EL611028	Transmission Rebuild	Mikhail Ivanov	Mr.

Data Normalization

Second Normal Form (2NF) *"Reduce Redundancy"*

Be in First Normal Form

Table contains a Primary Key

Provides unique identifier for each row
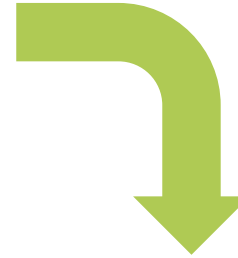
Ideally in a single column

All columns are entirely dependent on the table's primary key

Data Normalization

Data in 1NF

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Patricia Smith	Mrs.
1FAFP33Z24W147740	Oil Change	Patricia Smith	Ms.
1FAFP33Z24W147740	Alignment	Patricia Smith	Ms.
KNDJD736875757954	Oil Change	Mikhail Ivanov	Mr.
KNDJD736875757954	Brake Replacement	Mikhail Ivanov	Mr.
3N1AB7AP7EL611028	Transmission Rebuild	Mikhail Ivanov	Mr.

2NF Normalization

Services Performed Table

ID	Customer Vehicle ID	Services Performed
1	1	Oil Change
2	2	Oil Change
3	2	Alignment
4	3	Oil Change
5	3	Brake Replacement
6	4	Transmission Rebuild

Customer Vehicle Table

ID	VIN	Customer Name	Salutation
1	3D7LX39C86G106066	Patricia Smith	Mrs.
2	1FAFP33Z24W147740	Patricia Smith	Ms.
3	KNDJD736875757954	Mikhail Ivanov	Mr.
4	3N1AB7AP7EL611028	Mikhail Ivanov	Mr.

**These VINs were randomly generated from randomvin.com
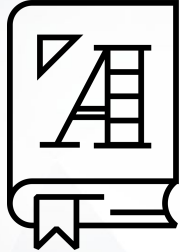
Transitive Dependency is the reliance of a column's value on another column through a third column.

Data Normalization

Transitive

- If **A** implies **B**, and **B** implies **C**, then **A** implies **C**.

Dependence

- One value relies on another.
- City relies on ZIP code; age depends on birthday.

For example:

- Consider these columns in the `Customer Vehicles` table: `VIN`, `Customer Name`, `Salutation`.
- `Customer Name` depends on `VIN`.
- `Salutation` depends on `Customer Name`.
- Hence, `Salutation` depends on `VIN`. This is an example of transitive dependency

Data Normalization

Must be in Second Normal Form

Contain non-transitively dependent columns

Transitively dependent columns depend on more than one column in the table

Data Normalization

2NF to 3NF Normalization

ID	VIN	Customer Name	Salutation
1	3D7LX39C86G106066	Patricia Smith	Mrs.
2	1FAFP33Z24W147740	Patricia Smith	Ms.
3	KNDJD736875757954	Mikhail Ivanov	Mr.
4	3N1AB7AP7EL611028	Mikhail Ivanov	Mr.

Customer Vehicles

ID	VIN	Customer ID
1	3D7LX39C86G106066	1
2	1FAFP33Z24W147740	2
3	KNDJD736875757954	3
4	3N1AB7AP7EL611028	3

Customer

ID	Name	Salutation_ID
1	Patricia Smith	3
2	Patricia Smith	2
3	Mikhail Ivanov	1

Salutation

ID	Salutation
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

Activity: Pet Normalizer

In this activity, you will practice data normalization skills using the provided data.

Activity: Pet Normalizer

- In pgAdmin, create a new database called `pets_db`.
- Use Excel to get the data into first normal form (1NF).
- Using the normalized CSV, create the following tables with continued normalized practices:
  - a table for owners that takes an ID and the owner's name
  - a table for pet names that takes two IDs, the pet's name, and the pet's type.
- Using the CSV file as guide, insert the data into respective tables.

- **Hint**: Be sure that each table has a unique primary key. 💡
- **Bonus**:
  - Create a service table that displays the different types of services that are offered.
  - Create a pet_names_updated table that takes an ID that will connect to the services table.
  - Join all three tables.

Time's Up! Let's Review.

Instructor Demonstration
Foreign Keys

Foreign Keys

Foreign Keys reference the primary key of another table.

