Data Modeling

Class Objectives

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

Apply data modeling techniques to database design.
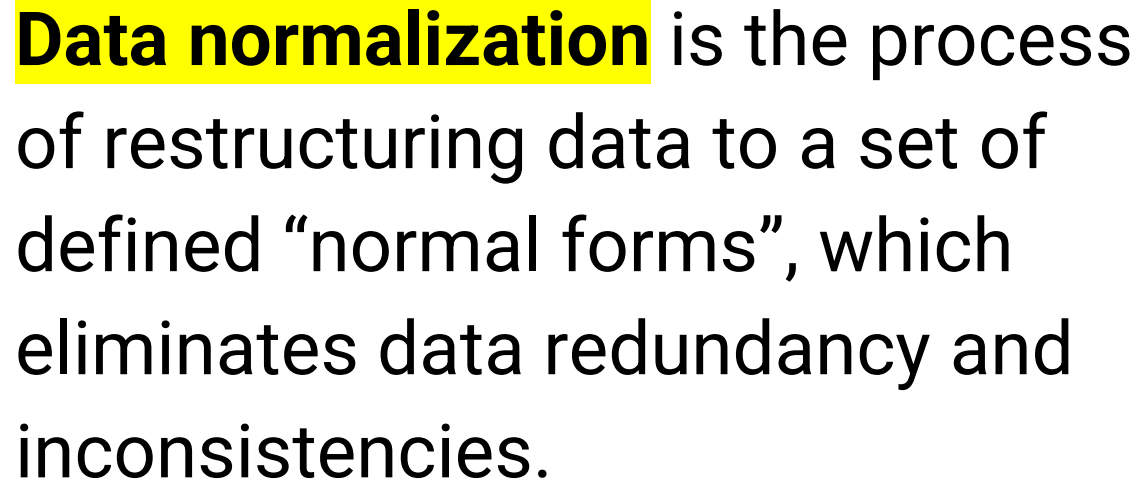
Normalize data.

Identify data relationships.

Create visual representations of a database through ERDs.

Data Normalization

Data normalization is the process of restructuring data to a set of defined "normal forms", which eliminates data redundancy and inconsistencies.

Data Normalization

Three most common forms:

01 First normal form (1NF)

02 Second normal form (2NF)

03 Third normal form (3NF)

Scenario

Patient comes to doctors office gets standard physical leaves

- What information should we collect?
    - Patient Name (First, Last)
    - Patient Address
    - Patient Insurance Information
    - Patient Payment Information
    - Tests Performed

Single Table

Patient Name	Date of Service	Patient Insurance Company	Patient Insurance ID	Patient Credit Card #	Services Performed
Sophie Timms	2021-02-13	BCBS	12930129	5011239023203891	Blood Sample, Urine Sample, Lab Work
Alex Ferguson	2022-03-10	UHC	095324	4998230920129201	Physical, EKG
Sophie Timms	2022-04-30	BCBS	12930129	5011239023203891	Physical

```
CREATE TABLE VISITS(
             PATIENT_NAME TEXT,
             DATE_OF_SERVICE DATE,
             PATIENT_INSURANCE_COMPANY TEXT,
…

)
PRIMARY KEY (PATIENT_NAME, DATE_OF_SERVICE)
```

First normal form (1NF)

Each field in a table row should contain a single value.

Patient Name	Services Performed	Date of Service	Patient Card #	Patient Insurance Company	Patient Insurance ID
Sophie Timms	Blood Sample, Urine Sample, Lab Work	2021-02-13	5011239023 203891	BCBS	12930129
Alex Ferguson	Physical, EKG	2022-03-10	4998230920 129201	UHC	095324
Sophie Timms	Physical	2022-04-30	5011239023 203891	BCBS	12930129

Patient Name	Services Performed	Date of Service	Patient Card #	Patient Insurance Company	Patient Insurance ID
Sophie Timms	Blood Sample	2021-02-13	50112390232038 91	BCBS	12930129
Sophie Timms	Urine Sample	2021-02-13	50112390232038 91	BCBS	12930129
Sophie Timms	Lab Work	2021-02-13	50112390232038 91	BCBS	12930129
Alex Ferguson	Physical	2022-03-10	49982309201292 01	UHC	095324
Alex Ferguson	EKG	2022-03-10	49982309201292 01	UHC	095324
Sophie Timms	Physical	2022-04-30	50112390232038 91	BCBS	12930129

Second Normal Form (2NF)

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.

