



DESIGNING DATABASE SCHEMAS

PRESENTATION





01

WHAT IS DATABASE SCHEMAS?

A database schema is the **blueprint** or **framework** that defines how data is organized, stored, and managed in a database. It outlines the structure of the database, including tables, fields, data types, and relationships between tables. Think of it as the architectural plan for a building—it determines how everything fits together and functions.

Key Elements of Schemas

1. **Table:** A collection of related data organized into rows (records) and columns (fields).
2. **Field:** A single piece of data within a table.
3. **Data Type:** The format of the data stored in a field.





WHY IS A WELL-STRUCTURED SCHEMA IMPORTANT?

employees

id	name	dept_id	job_level_id
54378	Darius	1	3
94722	Raven	2	3
45783	Eduardo	2	1

job_levels

id	name	min_salary
1	Executive	100000
2	Manager	70000
3	Contributor	35000

departments

id	dept_name	dept_head
1	Design	Jack
2	Content	Eduardo
3	Engineering	Vanessa

1. **Organization:** A schema organizes data into logical structures, making it easier to store, retrieve, and manage.
2. **Performance:** A well-designed schema ensures efficient data retrieval and processing.
3. **Scalability:** A good schema allows the database to grow without compromising performance.
4. **Data Integrity:** A schema enforces rules to ensure data accuracy and consistency.
5. **Relationships Between Data:** A schema defines how tables are related to each other.



02

TYPES OF DATABASE SCHEMAS

1. Conceptual Schema:

- Overview of the system, without technical details.
- Focuses on what the system needs to do, not how it will do it.

2. Logical Schema:

- Defines the data structure and relationships.
- Focuses on how data is organized and connected.

3. Physical Schema:

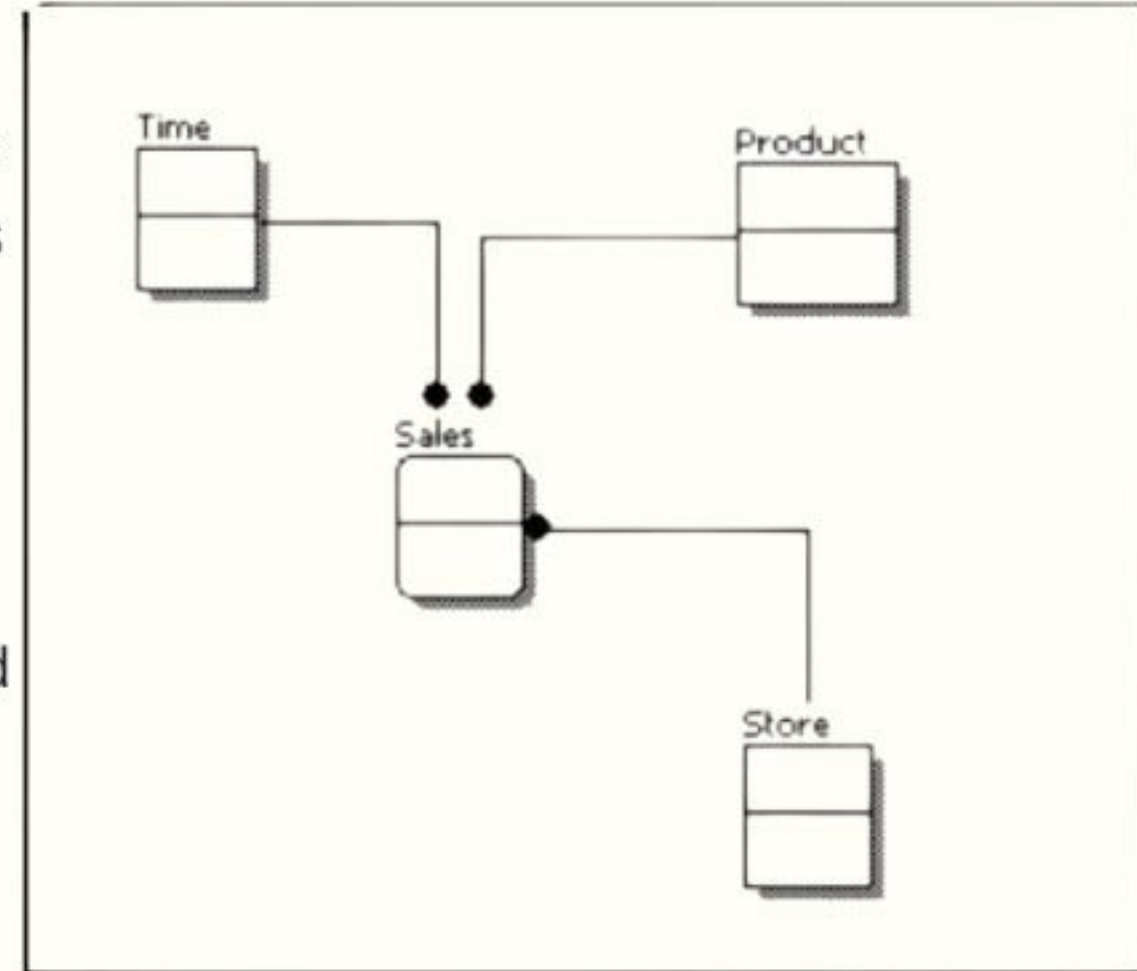
- Focuses on data storage and indexing strategies.
- Deals with the technical implementation of the database.