	Can have a different name

	Do not need to be unique

Primary Key

family_id	family
1	Smiths
2	Jones

Primary Key   Foreign Key

child_id	family_id	children
11	1	Chris
22	1	Abby
33	1	Susy
44	2	Steve
55	2	Mary
66	2	Dillion

Activity: Foreign Keys

In this activity, you will create and populate two new tables with foreign keys that reference existing data.

Suggested Time:
15 Minutes

Activity: Foreign Keys

- Create a `customer` table with a customer first name and customer last name.
- Create a `customer_email` table with a foreign key that references a field in the original `customer` table.
- Populate the `customer_email` table with emails.
- Create a `customer_phone` table with a foreign key that references a field in the original customer table.
- Populate the `customer_phone` table with phone numbers.
- Test foreign keys by writing a query to insert data in the `customer_phone` table that does not have a reference ID in the `customer` table.
- Join all three tables.
- **Hint**: Think about how you can select certain columns in a table. Use those columns as a reference to insert data into a table. 💡

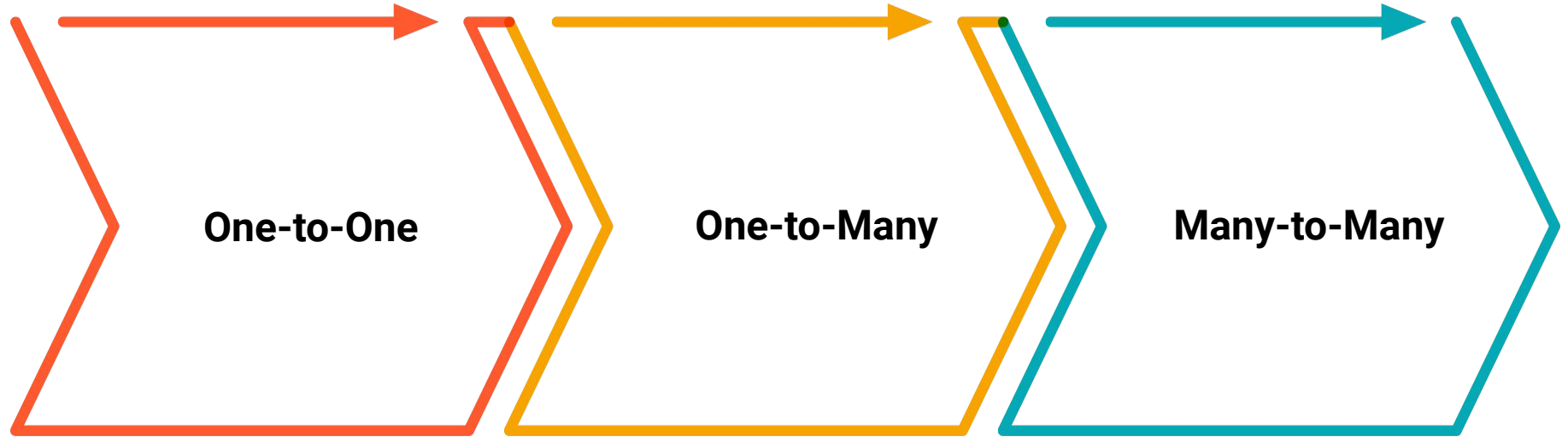**Note**: _Make sure all tables have primary keys that increment with each new row of data._

Time's Up! Let's Review.

Instructor Demonstration
Intro to Data Relationships

Intro to Data Relationships

One-to-One

One-to-Many

Many-to-Many

Intro to Data Relationships

One-to-One Relationship

ID	Name	Social Security
1	Homer	111111111
2	Marge	222222222
3	Lisa	333333333
4	Bart	444444444
5	Maggie	555555555

Each item in one column is linked to only one other item from the other column.

Here, each person in the Simpsons family can have only one social security number.
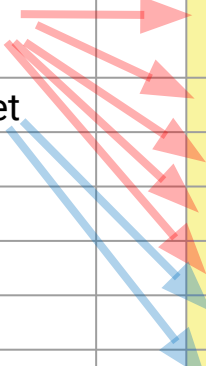
Each social security number can be assigned only to one person.

Intro to Data Relationships

One-to-Many Relationship

ID	Address		ID	Name	Social Security	AddressID
11	742 Evergreen Terrace		1	Homer	111111111	11
12	221B Baker Street		2	Marge	222222222	11
			3	Lisa	333333333	11
			4	Bart	444444444	11
			5	Maggie	555555555	11
			6	Sherlock	112233445	12
			7	Watson	223344556	12

Each address can be associated with multiple people.