Generally there could be a need to create a new table

First normal form (1NF)

Each field in a table row should contain a single value.

Patient Name	Services Performed	Date of Service	Patient Card #	Patient Insurance Company	Patient Insurance ID
Sophie Timms	Blood Sample	2021-02-13	5011239023203891	BCBS	12930129
Sophy Timms	Urine Sample	2021-02-13	5011239023203891	BCBS	12930129
Sofie Timms	Lab Work	2021-02-13	5011239023203891	BCBS	12930129
Alex Ferguson	Physical, EKG	2022-03-10	4998230920129201	UHC	095324
Alex Ferguson	Physical, EKG	2022-03-10	4998230920129201	UHC	095324
Sofie Timms	Physical	2022-04-30	5011239023203891	BCBS	12930129

Second Normal Form (2NF)

Adds a Primary Key, and all columns are directly dependent on that key. To transform the data we create **separate tables**

PATIENT ID	Patient Name	INSURANCE ID	Credit Card #
1	Sofie Timms	A	5011239023203891
2	Alex Ferguson	B	4998230920129201

INSURANCE ID	Company Name	Member ID
A	BCBS	12930129
B	UHC	095324

Performed Procedure ID	Patient ID	Date of Service	Service Performed Name
1	1	2021-02-13	Blood Sample
2	1	2021-02-13	Urine Sample
3	1	2021-02-13	Labwork
4	2	2022-03-10	Physical
5	2	2022-03-10	EKG
6	1	2022-04-30	Physical

Activity: Pet Normalizer

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

Suggested Time:

15 minutes

Activity: Pet Normalizer

Instructions	In pgAdmin, create a new database called `pets_db`.
	Use Excel to get the data into 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.

Foreign Keys

Foreign Keys

Foreign Keys reference the primary key of another table.

Can have a different name. It does not have to be unique.

Primary Key

	A	B
1	family_id	family
2	1	Smiths
3	2	Jones

Primary Key　　**Foreign Key**

	A	B	C
1	child_id	family_id	children
2	11	1	Chris
3	22	1	Abby
4	33	1	Suzy

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

Instructions	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.
	Make sure all tables have primary keys that increment with each new row of data.

Time's Up! Let's Review.

Data Relationships

Data Relationships
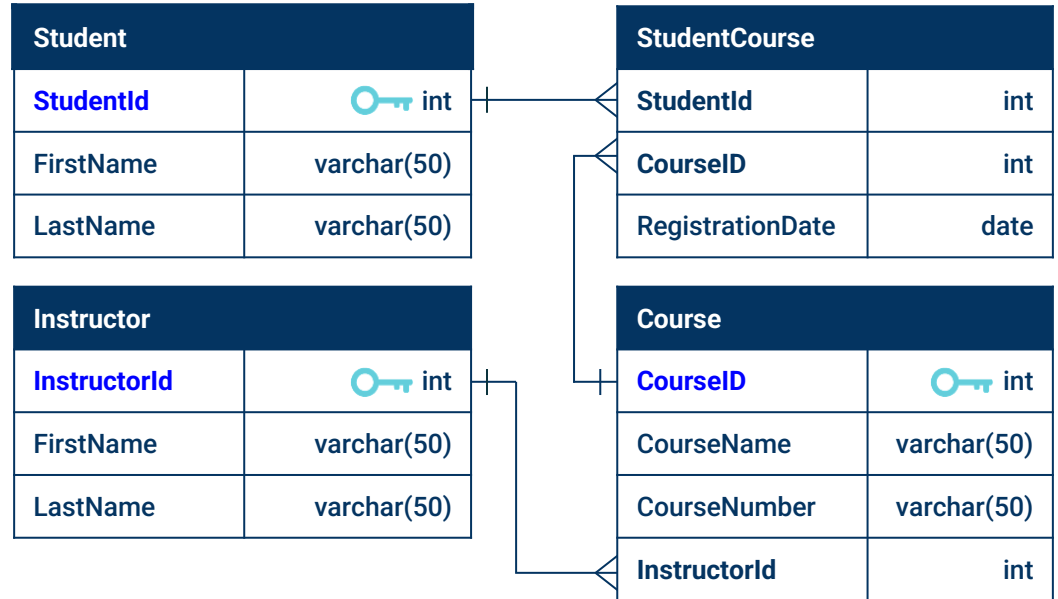
Relationships Link Tables/Entities.