CONCEPTUAL SCHEMA

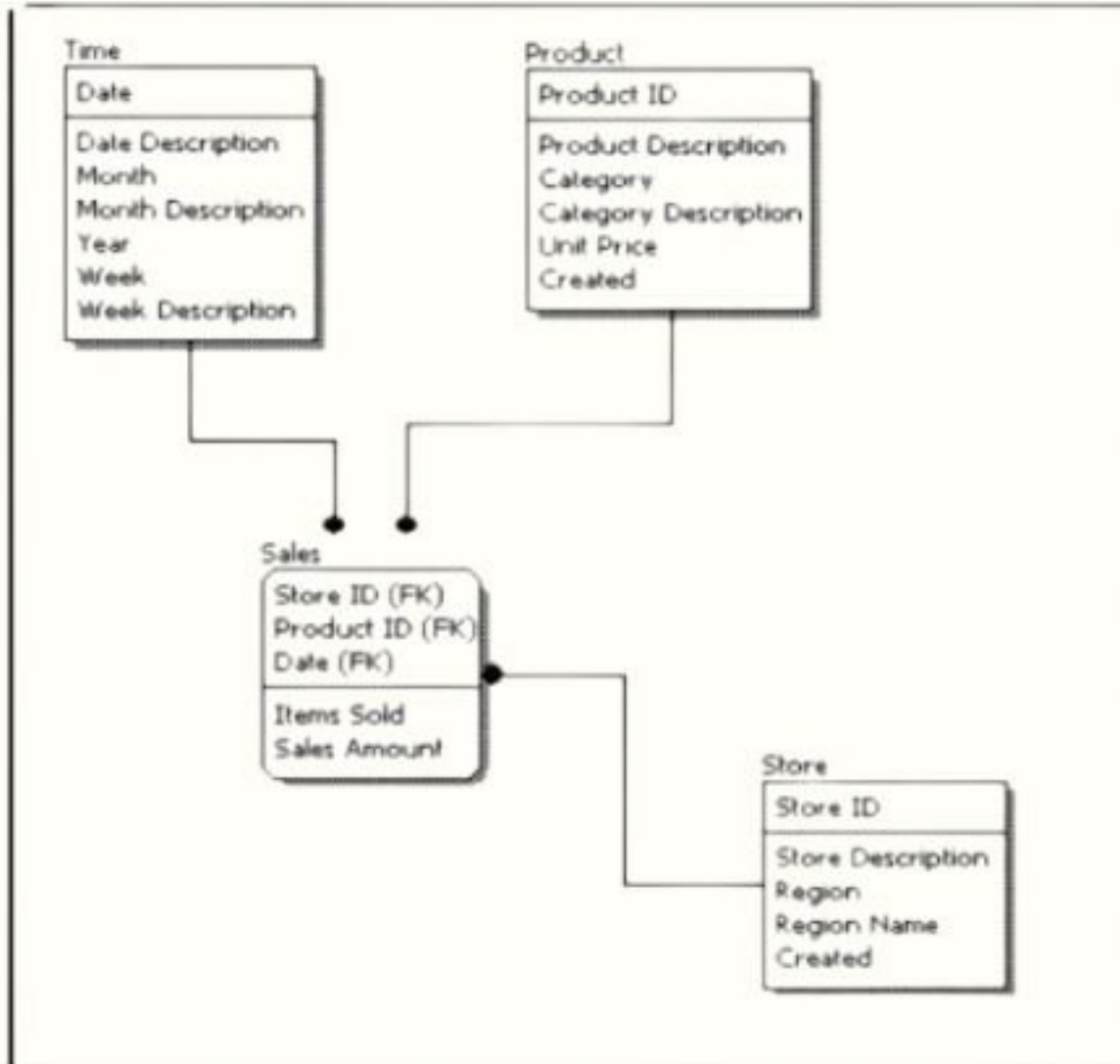
The **conceptual schema** is the big-picture view of the database. It focuses on **what the system needs to do**, not how it will do it. It's like the outline of a story—it describes the main ideas without getting into the details.

