



**PLURALSIGHT**

# SQL Foundations

Week 3



Proprietary and confidential

# STANDUP

**What was one success you had this week?**

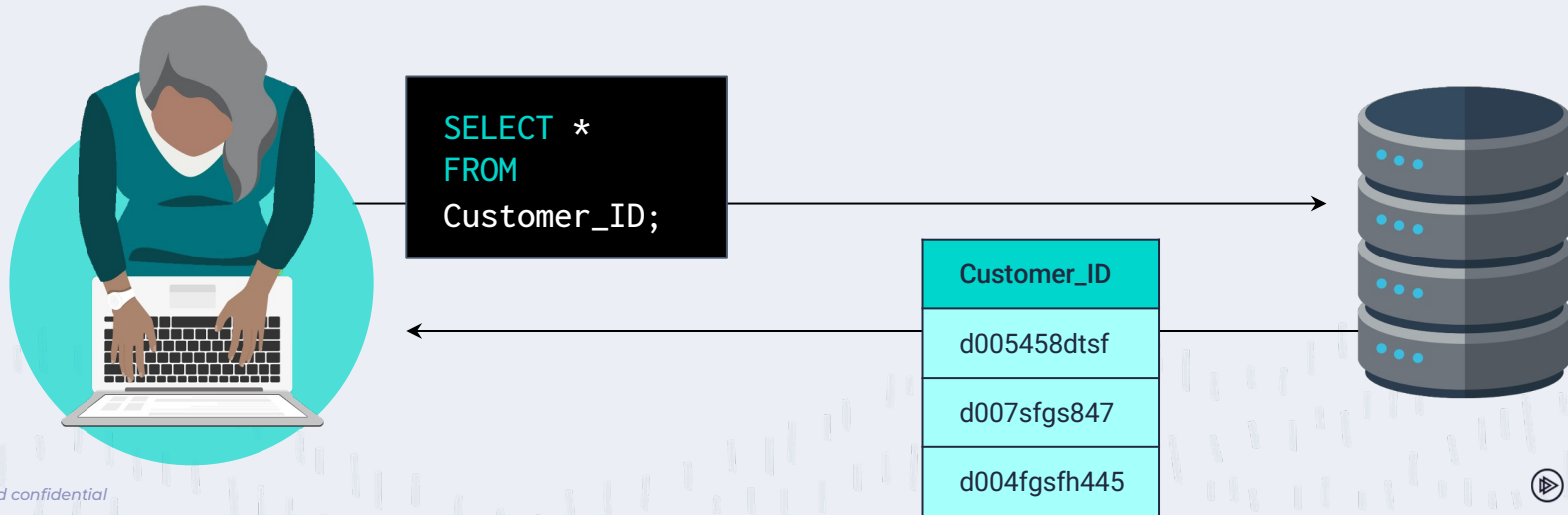
**What is something you are still struggling with?**

# SQL QUERY STRUCTURE

# What is SQL

SQL (often pronounced "sequel") stands for Structured Query Language.

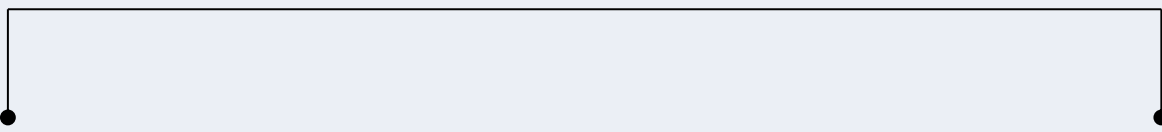
It is a powerful tool that enables programmers to create, populate, manipulate, and access databases. It also provides an easy method for dealing with server-side storage.



# What is SQL

Data using SQL is stored in tables on the server, much like spreadsheets you would create in Microsoft Excel.

This makes the data easy to visualize and search.



Customer_ID	Date_ID
d005458dtsf	6/26/2019
d007sfgs847	8/3/2018
d004fgsfh445	12/3/2018

Order_ID	Customer_ID	Date_ID
10001	d005458dtsf	6/26/2019
10002	d007sfgs847	8/3/2018
10003	d004fgsfh445	12/3/2018

# Using CRUD

**Create Read Update Delete** is a set of operations used with persistent storage.

Create	Create data in a table with the <b>INSERT</b> statement.
Read	Read data by using <b>SELECT</b> .
Update	Updated a table's data by using <b>UPDATE</b> .
Delete	Deleted data via <b>DELETE</b> .

# SQL Joins

INNER JOIN	Returns records that have matching values in both tables.
LEFT JOIN	Returns all records from the left table and the matched records from the right table.
RIGHT JOIN	Returns all records from the right table and the matched records from the left table.
CROSS JOIN	Returns records that match every row of the left table with every row of the right table. This type of join has the potential to make very large tables.
FULL OUTER JOIN	Places null values within the columns that do not match between the two tables, after an inner join is performed.



# ANATOMY OF SQL

- SELECT** Used to specify the **columns** or expression you want to retrieve from the database
- FROM** Used to specify the **table** from which you want to retrieve the data
- WHERE** An optional clause used to filter the rows (data) returned
- ORDER BY** An optional clause used to sort the rows returned by the query based on one or more columns.

# ANATOMY OF SQL

## What is this query retrieving?

```
1 | SELECT FIRST_NAME, LAST_NAME, AGE  
2 | FROM EMPLOYEE
```

```
1 | SELECT FIRST_NAME, LAST_NAME, AGE  
2 | FROM EMPLOYEE  
3 | ORDER BY FIRST_NAME ASC
```

```
1 | SELECT FIRST_NAME, LAST_NAME, AGE  
2 | FROM EMPLOYEE  
3 | ORDER BY FIRST_NAME DESC
```

```
1 | SELECT FIRST_NAME, LAST_NAME, AGE  
2 | FROM EMPLOYEE  
3 | WHERE AGE > 18
```

# Order By Aggregates

---

The **ORDER BY** function:



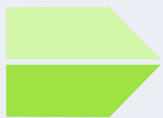
Is added towards the end of a query.



Returns in an ascending order by default.



Can also return in a descending order by adding **DESC**.



Can limit the return by adding **LIMIT**.



**NOTE:** Use the **ROUND** function to round up the number after the the decimal.

---

## Wildcard: % and \_

---

Use wildcards to substitute zero, one, or multiple characters in a string. The keyword **LIKE** indicates the use of a wildcard.

```
SELECT *  
FROM actor  
WHERE last_name LIKE 'Will%';
```

---

## Wildcard: % and \_

---

The **%** will substitute zero, one, or multiple characters in a query.  
In this example, all of the following are matches: Will, Willa, and Willows.

```
SELECT *  
FROM actor  
WHERE last_name LIKE 'Will%';
```