Types of relationships:

01 One-to-One

02 One-to-Many

03 Many-to-Many

Student		
StudentId	🔑	int
FirstName		varchar(50)
LastName		varchar(50)

StudentCourse	
StudentId	int
CourseID	int
RegistrationDate	date

Instructor		
InstructorId	🔑	int
FirstName		varchar(50)
LastName		varchar(50)

Course		
CourseID	🔑	int
CourseName		varchar(50)
CourseNumber		varchar(50)
InstructorId		int

One-to-One Relationship

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

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

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.

One-to-Many Relationship

This example has two tables. The first table lists only addresses.

The second table lists each person's Social Security number and address. As before, one Social Security number is unique to one individual.

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

One-to-Many Relationship

- Each address can be associated with multiple people.

- Each person has an address.

- The two tables, joined, would look like this.

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

Many-to-Many Relationship

- Each child can have more than one parent.

- Each parent can have more than one child.

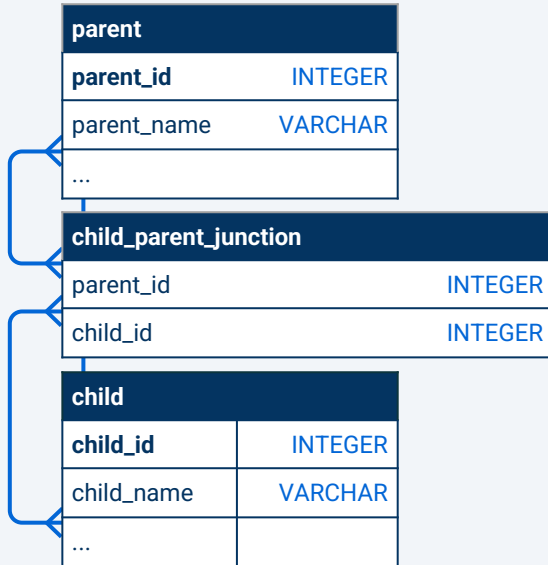| ID	Child	ID	Parent
1	Bart	11	Homer
2	Lisa	12	Marge
3	Maggie		

Many-to-Many Relationship

- 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**.

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

Junction Table

The junction table contains many parent_id's and many child_id's.

parent	
parent_id	INTEGER
parent_name	VARCHAR
...	

child_parent_junction	
parent_id	INTEGER
child_id	INTEGER

child	
child_id	INTEGER
child_name	VARCHAR
...	