This level helps stakeholders (e.g., business owners, users) understand the system's purpose and scope. For example, in a school database, it might describe the need to store student information, course details, and grades, without specifying how the data will be stored or organized.





LOGICAL SCHEMA



The **logical schema** defines the **data structure and relationships**. It focuses on how data is organized and connected, like the detailed plan for a story that describes the characters, plot, and how everything fits together.

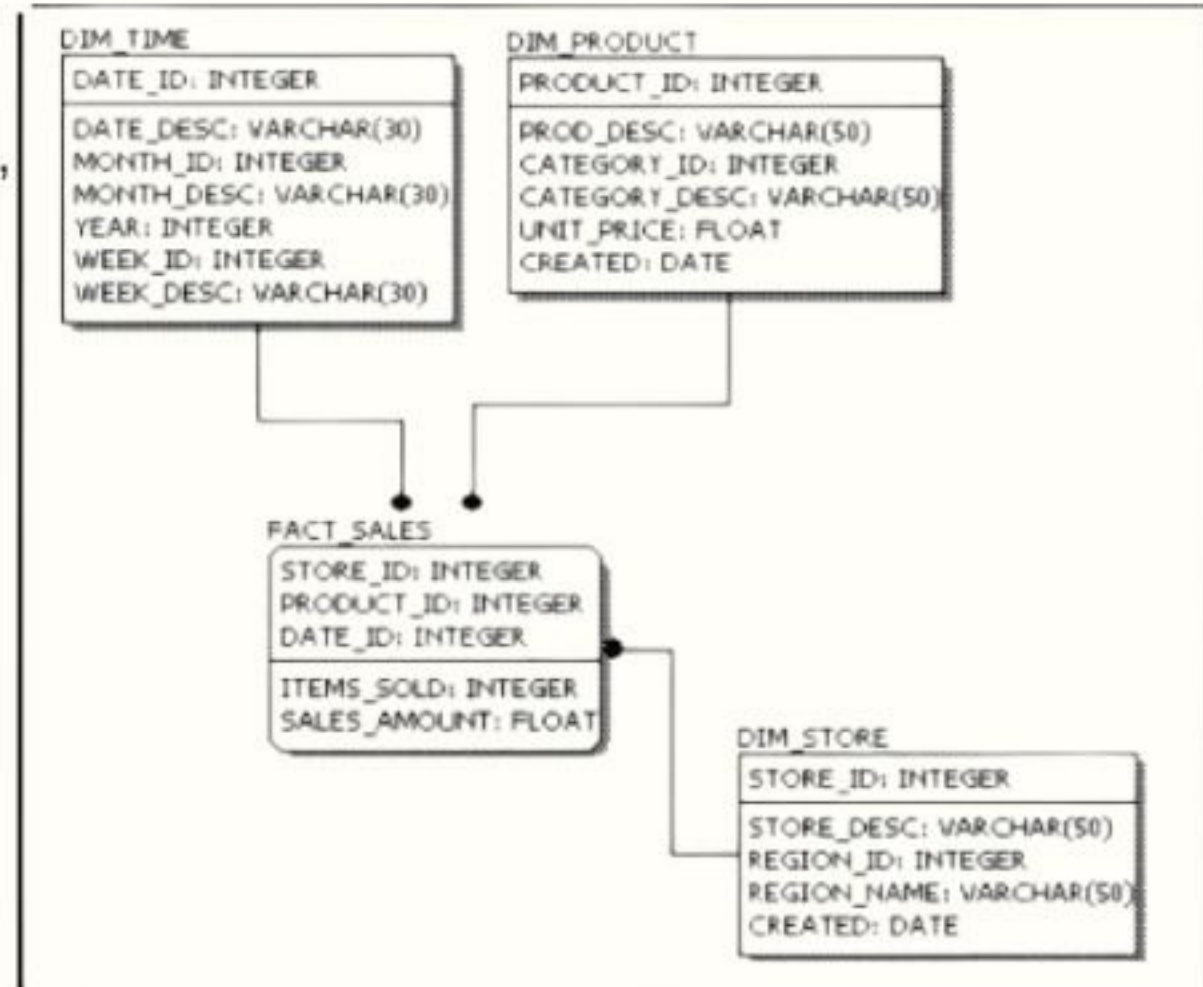
This level provides a clear structure for the database, making it easier to design and implement. For example, in a school database, it might define tables like Students, Courses, and Grades, and specify how they are related (e.g., a student can enroll in multiple courses).