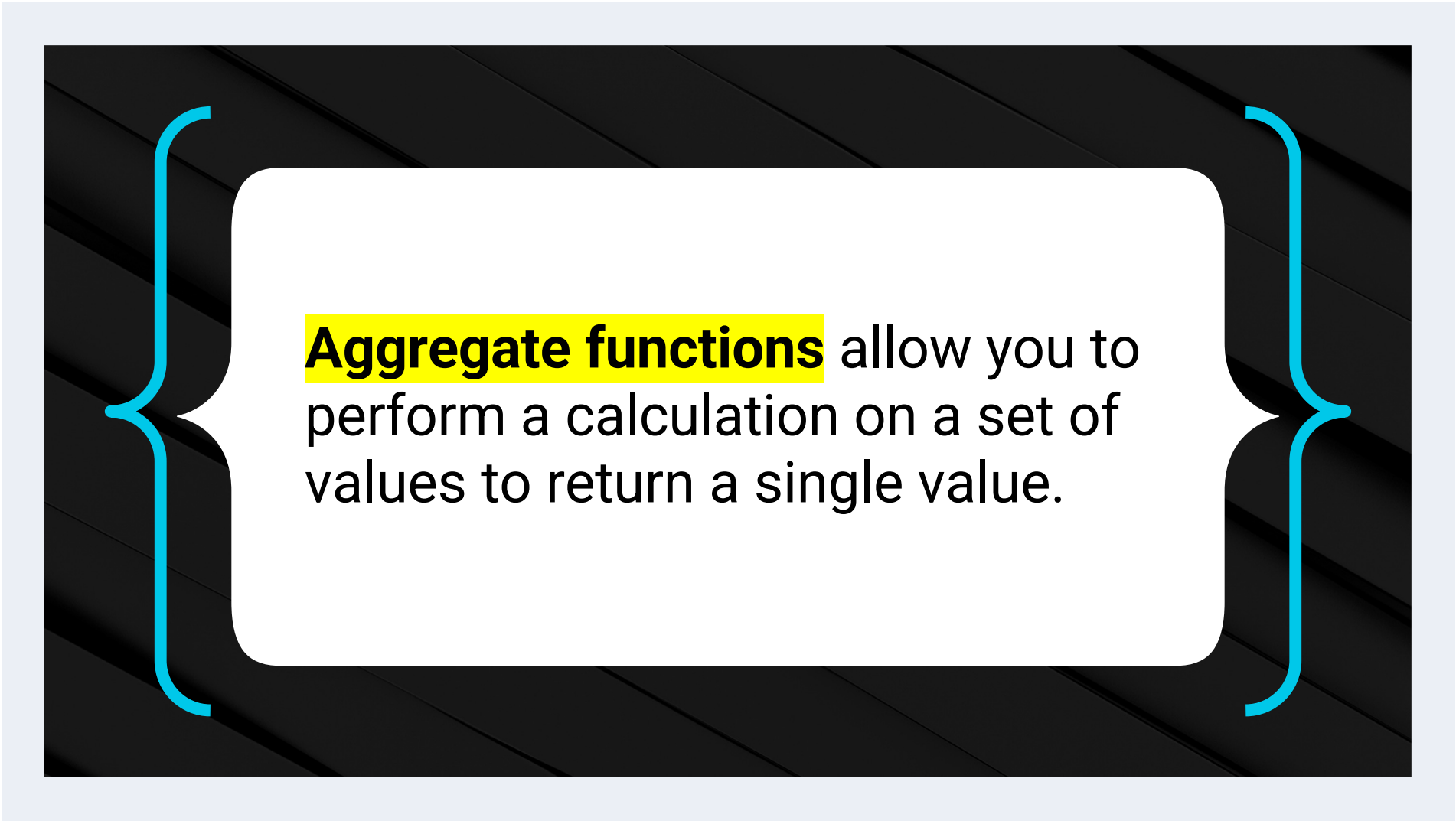
## Wildcard: % and \_

---

The **\_** will substitute only **one** character in a query.

**\_an** returns all actors whose first name contains three letters, the second and third of which are **an**.

```
SELECT *  
FROM actor  
WHERE first_name LIKE '_an';
```



**Aggregate functions** allow you to perform a calculation on a set of values to return a single value.

# Aggregate Functions

---

The most commonly used aggregate functions are:

AVG	Calculates the average of a set of values
COUNT	Counts the rows in a specific table or view
MIN	Returns the minimum value in a set of values
MAX	Returns the maximum value in a set of values
SUM	Calculates the sum of a set of values

---



# Aggregate Functions

---

Aggregate functions are often used with:

01

The **GROUP BY** clause

02

The **HAVING** clause

03

The **SELECT** statement

---

# Introduction to Subqueries

## Subqueries

---

A subquery is nested inside a larger query. Subqueries occur in:

01

The **SELECT** statement

02

The **FROM** clause

03

The **WHERE** clause

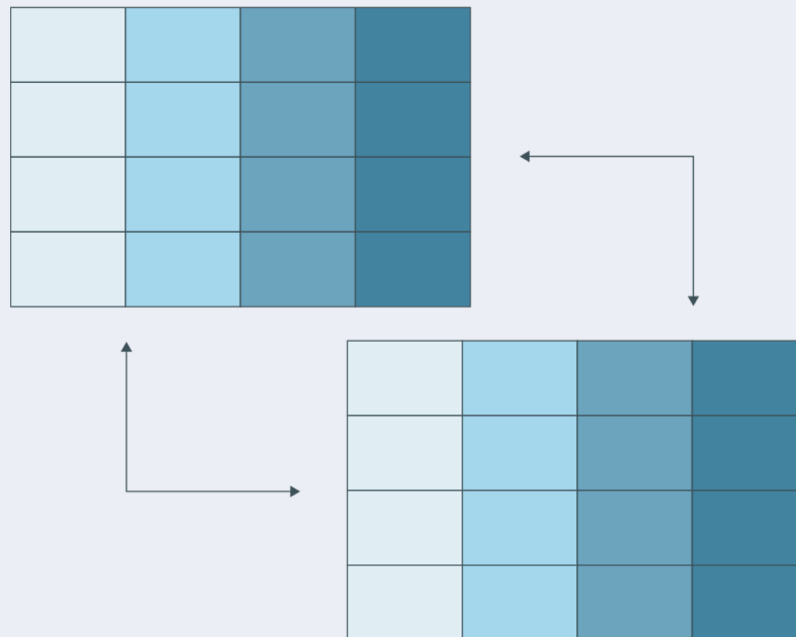
---

# SQL Views

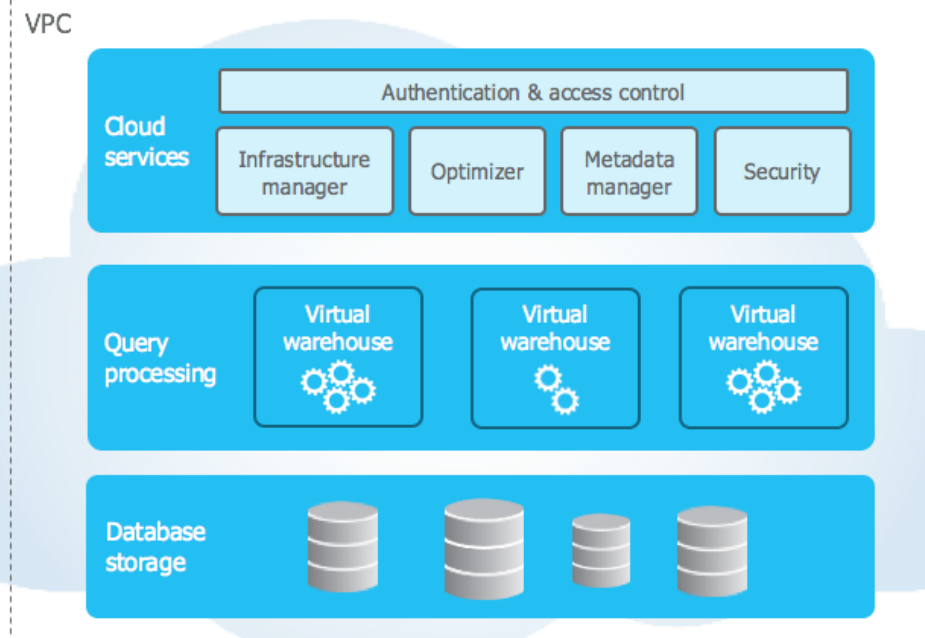
---

Views are created by using the `CREATE VIEW` statement.

Views are created from a single table, multiple tables, or another view.



# Snowflake Architecture



# Saving Query and Query Results

## Standard or Permanent Tables

- Standard tables are permanent and provide full data protection and recovery capabilities, including time travel and fail-safe.
- **Used for storing persistent data.**

## Temporary Tables

- These are session-specific, meaning they last only for the duration of your session. They can be useful for intermediate computations.
- **Useful for storing intermediate results and data that do not need to persist.**

## Transient Tables

- Transient tables are similar to regular tables, but they don't have the fail-safe (historical data) feature, meaning Snowflake doesn't keep the historical data of the table for restoring. This might save on storage costs.
- **Suitable for staging data or storing data that can be easily re-created.**

## External Tables

- External tables allow querying data stored in external stages (like S3 buckets) without loading it into Snowflake.
- **Useful for querying large datasets stored outside of Snowflake.**

# Saving Query and Query Results

## Standard Views

- A view creates a virtual table based on the result set of a SQL statement.
- They don't store data physically but represent data that is stored in tables.
- **Useful for simplifying complex queries and creating reusable query logic.**

## Materialized Views

- A materialized view stores the result of a query physically, and it gets refreshed based on the criteria you set (on each transaction or on a set schedule).
- They are useful when you have complex calculations that you don't want to run every time you query the data.
- Useful for improving performance of frequently run queries.

## Secure Views

- Secure views hide the underlying query logic and data from unauthorized users.
- Useful for sharing data securely with external or internal users.