Each person has an address.

The two tables, joined, would look like this.

Intro to Data Relationships

Many-to-Many Relationship

ID	Child		ID	Parent
1	Bart		11	Homer
2	Lisa		12	Marge
3	Maggie			

Each child can have more than one parent.

Each parent can have more than one child.

Intro to Data Relationships

Many-to-Many Relationship

ChildID	Child	ParentID	Parent
1	Bart	11	Homer
1	Bart	12	Marge
2	Lisa	11	Homer
2	Lisa	12	Marge
3	Maggie	11	Homer
3	Maggie	12	Marge

Each child can have more than one parent.

Each parent can have more than one child.

The two tables are joined in a junction table.

Intro to Data Relationships

Junction Table

parent

parent_id	INTEGER
parent_name	VARCHAR
...	

child_parent_junction

parent_id	INTEGER
child_id	INTEGER

child

child_id	INTEGER
child_name	VARCHAR
...	
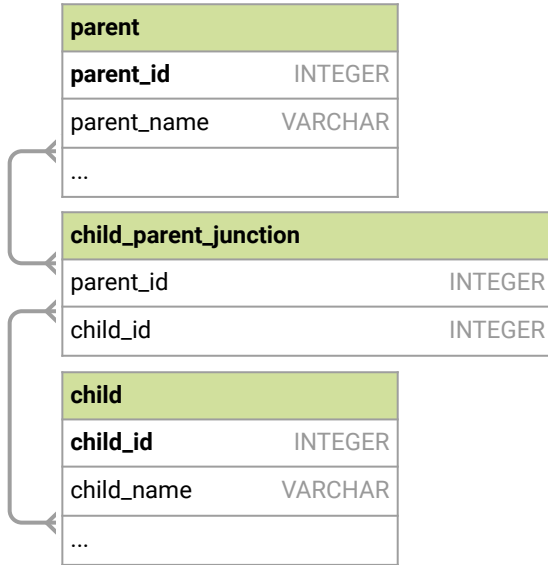
The Junction table contains many parent_id's and many child_id's

	parent_id integer	child_id integer
1	11	1
2	11	2
3	11	3
4	12	1
5	12	2
6	12	3

Join child and parent table to junction table

	parent_name character varying (255)	child_name character varying (255)
1	Homer	Bart
2	Homer	Lisa
3	Homer	Maggie
4	Marge	Bart
5	Marge	Lisa
6	Marge	Maggie

Activity: Data Relationships

In this activity, you will create table schemata for students and available courses, and then create a junction table to display all courses taken by students.

Suggested Time:
15 Minutes

Activity: Data Relationships

You are the database consultant at a new university. Your job is to design a database model for the registrar. The database will keep track of information on students, courses offered by the university, and the courses each student has taken.

- Create a `students` table that keeps track of the following:
  - Unique ID number of each student
  - Last and first names of each student
- Create a `courses` table that keeps track of the following:
  - Unique ID number of each course
  - Name of each course
- Create a `student_courses_junction` that keeps track of the following:
  - All courses that have been taken by each student
  - Term in which a course was taken by a student (spring or fall)
- Which data model is appropriate here: one to one, one to many, or many to many?
- **Bonus:**
  - If time allows, join and query the tables to get all data on the students.

Time's Up! Let's Review.

Take a Break!

Countdown timer

40:00

(with alarm)

Instructor Demonstration
Entity Relationship Diagram

An **entity relationship diagram**, or **ERD**, is a visual representation of entity relationships within a database.