PHYSICAL SCHEMA

The **physical schema** focuses on the **technical implementation** of the database, including data storage, indexing, and performance optimization. It's like the construction details for a story—it describes how the story will be printed, formatted, and distributed.

This level ensures the database is efficient, scalable, and performs well under real-world conditions. For example, in a school database, it might specify that student data is stored in a table with an index on the StudentID field to speed up searches.





03

NORMALIZATION TYPES

FIRST NORMAL FORM (1NF)

BOYCE-CODD NORMAL FORM (BCNF)

SECOND NORMAL FORM (2NF)

FOURTH NORMAL FORM (4NF)

THIRD NORMAL FORM (3NF)

FIFTH NORMAL FORM (5NF)



FIRST NORMAL FORM (1NF)

First Normal Form (1NF) is the most basic level of normalization in a DBMS. The rules for achieving 1NF are as follows:

- Each table should have a **primary key**, which uniquely identifies each record in the table.
- Each column in the table should contain only **atomic values**, which means that a single cell should contain a **single value** and **not a list of values**.
- There should be no repeating groups of data.



03

NORMALIZATION TYPES

FIRST NORMAL FORM (1NF)

BOYCE-CODD NORMAL FORM (BCNF)

SECOND NORMAL FORM (2NF)

FOURTH NORMAL FORM (4NF)

THIRD NORMAL FORM (3NF)

FIFTH NORMAL FORM (5NF)



FIRST NORMAL FORM (1NF)

First Normal Form (1NF) is the most basic level of normalization in a DBMS. The rules for achieving 1NF are as follows:

- Each table should have a **primary key**, which uniquely identifies each record in the table.
- Each column in the table should contain only **atomic values**, which means that a single cell should contain a **single value** and **not a list of values**.
- There should be no repeating groups of data.



EXAMPLE OF TABLE NOT IN 1NF

Studio	Director	Movies
Marvel	Kevin Feige	The Avengers
		Captain America
DCEU	Zack Snyder	Batman Vs Superman
		Suicide Squad

- This table contains Attribute values which are not single. This is not in Normalised form.
- To make it into 1NF we have to decompose table into atomic elements.