# Data Modeling Defined

## What is data modeling?

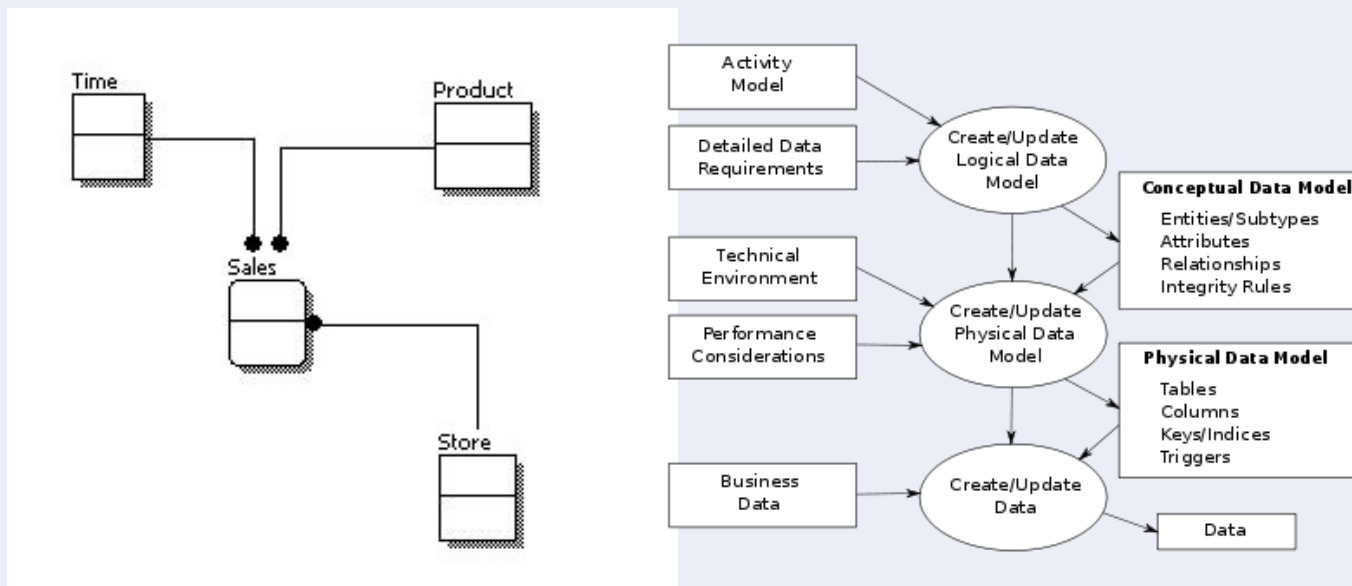
“A set of symbols and text which precisely explain a subset of real information.”

-Steve Hoberman and George McGeachie (Technics, 2011)

**Data modeling** is often the first step in database design and object-oriented programming . It consists of a conceptual model of how data items relate to each other.



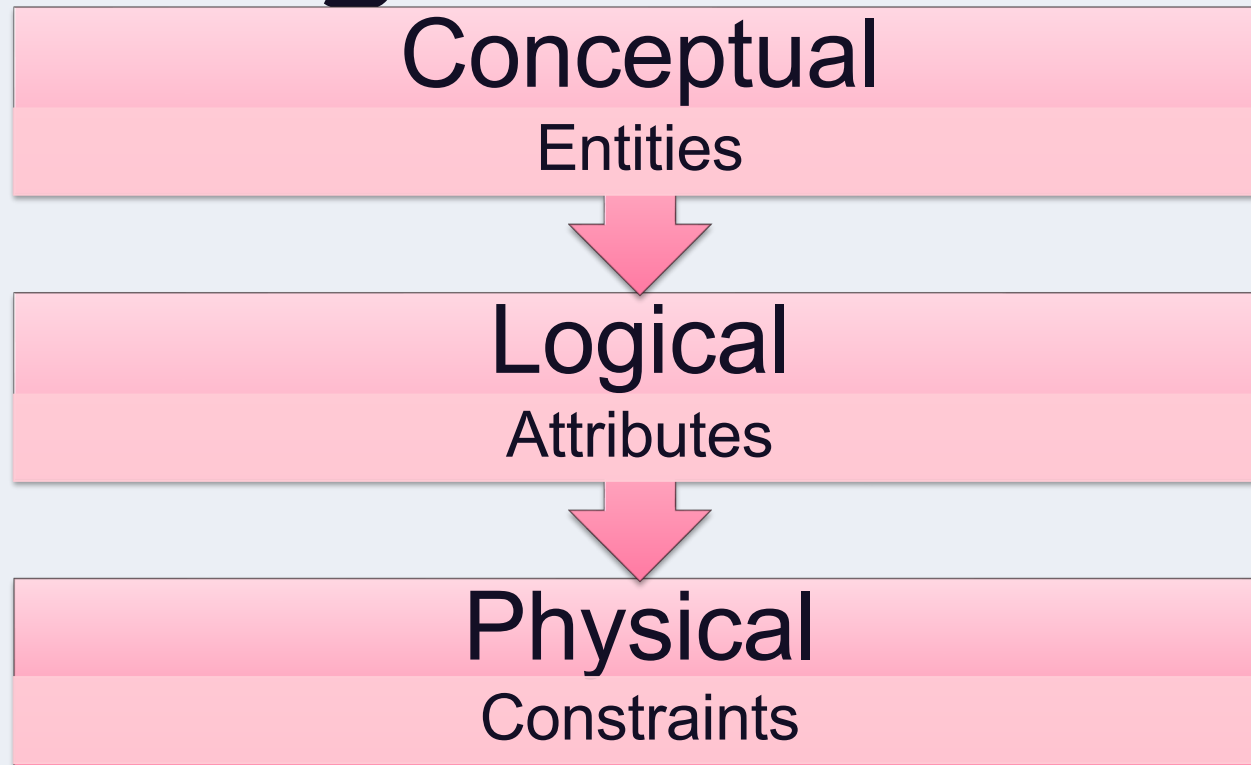
# What is Data Modeling?