Entity Relationship Diagram

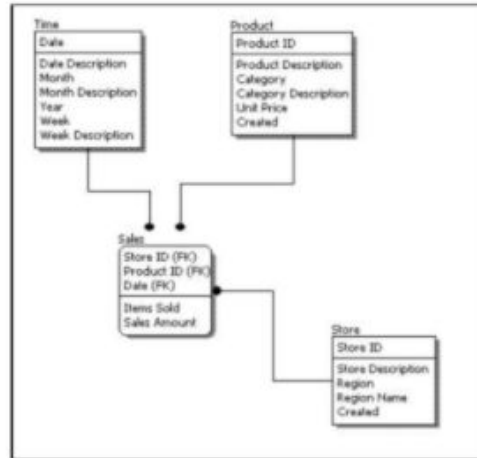**ERD use the following notation to create the models.**
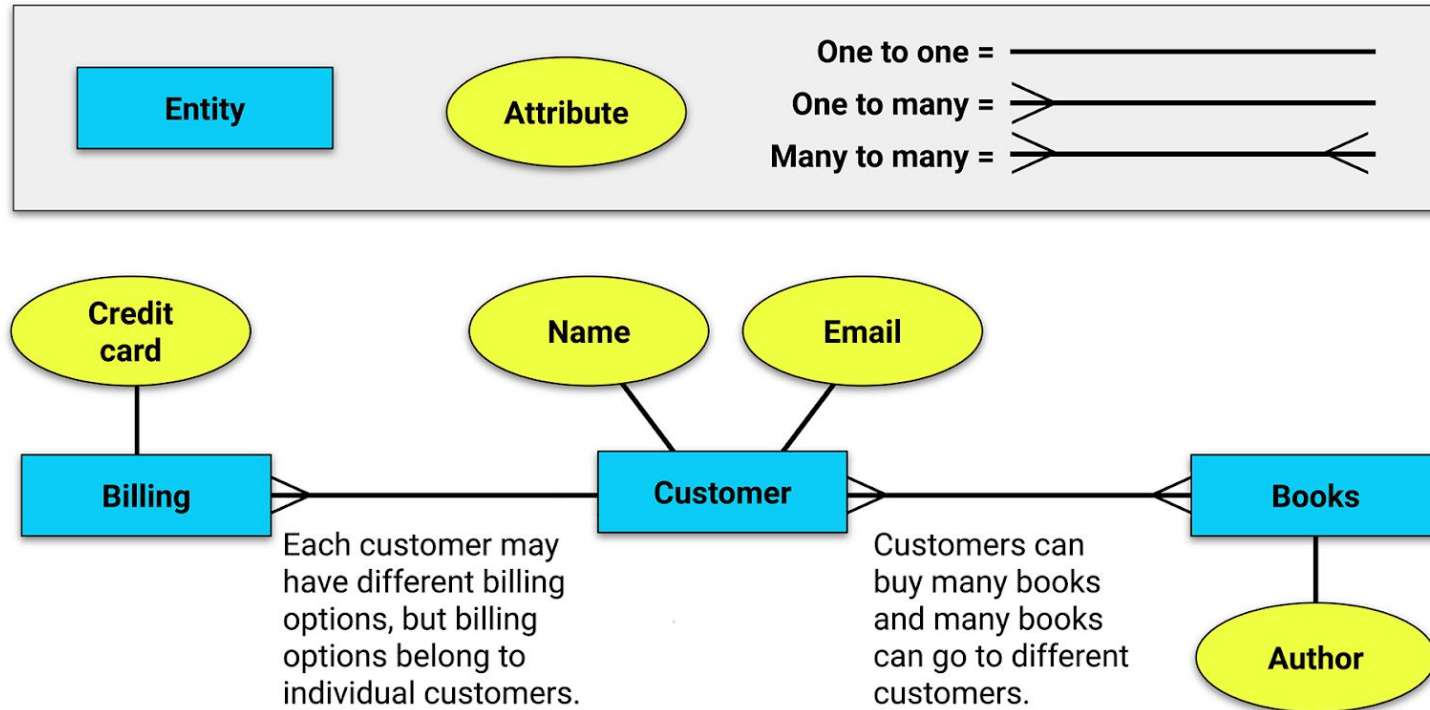
Entity Relationship Diagram