	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

Instructions

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

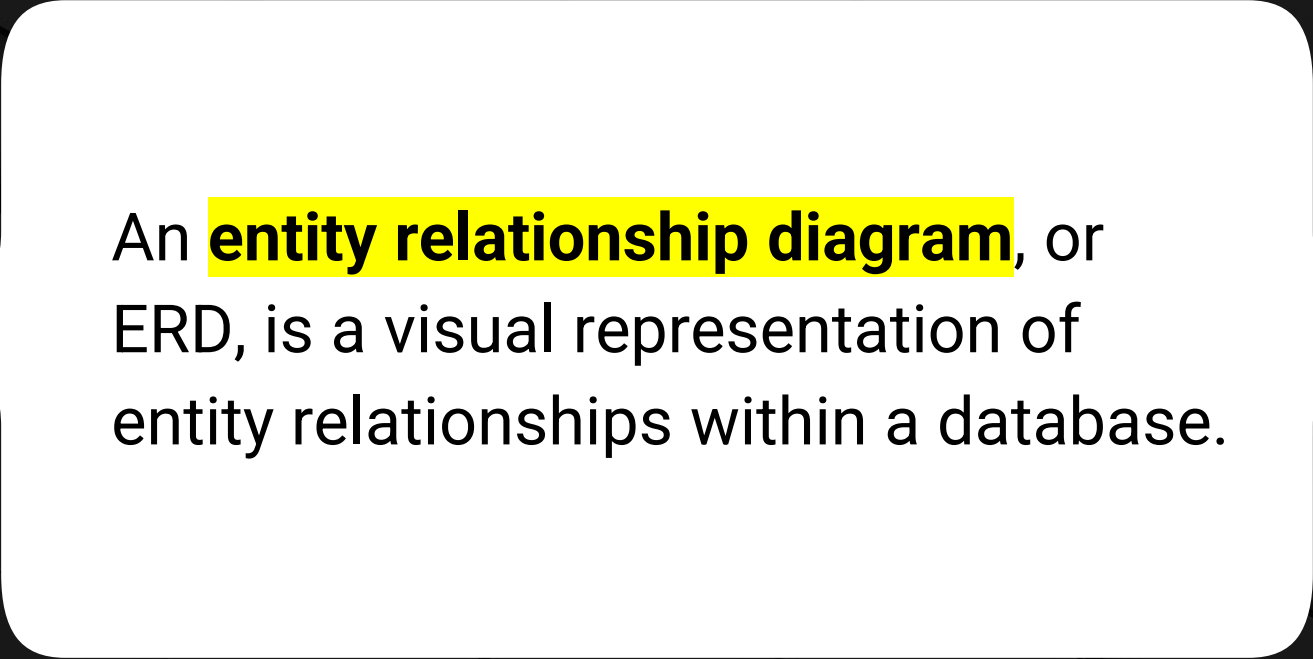Make sure all tables have primary keys that increment with each new row of data.
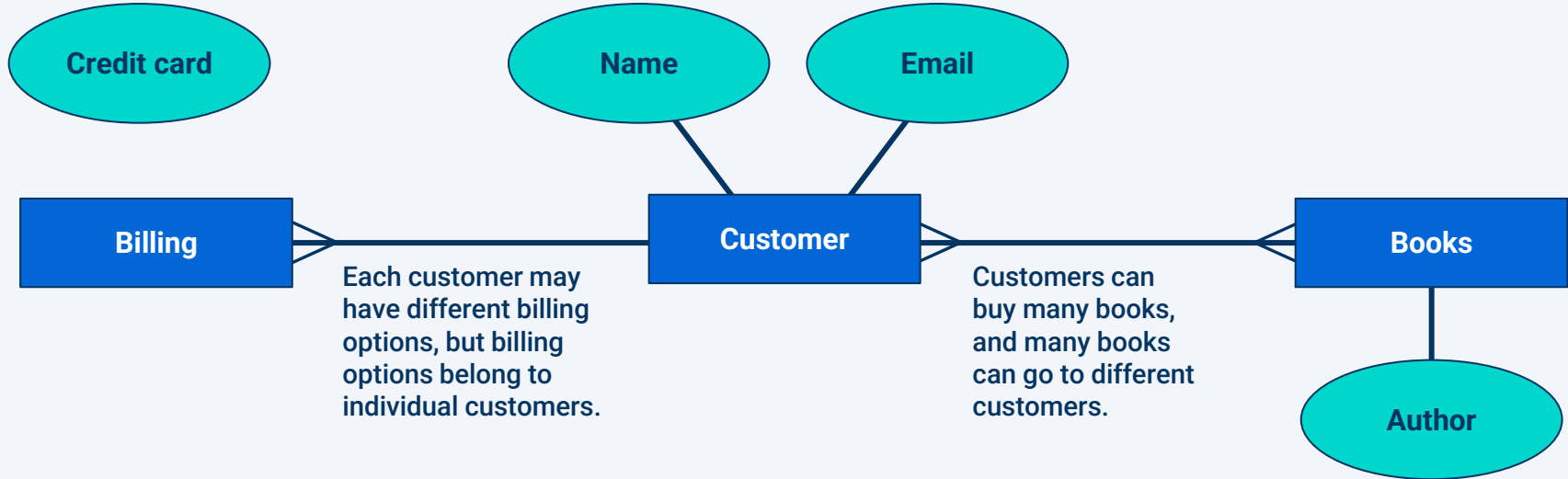
Time's Up! Let's Review.