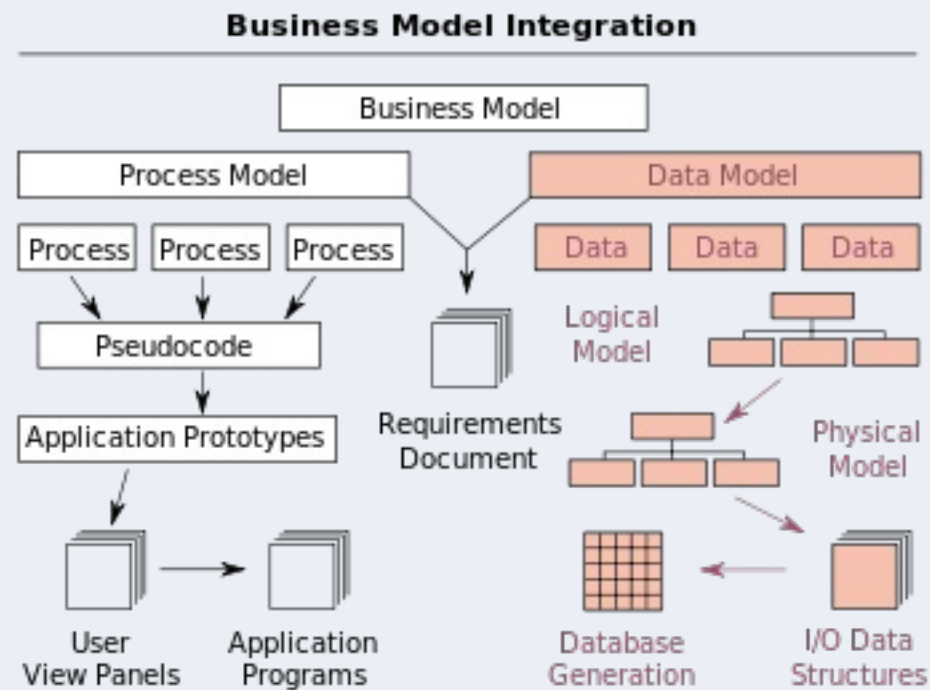
# **Data Modeling Forms and Standards**

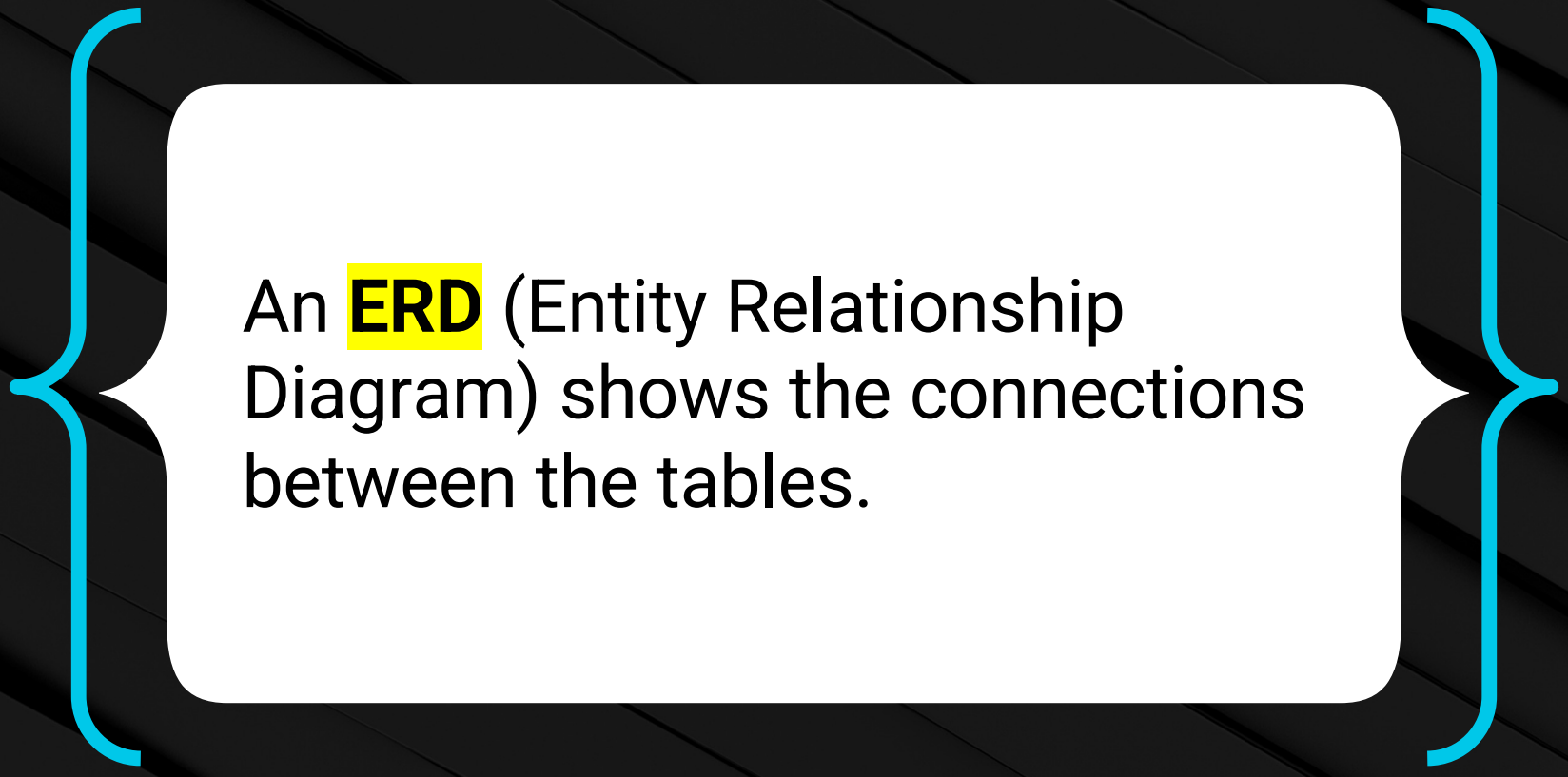
- Helps business and IT users relate to one another
- Facilitates the management of data as a resource
- Improves and expedites the integration of information systems
- Informs the designing databases/data warehouses (aka data repositories)

# Modeling



# Modeling Evolution





An **ERD** (Entity Relationship Diagram) shows the connections between the tables.

# Entity Relationship Diagram

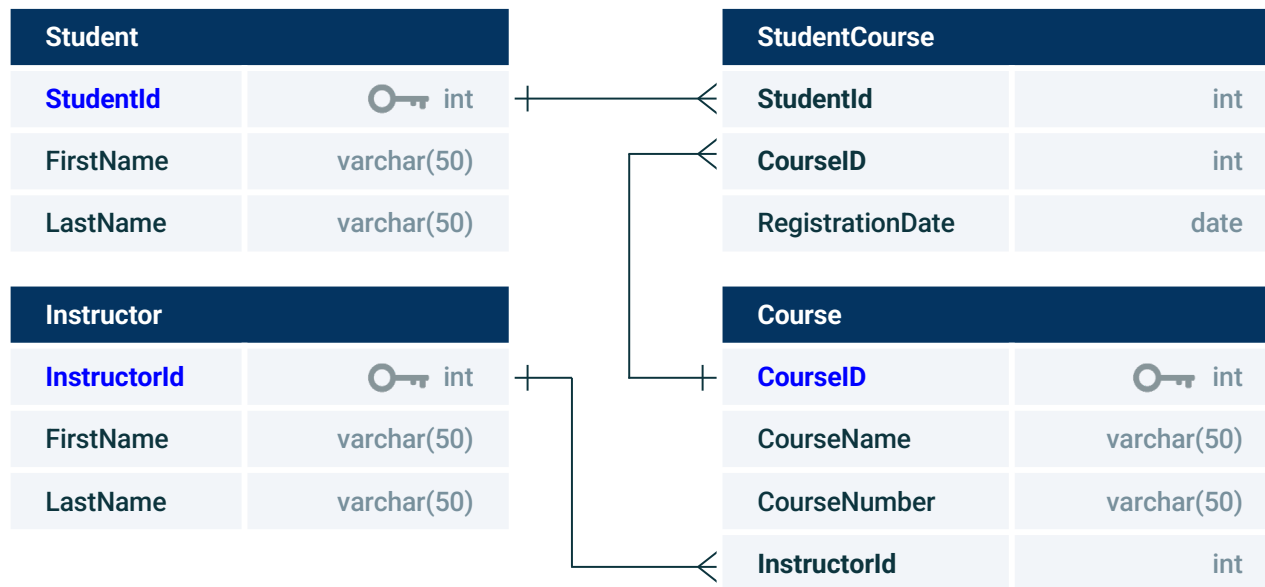
It's called an Entity Relationship Diagram because it shows:



The entities in your data model



The relationship between entities

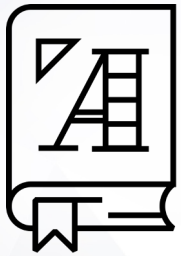


The schema makes it easier to identify the tables we need as well as the keys we will use to link our subqueries.

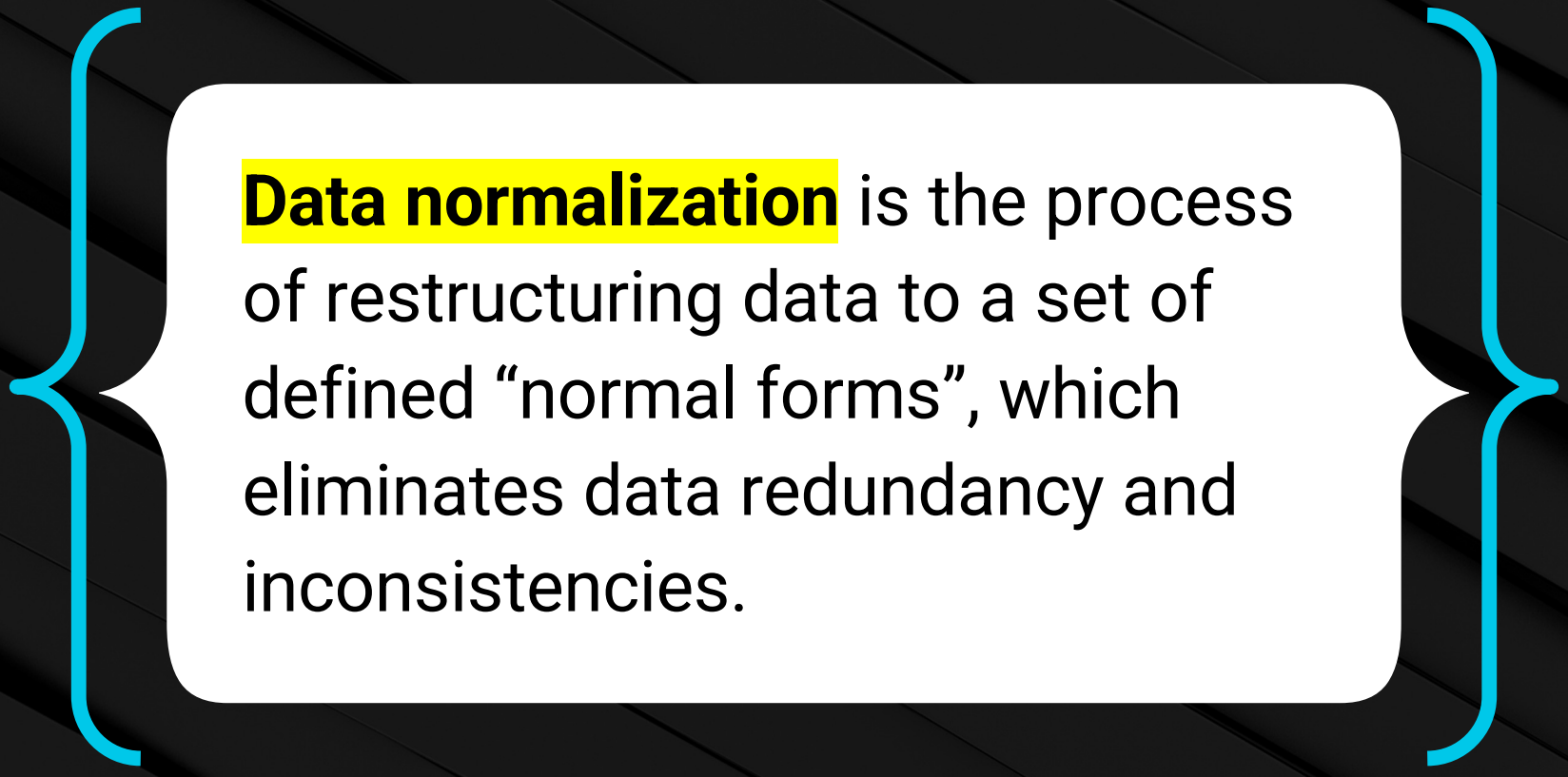


# Data Normalization





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



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



The process of data normalization **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)