TABLE IN 1NF AFTER ELIMINATING

Studio	Director	Movies
Marvel	Kevin Feige	The Avengers
Marvel	Kevin Feige	Captain America
DCEU	Zack Snyder	Batman Vs Superman
DCEU	Zack Snyder	Suicide Squad

Now it is in 1NF



SECOND NORMAL FORM (2NF)

Second Normal Form (2NF) builds upon the **rules of First Normal Form (1NF)** by addressing the issue of **partial dependencies**. In 2NF, a table must not have any partial dependencies. A partial dependency exists when a non-primary key column depends on only part of a composite primary key.



EXAMPLE OF TABLE NOT IN 2NF

- Here the Primary key is (studio, movie) and city depends only on the studio and not on the whole key.
- So, this is not in 2NF form.

<u>Studio</u>	<u>Movie</u>	Budget	city
Marvel	Avengers	100	New York
Marvel	Captain America	120	New York
DCEU	Batman Vs Superman	150	Gotham
DCEU	Suicide Squad	75	Gotham



SOLUTION OF 2NF:

- Old Scheme > {Studio, Movie, Budget, City}
- New Scheme > {Movie, Studio, Budget}
- New Scheme → {Studio, City}

<u>Movie</u>	<u>Studio</u>	Budget
The Avengers	Marvel	100
Captain America	Marvel	120
Batman Vs Superman	DCEU	150
Suicide Squad	DCEU	75

<u>Studio</u>	City
Marvel	New York
DCEU	Gotham



THIRD NORMAL FORM (3NF)

Third Normal Form (3NF) builds upon the **rules of Second Normal Form (2NF)** by addressing the issue of **transitive dependencies**. In 3NF, a table must not have any transitive dependencies. A **transitive dependency** exists when a **non-primary key column depends on another non-primary key column**, rather than on the primary key.

To achieve 3NF, a table must already be in 2NF and all non-primary key columns must be directly dependent on the primary key.



EXAMPLE OF TABLE NOT IN 3NF

<u>Studio</u>	StudioTemp	City
Marvel	96	New York
DCEU	99	Gotham
Fox	96	New York
Paramount	95	Hollywood

Here Studio is the primary key and both studio temp and city depends entirely on the Studio.

1. Primary Key $\rightarrow \{\text{Studio}\}$
2. $\{\text{Studio}\} \rightarrow \{\text{StudioCity}\}$
3. $\{\text{StudioCity}\} \rightarrow \rightarrow \{\text{CityTemp}\}$
4. $\{\text{Studio}\} \rightarrow \{\text{CityTemp}\}$
5. City Temp transitively depends on Studio hence violates 3NF

It is called **transitive dependency**.



SOLUTION OF 3NF:

- Old Scheme - {Studio, StudioCity, CityTemp}
- New Scheme → {Studio, StudioCity}
- New Scheme → {StudioCity, CityTemp}

<u>Studio</u>	Studio City
Marvel	New York
DCEU	Gotham
FOx	New York
Paramount	Hollywood

Studio City	CityTemp
New York	96
Gotham	95
Hollywood	99



BOYCE-CODD NORMAL FORM (BCNF)

Boyce-Codd Normal Form (BCNF) is a higher level of normalization than Third Normal Form (3NF). It addresses the issue of **non-trivial functional dependencies**. A functional dependency is said to be non-trivial if it doesn't hold true for the case where the left-hand side is a subset of the primary key.

A table is in BCNF if and only if for every non-trivial functional dependency $X \rightarrow Y$, X is a superkey. A superkey is a set of columns that can uniquely identify a row in a table.