Entity Relationship Diagrams

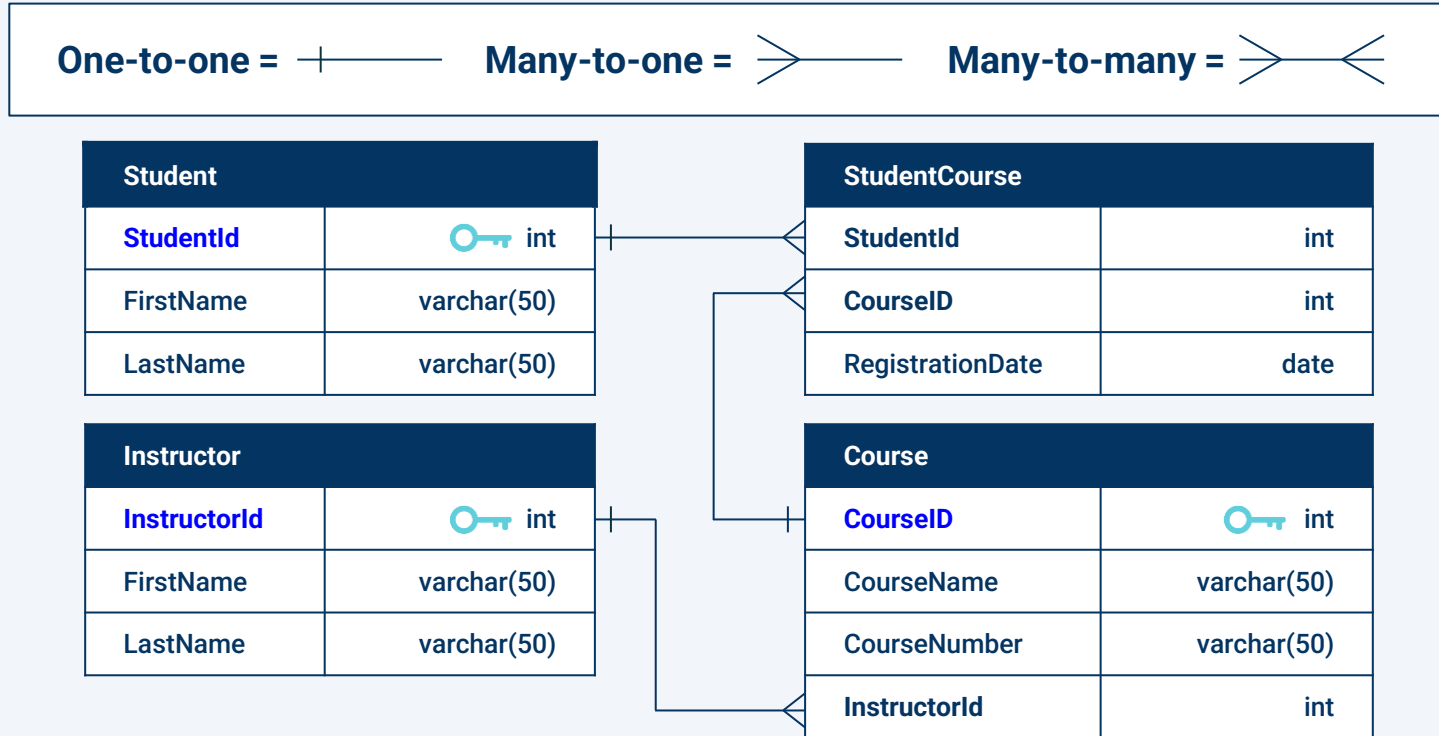An **entity relationship diagram**, or ERD, is a visual representation of entity relationships within a database.

Entity Relationship Diagram

ERD uses the following notations to create the relationships.