# First normal form (1NF)

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

## Raw Data

VIN	Services Performed	Customer Name	Model
3D7LX39C86G106066	Oil Change	Marcia Jackson	Prius
1FAFP33Z24W147740	Oil Change, Alignment	Patricia Smith	Equinox
KNDJD736875757954	Oil Change, Brake Replacement	Mikhail Ivanov	CRV
3N1AB7AP7EL611028	Transmission Rebuild	Lucas Gonzalez	Tahoe

## Normalization

### First Normal Form (Each row is unique)

VIN	Services Performed	Customer Name	Salutation
3D7LX39C86G106066	Oil Change	Marcia Jackson	Prius
1FAFP33Z24W147740	Oil Change	Patricia Smith	Equinox
1FAFP33Z24W147740	Alignment	Patricia Smith	Equinox
KNDJD736875757954	Oil Change	Mikhail Ivanov	CRV
KNDJD736875757954	Brake Replacement	Mikhail Ivanov	CRV
3N1AB7AP7EL611028	Transmission Rebuild	Lucas Gonzalez	Tahoe

# First Normal Form (1NF)

---



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



Each row is unique.

- Rows can have fields that repeat.
- Whole rows do not fully match.

**Raw Data**

family	children
Smith	Chris, Abby, Susy
Jones	Steve, Mary, Dillion

**Normalization**



**First Normal Form**

family	children
Smith	Abby
Smith	Susy
Jones	Mary
Smith	Chris
Jones	Dillion
Jones	Mary

## Second Normal Form (2NF)

Adds a Primary Key, and all columns are directly dependent on that key. To transform the data below, we'll need separate tables for Vehicle, Customer, and Service Performed.

VIN	Services Performed	Customer Name	Model	Make
3D7LX39C86G106066	Oil Change	Marcia Jackson	Prius	Toyota
1FAFP33Z24W147740	Oil Change	Patricia Smith	Escape	Ford
1FAFP33Z24W147740	Alignment	Patricia Smith	Escape	Ford
KNDJD736875757954	Oil Change	Mikhail Ivanov	CRV	Honda
KNDJD736875757954	Brake Replacement	Mikhail Ivanov	CRV	Honda
3N1AB7AP7EL611028	Transmission Rebuild	Lucas Gonzalez	Tahoe	Chevy

2NF Normalization



## Second Normal Form (2NF)

Customer		
ID	First	Last
1	Marcia	Jackson
2	Patricia	Smith
3	Mikhail	Ivanov
4	Lucas	Gonzalez

This is the same data in 2NF; note that yellow columns are Primary Keys and blue columns are Foreign Keys which reference the Primary Keys from other tables.

Vehicle			
VIN	Customer ID	Model	Make
3D7LX39C86G106066	1	Prius	Toyota
1FAFP33Z24W147740	2	Equinox	Chevy
KNDJD736875757954	3	CRV	Honda
3N1AB7AP7EL611028	4	Tahoe	Chevy

Services		
ID	Vehicle	Service
1	3D7LX39C86G106066	Oil Change
2	1FAFP33Z24W147740	Oil Change
3	1FAFP33Z24W147740	Alignment
4	KNDJD736875757954	Oil Change
5	KNDJD736875757954	Brake Replacement
6	3N1AB7AP7EL611028	Transmission Rebuild



# Second Normal Form (2NF)

Must be in first normal form

Single column primary key

- Primary key
- Identifies the table and row uniquely

Generally, there could be a need to create a new table.

Data in 1NF

family	children
Smith	Abby
Smith	Susy
Jones	Mary
Smith	Chris
Jones	Dillion
Jones	Mary

2NF Normalization



Family Table

family_id	family
1	Smith
2	Jones

Child Table

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

## Second Normal Form (2NF)

---

Table contains a primary key.



Provides unique identifier for each row.



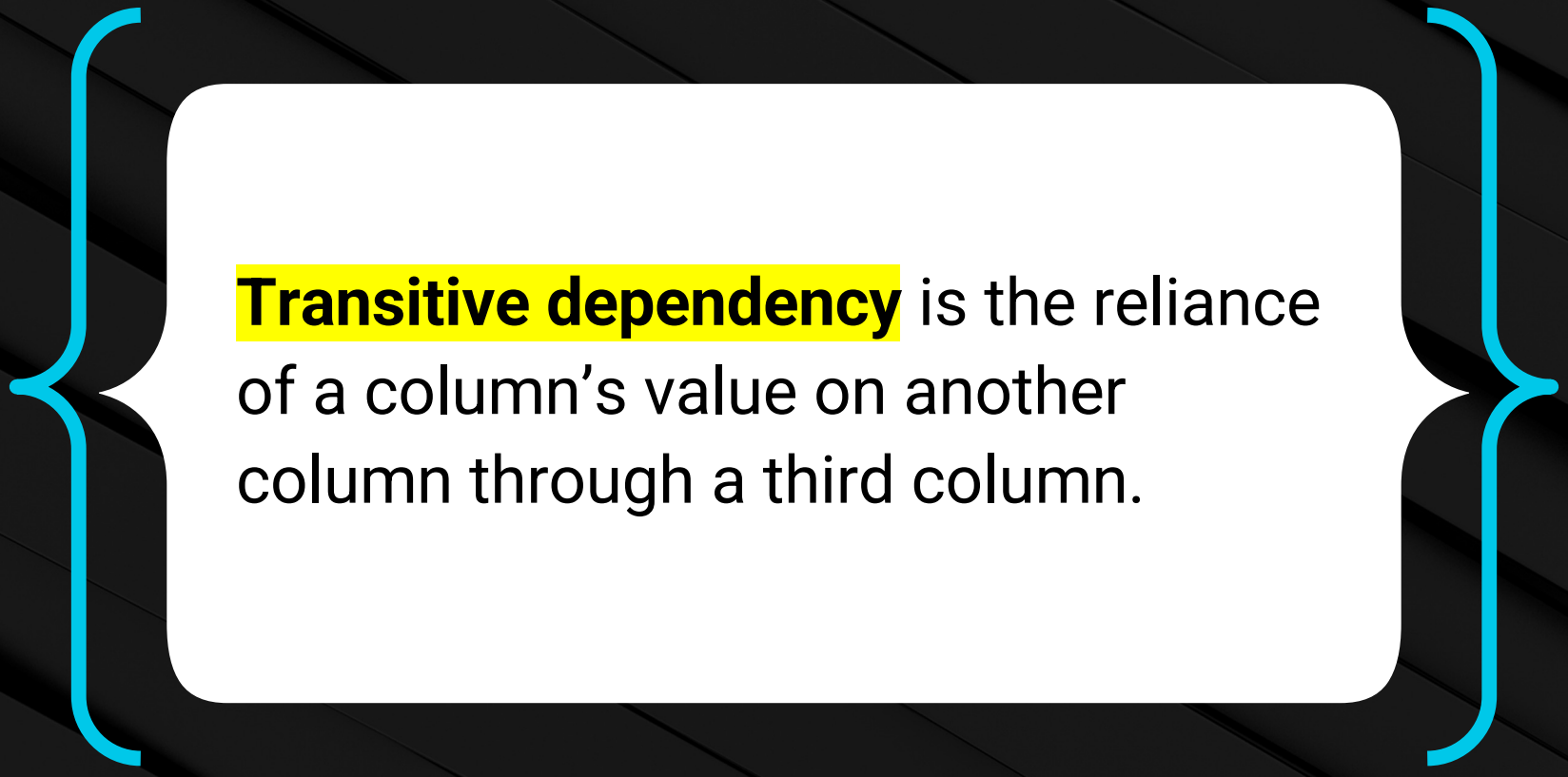
Ideally in a single column.