EXAMPLE OF BCNF

- Scheme > {MovieTitle, MovieID, PersonName, Role, Payment}
- Key1 > {MovieTitle, PersonName}
- Key2 > {MovieID, PersonName}

MovieTitle	MovieID	PersonName	Role	Payment
The Avengers	M101	Robert Downet Jr.	Tony Stark	200m
The Avengers	M101	Chris Evans	Chris Rogers	120m
Batman Vs Superman	D101	Ben Afflek	Bruce Wayne	180m
Batman Vs Superman	D101	Henry Cavill	Clarke Cent	125m
A walk to remember	P101	Mandy Moore	Jamie Sullivan	50m

Dependency between MovieID & MovieTitle Violates BCNF



SOLUTION OF BCNF:

- Place the two candidate primary keys in separate entities
- Place each of the remaining data items in one of the resulting entities according to its dependency on the primary key.
- New Scheme > {MovieID, PersonName, Role, Payment}
- New Scheme > {MovieID, MovieTitle}

<u>MovieID</u>	<u>PersonName</u>	Role	<u>Payment</u>
M101	Robert Downey Jr.	Tony Stark	200m
M101	Chris Evans	Chris Rogers	125m
D101	Ben Afflek	Bruce Wayne	175m
D101	Henry Cavill	Clarke Cent	120m
P101	Mandy Moore	Jamie Sullivan	50m

<u>MovieID</u>	<u>MovieTitle</u>
M101	The Avengers
D101	Batman VS Superman
P101	A walk to remember



FOURTH NORMAL FORM (4NF)

Fourth Normal Form (4NF) is a higher level of normalization than Boyce-Codd Normal Form (BCNF). It addresses the **issue of multi-valued dependencies**. A multi-valued dependency is a type of dependency where a non-primary key column is functionally dependent on two or more independent non-primary key columns.

A table is in 4NF if and only if it does not contain any multi-valued dependencies.



EXAMPLE OF 4NF

- Scheme \rightarrow {MovieName, ScreeningCity, Genre}

Movie	ScreeningCity	Genre
The Avengers	Los Angeles	Sci-Fi
The Avengers	New York	Sci-Fi
Batman vs Superman	Santa Cruz	Drama
Batman vs Superman	Durham	Drama
A Walk to remember	New York	Romance

- Many Movies can have the same Genre and Many Cities can have the same movie.
- So this table violates 4NF.



SOLUTION OF 4NF:

- Move the two multi-valued relations to separate tables
- Identify a primary key for each of the new entity.
- **New Scheme** → {MovieName, ScreeningCity}
- **New Scheme** > {MovieName, Genre}

MovieName	ScreeningCity
Batman vs Superman	Santa Cruz
The Avengers	Los Angeles
A Walk to remember	New york
Batman vs Superman	Durham
The Avengers	New york

MovieName	Genre
Batman vs Superman	Drama
The Avengers	Sci-Fi
A Walk to remember	Romance

We split the table into two tables with one multivalued value in each.



FIFTH NORMAL FORM (5NF)

Fifth Normal Form (5NF), also known as **Projection-Join Normal Form (PJ/NF)** is a higher level of normalization than Fourth Normal Form (4NF). It addresses the issue of **join dependency**. A join dependency is a type of dependency where a table can be split into two or more tables, each of which contains a subset of the original table's columns, and can be recombined by joining the tables on their common columns.

A table is in 5NF if and only if it does not contain any join dependencies.



EXAMPLE OF 5NF

<u>Theatre</u>	<u>Company</u>	<u>Movie</u>
T ₁	Paramount	A walk to remember
T ₂	Marvel	The Avengers
T ₂	Marvel	Age of Ultron
T ₂	Marvel	Dr. Strange
T ₃	DCEU	Batman Vs Superman
T ₄	Sony	Spiderman Homecoming

- Here Product is related to each company and MVD: **Theatre** >> Company, **Movie**



SOLUTION OF BCNF:

<u>Theatre</u>	<u>Company</u>
T ₁	Paramount
T ₂	Marvel
T ₃	DCEU
T ₄	Sony

<u>Theatre</u>	<u>Movie</u>
T ₂	The Avengers
T ₁	A walk to remember
T ₂	Age of Ultron
T ₂	Dr. Strange
T ₃	Batman vs Superman
T ₄	Spiderman Homecoming



04

DBMS KEYS

- **Keys** are like **unique identifiers** for records in a database. They help us find, organize, and relate data efficiently.
- Think of keys as the "address" of a record—they ensure that each piece of data can be uniquely identified and accessed.