One-to-one = —————— Many-to-one = ——> Many-to-many = —>—<— | Entity | (Attribute)

(Credit card)   (Name)   (Email)

| Billing | ——> | Customer | ——< | Books |

Each customer may have different billing options, but billing options belong to individual customers.

Customers can buy many books, and many books can go to different customers.

(Author)

Entity Relationship Diagram

A typical ERD design.

Entity Relationship Diagram

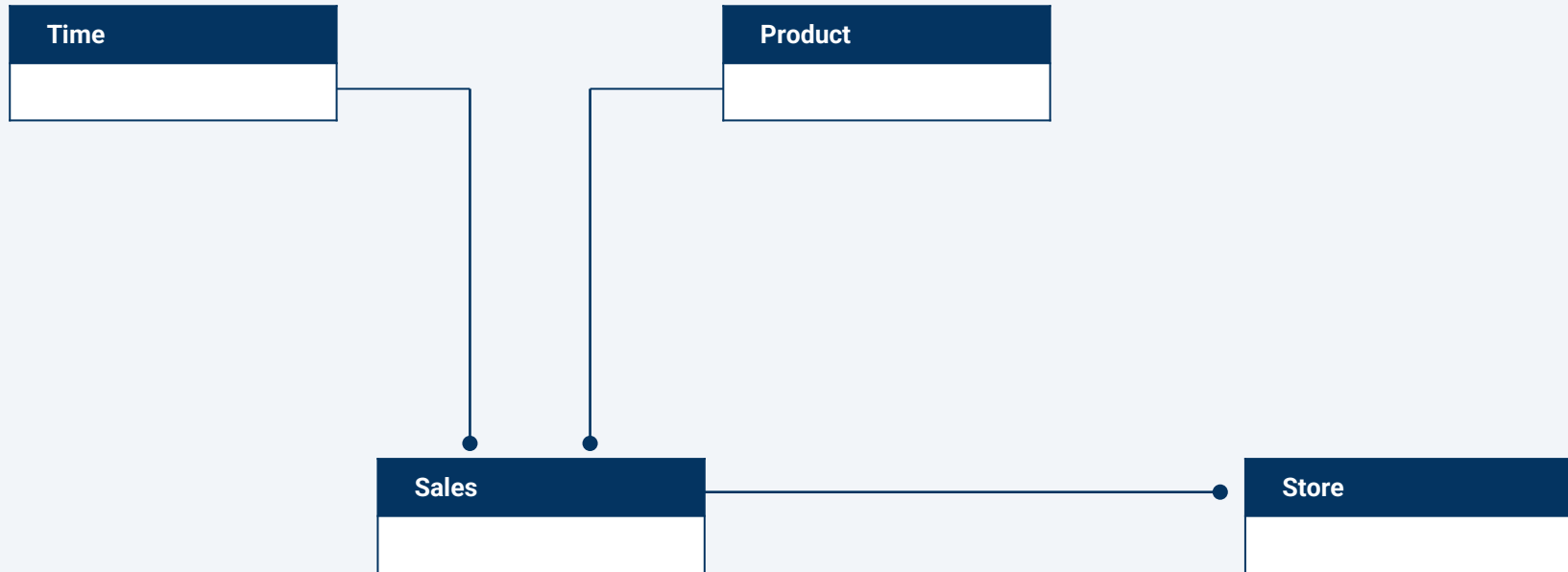Three Types of ERDs or Data Models

01 Conceptual Model Design
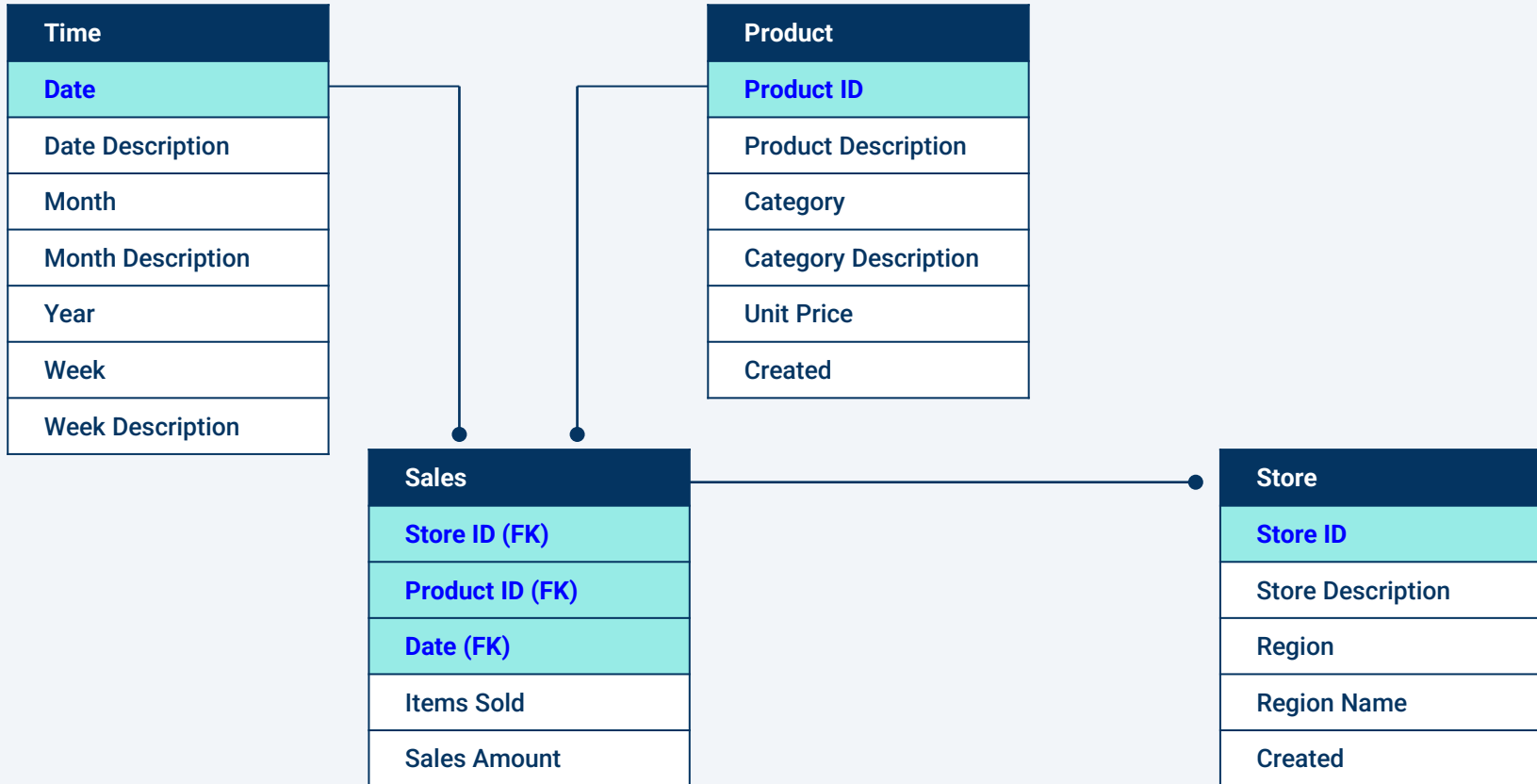
02 Logical Model Design

03 Physical Model Design

Conceptual Model Design

Logical Model Design

Time

Date
Date Description
Month
Month Description
Year
Week
Week Description

Product

Product ID
Product Description
Category
Category Description
Unit Price
Created

Sales

Store ID (FK)
Product ID (FK)
Date (FK)
Items Sold
Sales Amount

Store

Store ID
Store Description
Region
Region Name
Created

Physical Model Design

Time

Date_ID: Integer	
Date_Desc: Varchar(30)	
Month_ID: Integer	
Month_Desc: Varchar(30)	
Year: Integer	
Week_ID: Integer	
Week_Desc: Varchar(30)	

Product

Product_ID: Integer	
Product_Desc: Varchar(50)	
Category_ID: Integer	
Category_Desc: Varchar(50)	
Unit_Price: Float	
Created: Data	