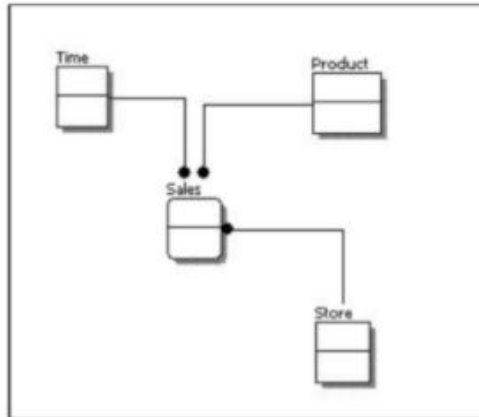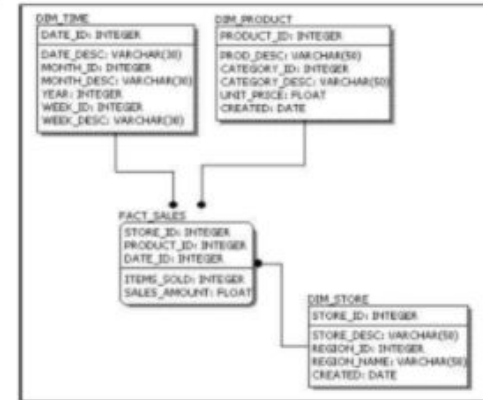
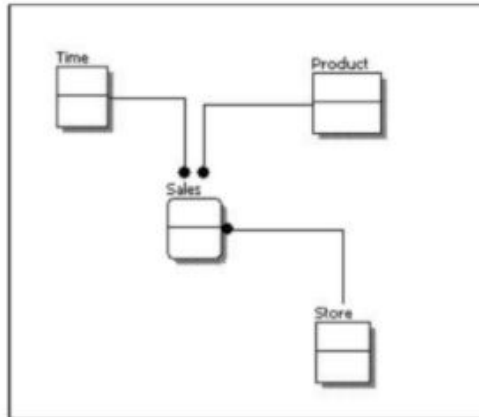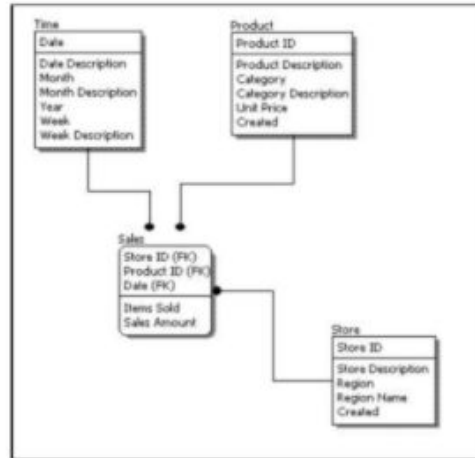**Three Types of ERDs or Data Models**