Types of Keys:

1. **Primary Key:** The main unique identifier for a table. No two records can have the same primary key.
2. **Foreign Key:** A key that links two tables together. It refers to the primary key of another table.
3. **Candidate Key:** Any column or set of columns that could be used as a primary key (it's unique and not null)
4. **Alternate Key:** A candidate key that wasn't chosen as the primary key.
5. **Composite Key:** A primary key made up of two or more columns.
6. **Surrogate Key:** A system-generated key (usually a number) that has no real-world meaning but is used to uniquely identify records.




04

RELATIONSHIPS IN DBMS

Relationships in a database are the **connections between tables** that **link** and **organize data** in meaningful ways, enabling us to model real-world scenarios like students to courses or employees to departments. Without relationships, databases would just be isolated tables with no way to connect related information.

Types of Relationships in Database:

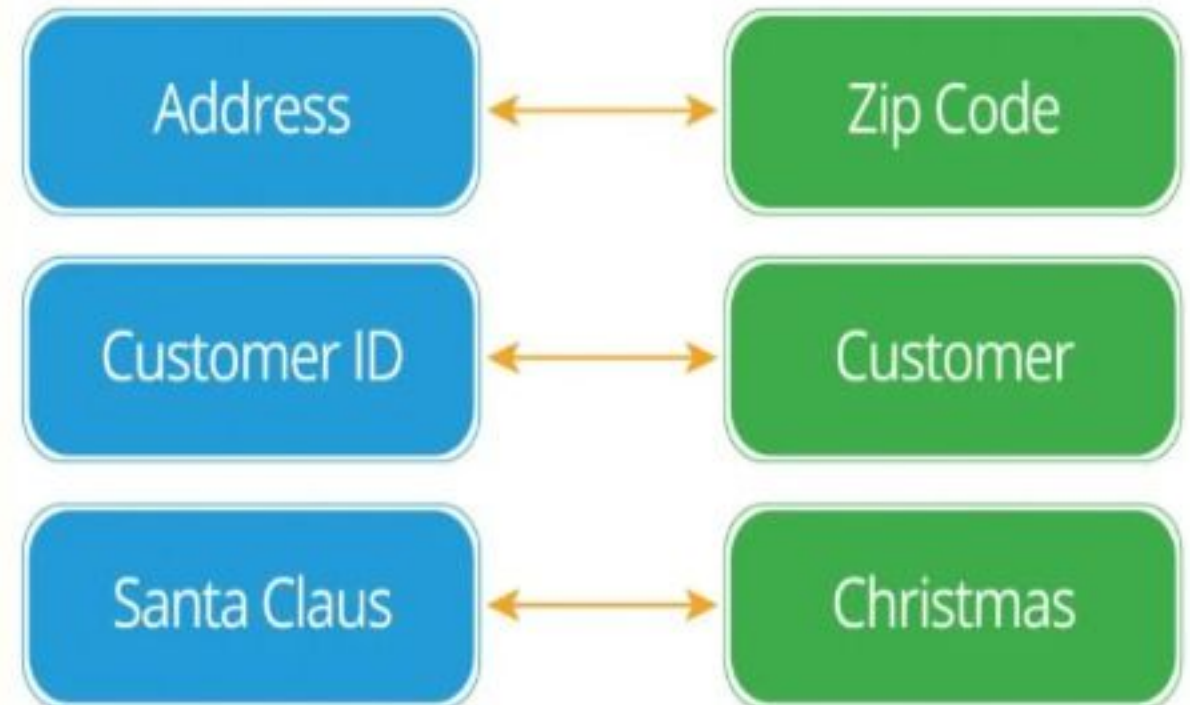
1. One-to-One (1:1)
 2. One-to-Many (1:N)
 3. Many-to-Many (N:M)
- 



ONE-TO-ONE (1:1)