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



**Generally there could be a  
need to create a new table**



**Transitive dependency** is the reliance of a column's value on another column through a third column.

# Transitive Dependency

---

Transitive	If $A \Rightarrow B$ and $B \Rightarrow C$ then $A \Rightarrow C$
Dependence	<ul style="list-style-type: none"><li>• One value relies on another.</li><li>• City relies on ZIP code; age depends on birthday.</li></ul>
For example	<ul style="list-style-type: none"><li>• Consider the <b>VIN</b>, <b>Customer</b>, <b>Model</b>, and <b>Make</b> columns in the <b>Vehicle</b> table.</li><li>• <b>Customer</b> depends on <b>VIN</b>.</li><li>• <b>Model</b> depends on <b>Customer</b>.</li><li>• <b>Make</b> thus depends on <b>Model</b>.</li><li>• This is an example of transitive dependency</li></ul>

## Third Normal Form (3NF) – “Simplify the Relationships”

---

To transition to 3NF, data must be in 2NF. Additionally, no column can imply another column in the same table. For instance, “Make” is implied by “Model” in the table below. That is, a model “Prius” will always have a make of “Toyota”.

Vehicle			
VIN	Customer	Model	Make
3D7LX39C86G106066	1	Prius	Toyota
1FAFP33Z24W147740	2	Equinox	Chevy
KNDJD736875757954	3	CRV	Honda
3N1AB7AP7EL611028	4	Tahoe	Chevy

## Third Normal Form (3NF) – “Simplify the Relationships”

To break the same data into 3NF, we’d use the tables below.

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

Make	
ID	Make
1	Toyota
2	Chevy
3	Honda

Model		
VIN	Model	Make
1	Prius	1
2	Equinox	2
3	CRV	3
4	Tahoe	2

# Third Normal Form (3NF)

Must be in second normal form

Contains non-transitively dependent columns

owner_id	owner_name	owner_address	store_name
11	Marshall	123, Fake Street	Soups and Stuff
22	Susan	44, New Drive	Sink Emporium
33	Molly	99, Old Lane	Tasty Burgers

## 3NF Normalization



owner_id	owner_name	owner_address
11	Marshall	123, Fake Street
22	Susan	44, New Drive
33	Molly	99, Old Lane

store_id	store_name	Owner_id (fk)
1	Soups and Stuff	11
2	Sink Emporium	22
3	Tasty Burgers	33



# 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**

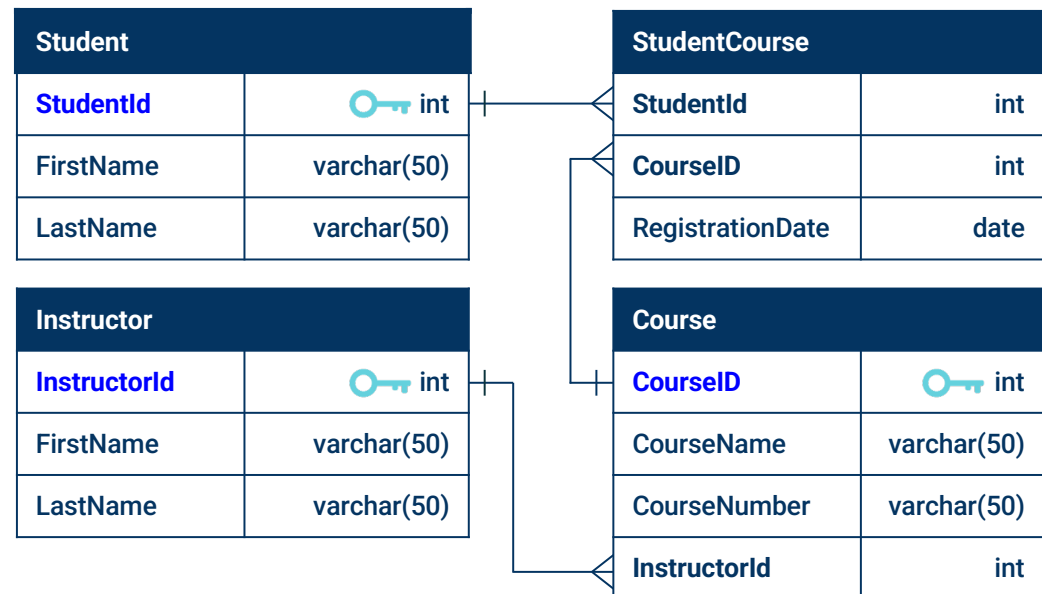
	A	B	C
1	child_id	family_id	children
2	11	1	Chris
3	22	1	Abby
4	33	1	Suzy

# Data Relationships

Relationships Link Tables/Entities.

Types of relationships:

- 01 One-to-One
- 02 One-to-Many
- 03 Many-to-Many



## 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

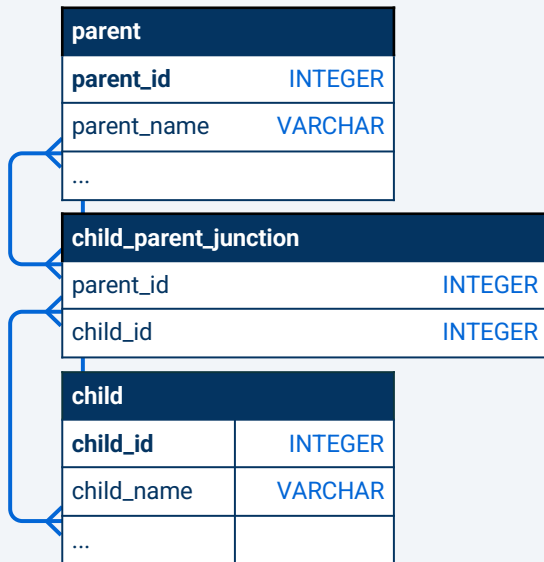
---

- 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.