Conceptual Model Design — Logical Model Design — Physical Model Design
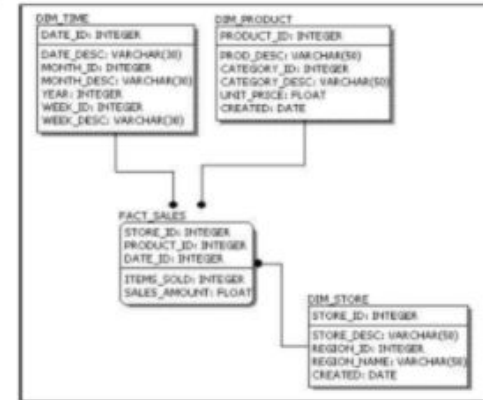
Entity Relationship Diagram

Three Types of ERDs or Data Models
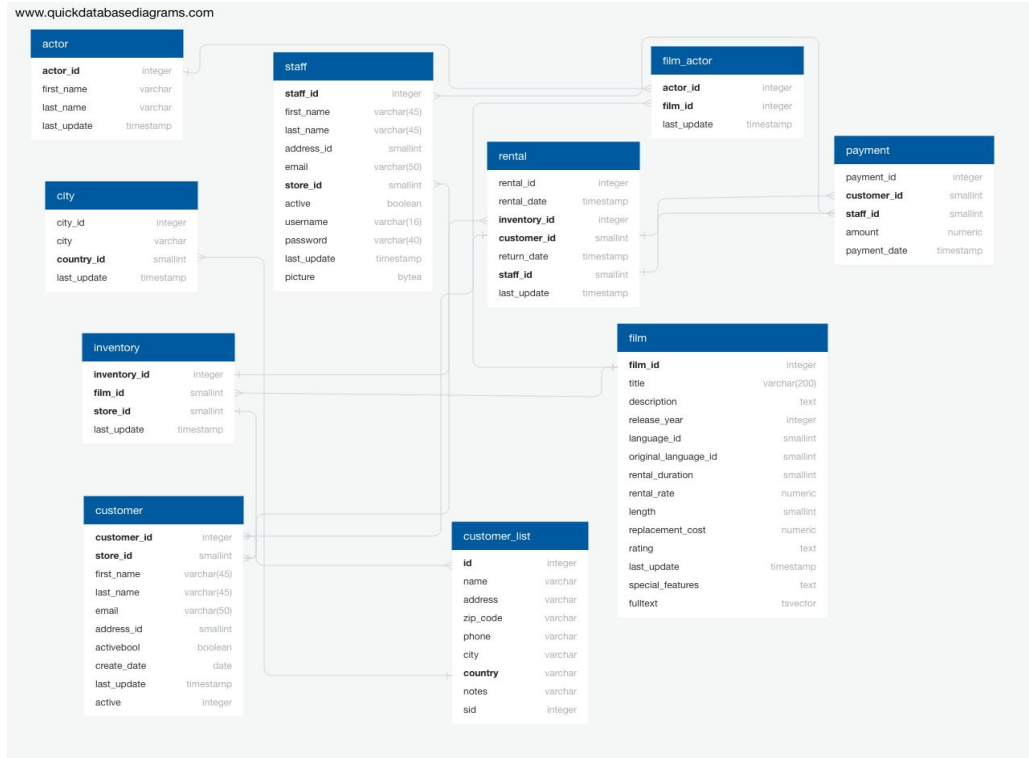

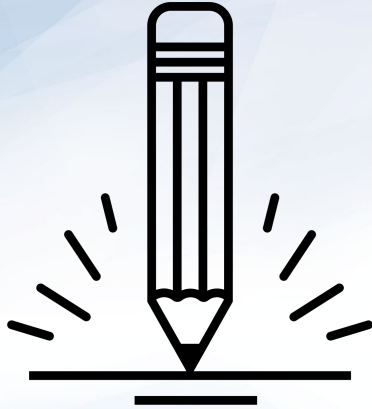Conceptual Model Design   Logical Model Design   Physical Model Design

Entity Relationship Diagram

Entities, their data types, and relationships are all illustrated in the diagram

Activity: Designing an ERD, Part I

In this activity, you will create a conceptual **ERD** for a gym owner.
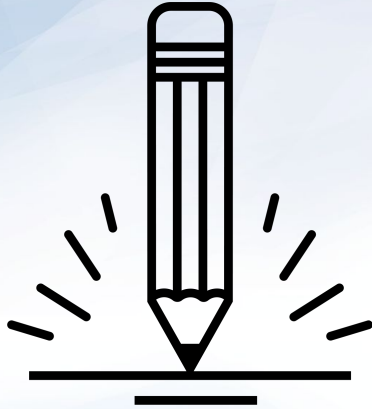
Activity: Designing an ERD, Part I

In this activity, you will work with a partner given the following scenario:

You are meeting with a gym owner who wants to organize his data in a database. Create a conceptual ERD for the gym owner.

- Create a conceptual ERD by determining the entities that will be present in the database, along with their attributes. Be sure to include the following: trainers, members, and gym as well as one more entity that you think is necessary.
- Create a diagram using the Quick Database Diagrams tool.
- When you are satisfied with the conceptual diagram, update it to a logical ERD by including column data types and primary keys.
- Update your existing diagram to reflect the changes.
- Hint:
  - Check your slack for the documentation for more in-depth explanations of entity relationship diagrams.

Time's Up! Let's Review.

Activity: Designing an ERD, Part II

In this activity, you and and your partner will continue designing an entity relationship diagram for the gym by transitioning your logical **ERD** created in the previous activity to a physical **ERD.**

Activity: Designing an ERD, Part II

- Using the starter code provided, return to Quick Database Diagrams and transition your logical ERD to a physical ERD by creating the relationships between tables.
- When you are satisfied with your ERD, write a corresponding schema file containing your `CREATE TABLE` statements
- In pgAdmin, connect to your server and create a new database named `gym`. Then open a query tool.
- Paste the code from your schema file in pgAdmin, and then execute the code.