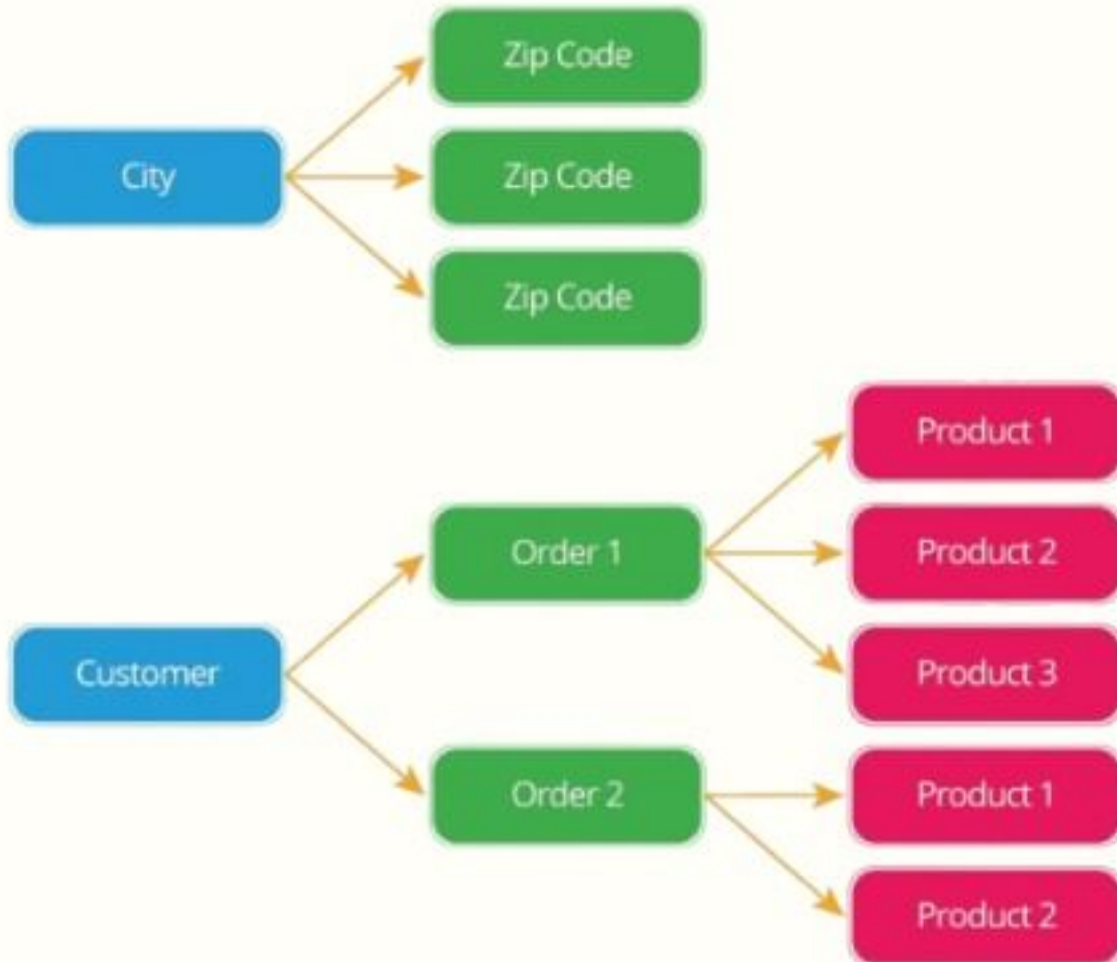
This is the least common type of relationship, but it's the easiest to visualize. In this relationship, there is one and only one record on each side of the relationship. Each row in a table is connected to a single row in another table.

A **one-to-one relationship** is always **one-to-one**, no matter which table you start with.





ONE-TO-MANY (1:N)



In this relationship, there is **one record on one side** of the relationship, and **zero, one, or many** on the other.

This is the most common relationship type. From the linked table, the one-to-many relationship becomes a many-to-one relationship. For example, a biological mom can have many children, but each child can only have one biological mom.



MANY-TO-MANY (N:M)

This is the most flexible relationship type. There is zero, one, or many records on one side of the relationship, and zero, one, or many on the other.