Sales

Store_ID: Integer	
Product_ID: Integer	
Date_ID: Integer	
Items_Sold: Integer	
Sales_Amount: Float	

Store

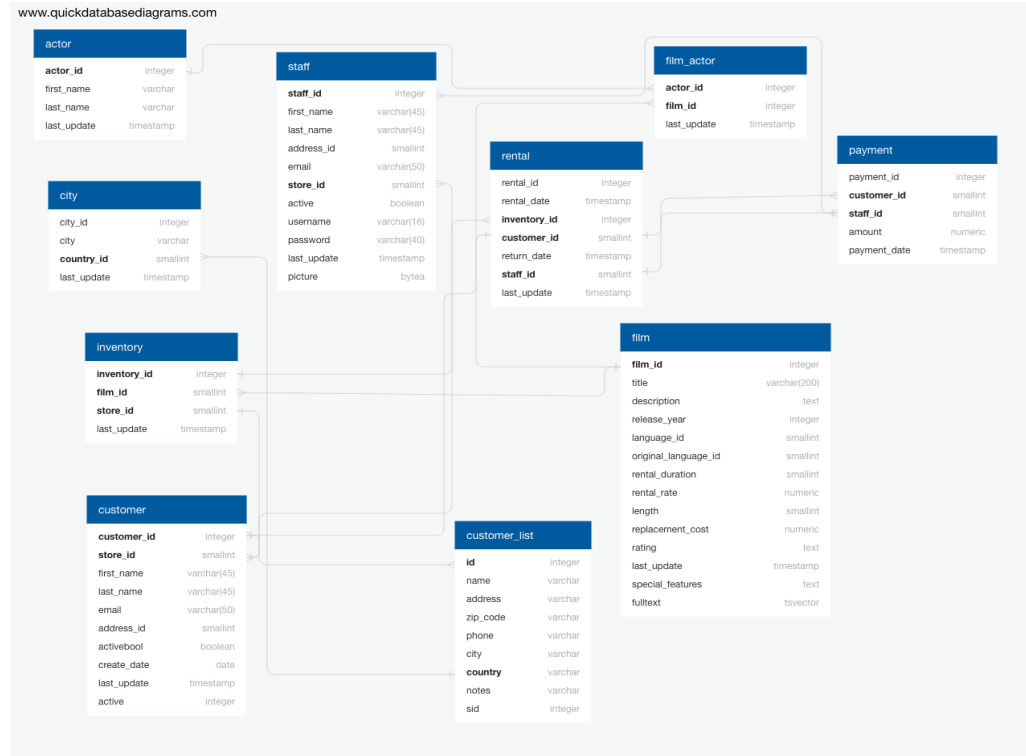|	
Store_ID: Integer	
Store_Desc: Varchar(50)	
Store_Desc: Integer	
Region_Name: Varchar(50)	
Created: Date	

Entity Relationship Diagram

An ERD illustrates entities, their data types, and relationships.

Partner Activity: Designing an ERD, Part 1

In this activity, you will create a conceptual ERD for a gym owner.

Suggested Time:

15 minutes

Partner Activity: Designing an ERD, Part 1

Instructions	
Instructions You are meeting with a gym owner who wants to organize his data in a database. Create a conceptual ERD for the gym owner.	Determine 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.
	Which data model is appropriate here: one to one, one to many, or many to many?
Hint	Check Slack for the **documentation** for more in-depth explanations of entity relationship diagrams.

Time's Up! Let's Review.

Partner Activity: Designing an ERD, Part 2

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.

Suggested Time:

15 minutes

Partner Activity: Designing an ERD, Part 2

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:

_	one TO one	0_	zero or one TO one
_<	one TO many	0_0	zero or one TO zero or one
>_	many TO one	_0<	one TO zero or many
>_<	many TO many	>0_	zero or many TO one
_0	one TO zero or one		

Partner Activity: Designing an ERD, Part 2

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;
```

Time's Up! Let's Review.

Instructor Demonstration

Introduction to Unions

Activity: Unions

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

Suggested Time:

15 minutes

Activity: Unions

Instructions	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.