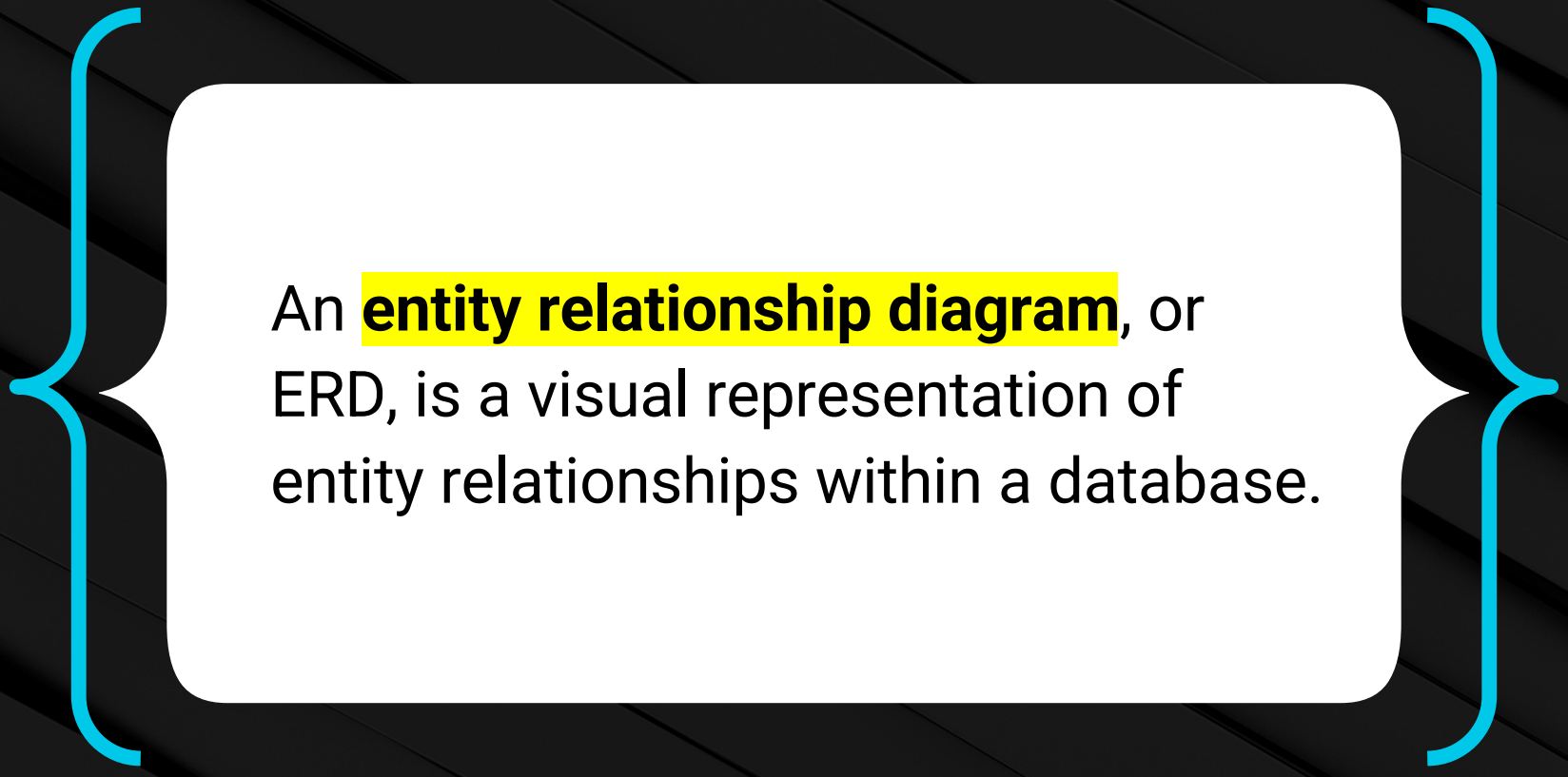


	parent_id integer	child_id integer
1	11	1
2	11	2
3	11	3
4	12	1
5	12	2
6	12	3

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





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.

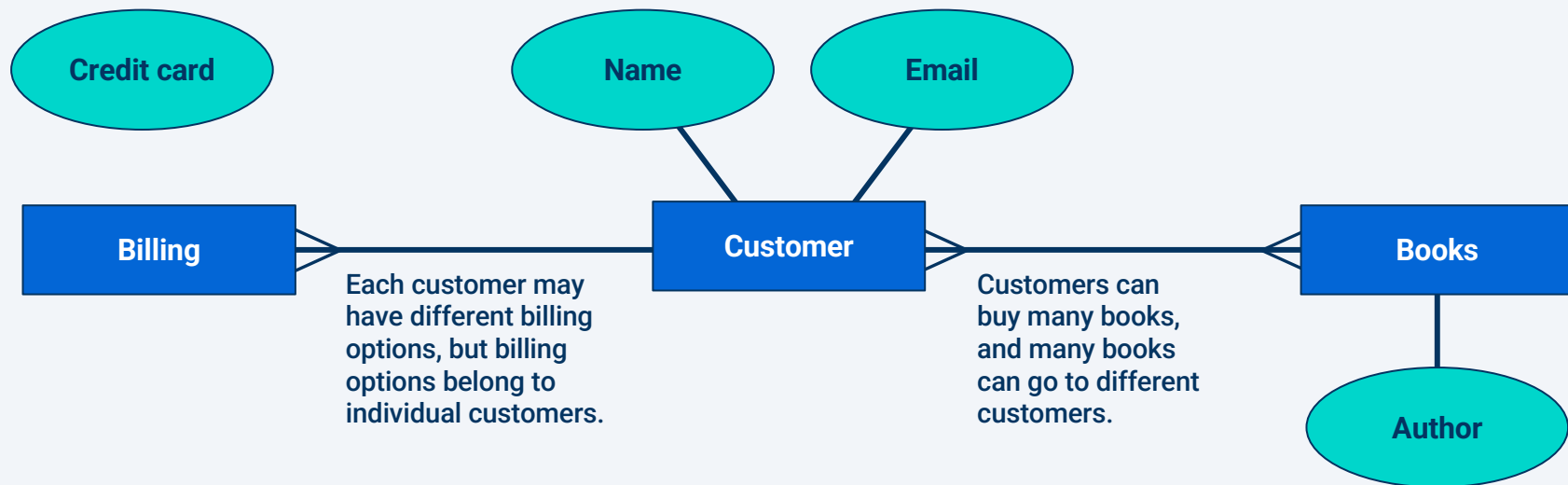
One-to-one = ———

Many-to-one = ———>

Many-to-many = ———>———<




Entity

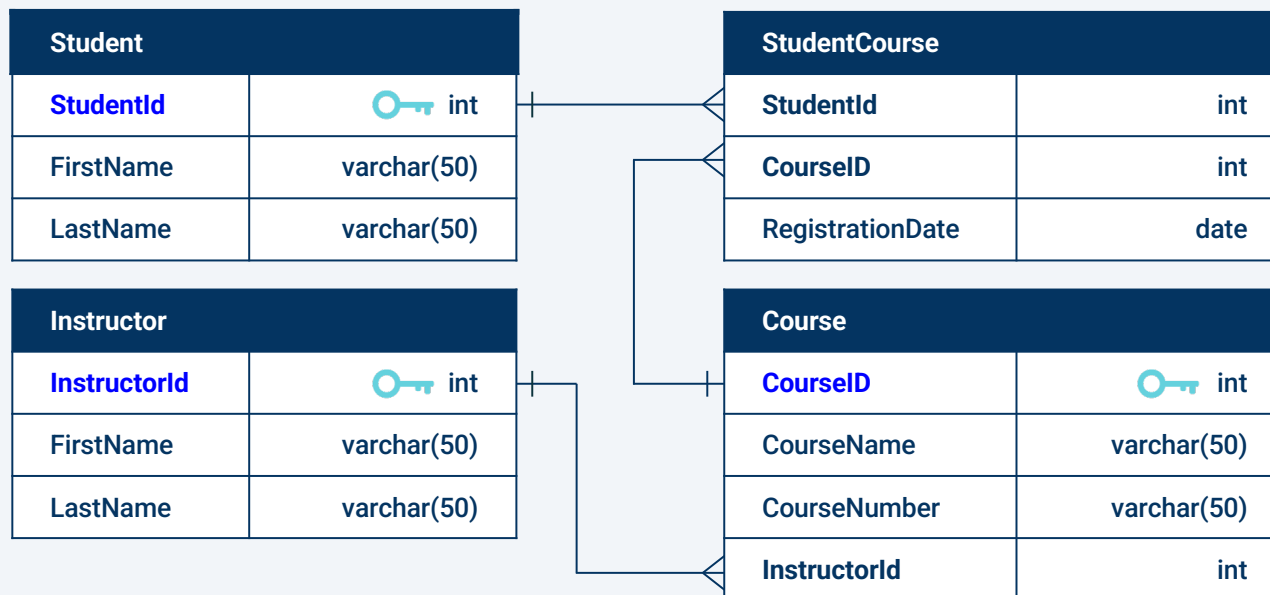
Attribute



# Entity Relationship Diagram

A typical ERD design.