Activity: Designing an ERD, Part II

- **Hints**
  - Foreign keys are added to each table represented by the FK acronym, followed by the relationship, e.g., `OrderID INT FK >- Order.OrderID`.
  - You will need to add foreign keys to your tables in order to map the data relationships
  - Remember to document the relationships between entities using the correct symbols. Here are the allowed relationship types:

- Keep in mind the following:
  - Each member belongs to only one gym.
  - Trainers work for only one gym, but a gym has many trainers.
  - Each member must have a trainer, but each trainer may instruct multiple members.
  - Each member has one credit card on file.
- Once you have created tables in pgAdmin, you can check the table creation with the following syntax: `SELECT * FROM Members;`

Allowed relationship types are:

–	– one TO one
–<	– one TO many
>–	– many TO one
>–<	– many TO many
–0	– one TO zero or one
0–	– zero or one TO one
0–0	– zero or one TO zero or one
–0<	– one TO zero or many
>0–	– zero or many TO one

Time's Up! Let's Review.
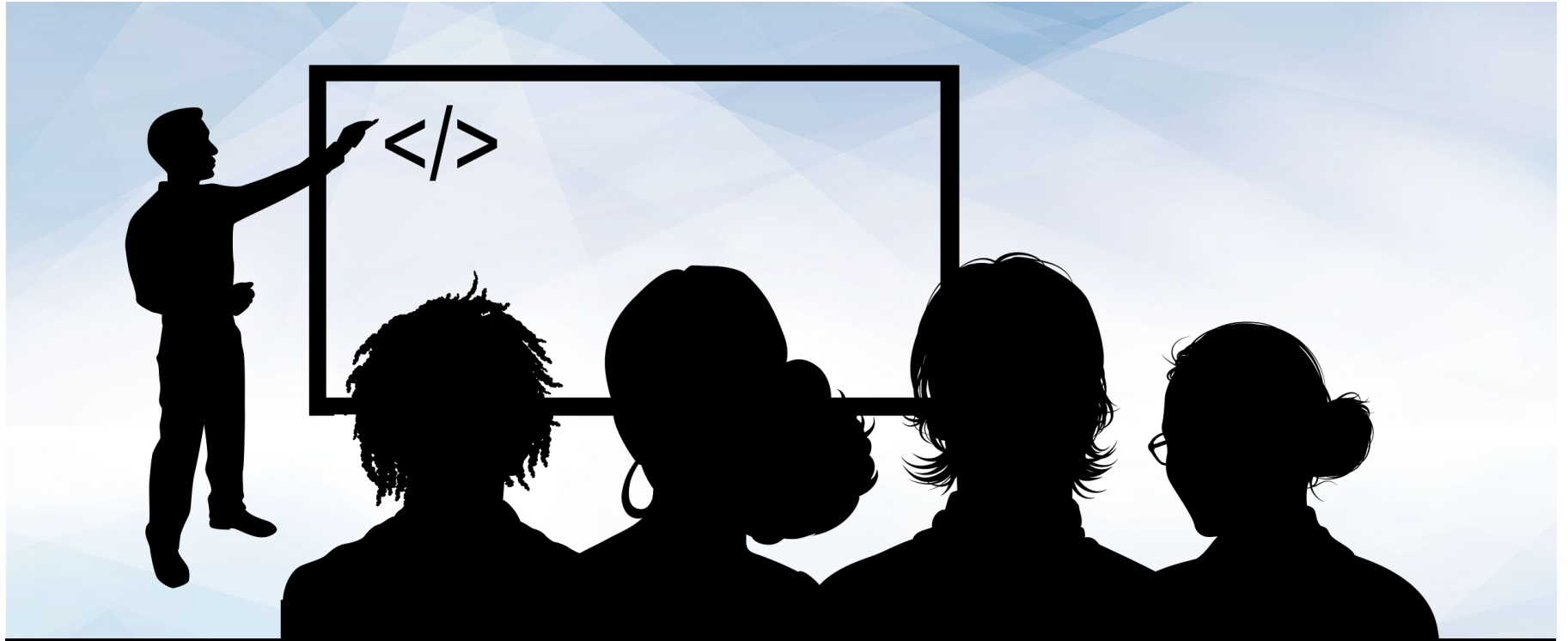
Instructor Demonstration
Introduction to Unions

Introduction to Unions

`<Time to code>`

Activity: Unions

In this activity, you will practice more with unions, by combining data from multiple tables without the use of joins.

Suggested Time:
15 Minutes

Activity: Designing an ERD, Part II

- Using `UNION`, write a PostgreSQL statement to query the number of rows in tables `city` and `country`.
- Use `UNION` to display from the tables `customer` and `customer_list` the ID of all customers who live in the city of London. Determine whether both tables contain the same customers by using `UNION ALL`.
- **Hint:**
  - For the second problem, consider using subqueries.

Time's Up! Let's Review.

The End