One-to-one =  Many-to-one =  Many-to-many = 



# Entity Relationship Diagram

---

## Three Types of ERDs or Data Models

01

Conceptual Model Design

02

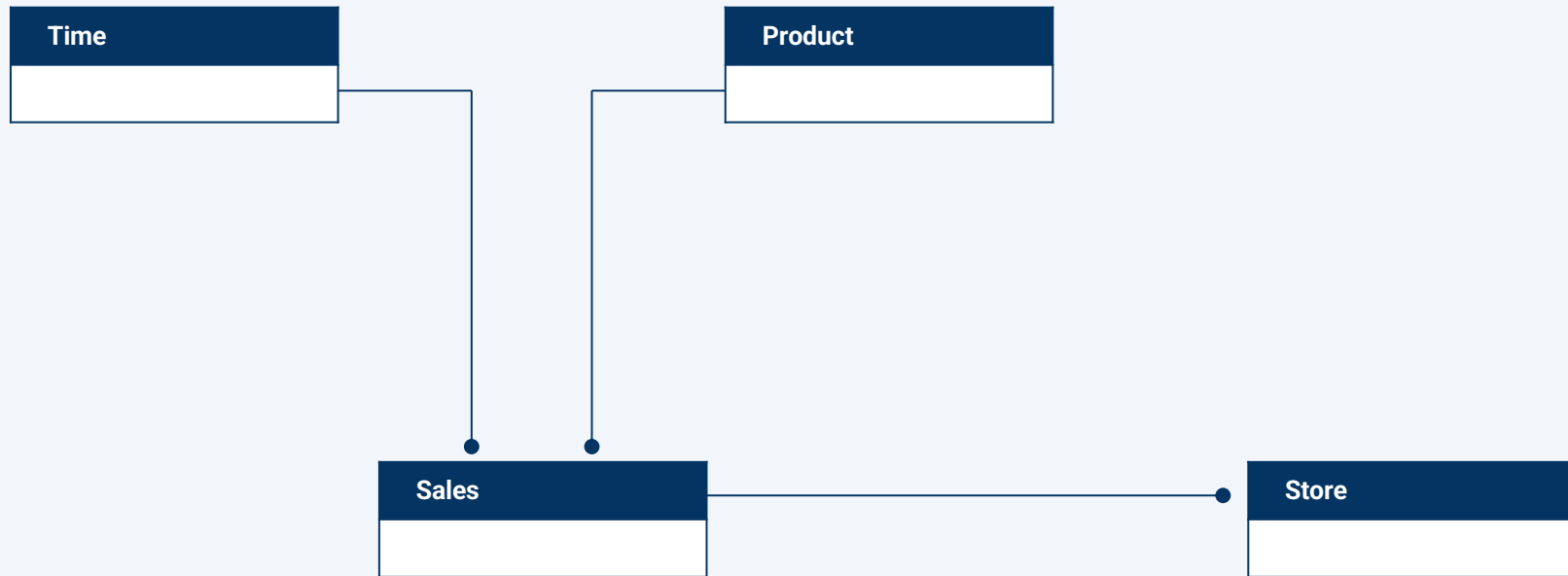
Logical Model Design

03

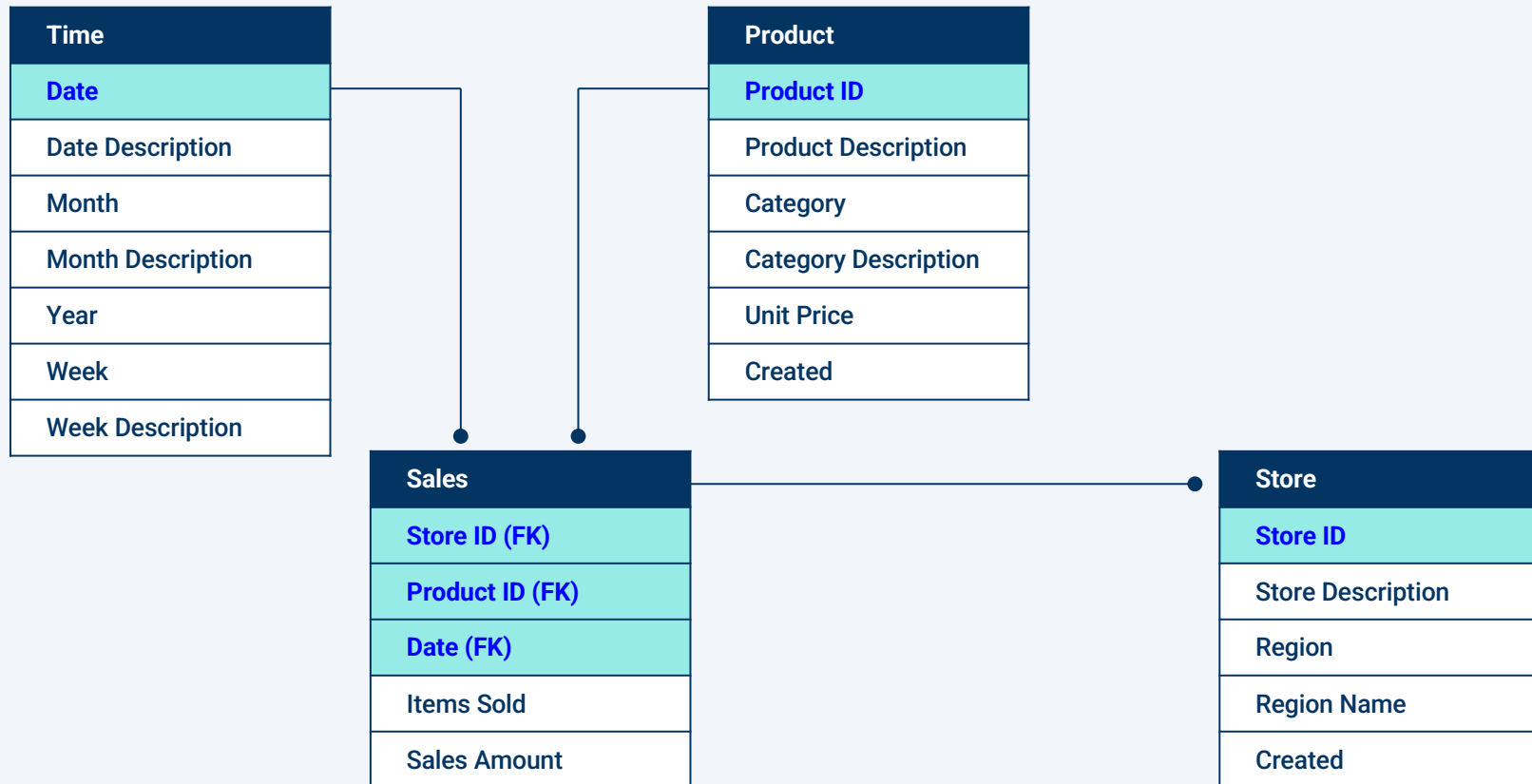
Physical Model Design

# Conceptual Model Design

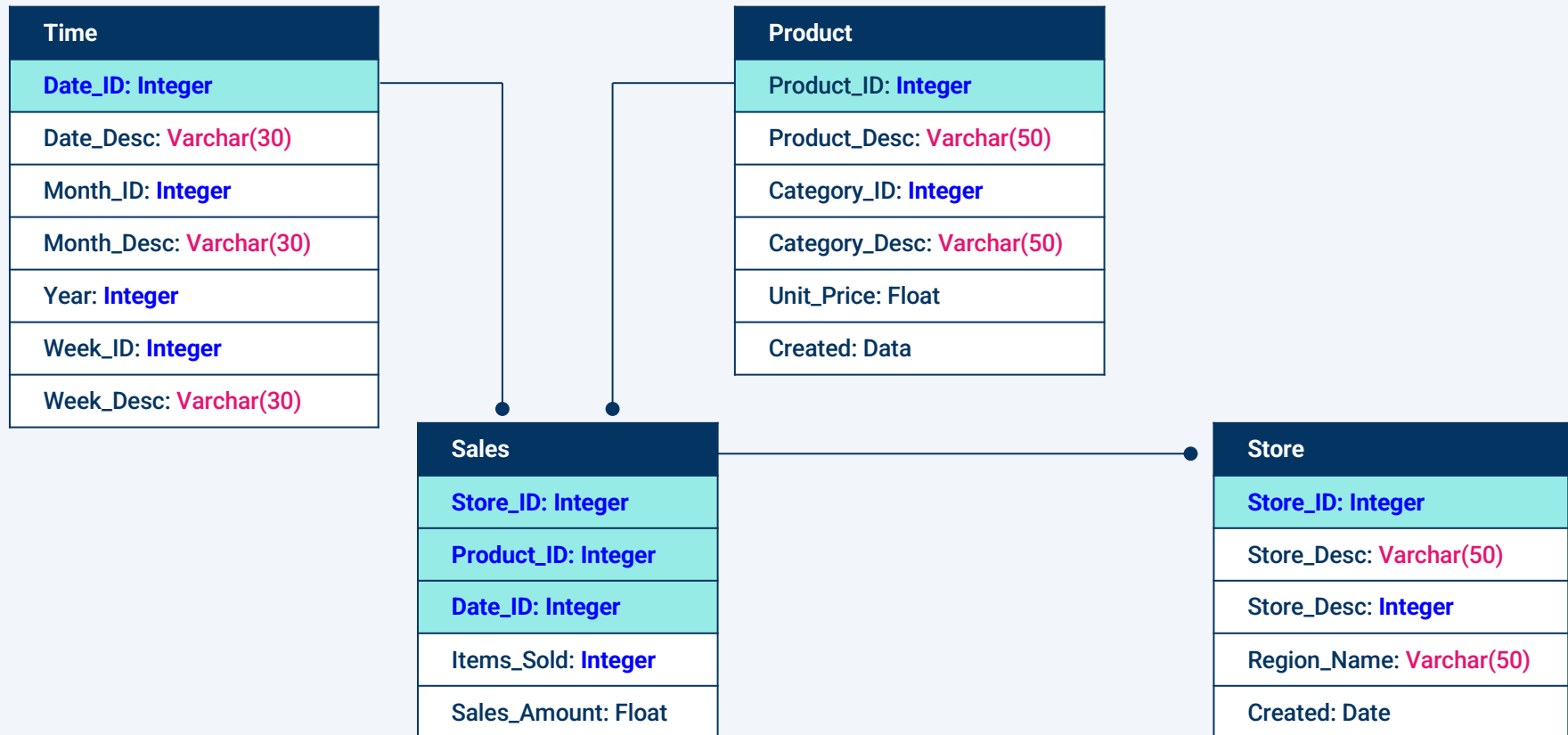
---



# Logical Model Design



# Physical Model Design



# Entity Relationship Diagram

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

