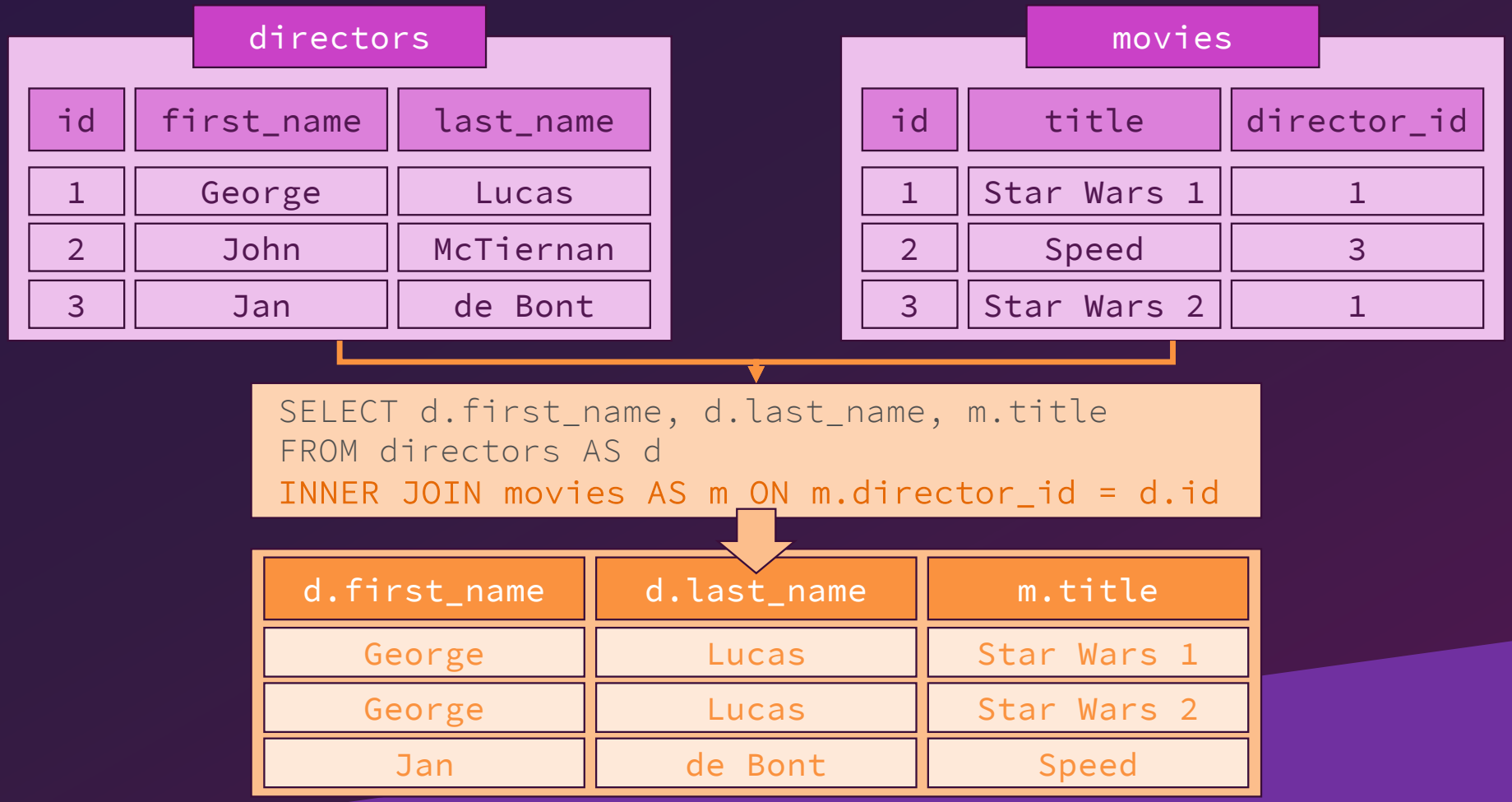
A First Look At Foreign Keys & Related Data



A First Look At Foreign Keys & Related Data

directors

id	first_name	last_name
1	George	Lucas
2	John	McTiernan
3	Jan	de Bont

movies

id	title	director_id
1	Star Wars 1	1
2	Speed	3
3	Star Wars 2	1

Primary Key

Foreign Key

Data is "connected" across tables via keys

The primary key of table A is used as a foreign key in table B

Every table has at **most one primary key** but every table **may use multiple foreign keys**

ACADEMIND

Why are we splitting data?

ACADEMIND

Data should be normalized

ACADEMIND

But how should you split data?

Understanding Data Normalization

A concept that reduces data redundancy and increases data maintainability

Goal: Split compound and grouped data into multiple, standalone values

Understanding Data Normalization

A concept that reduces data redundancy and increases data maintainability

Goal: Split compound and grouped data into multiple, standalone values

Full Name
Schwarzmüller, Maximilian

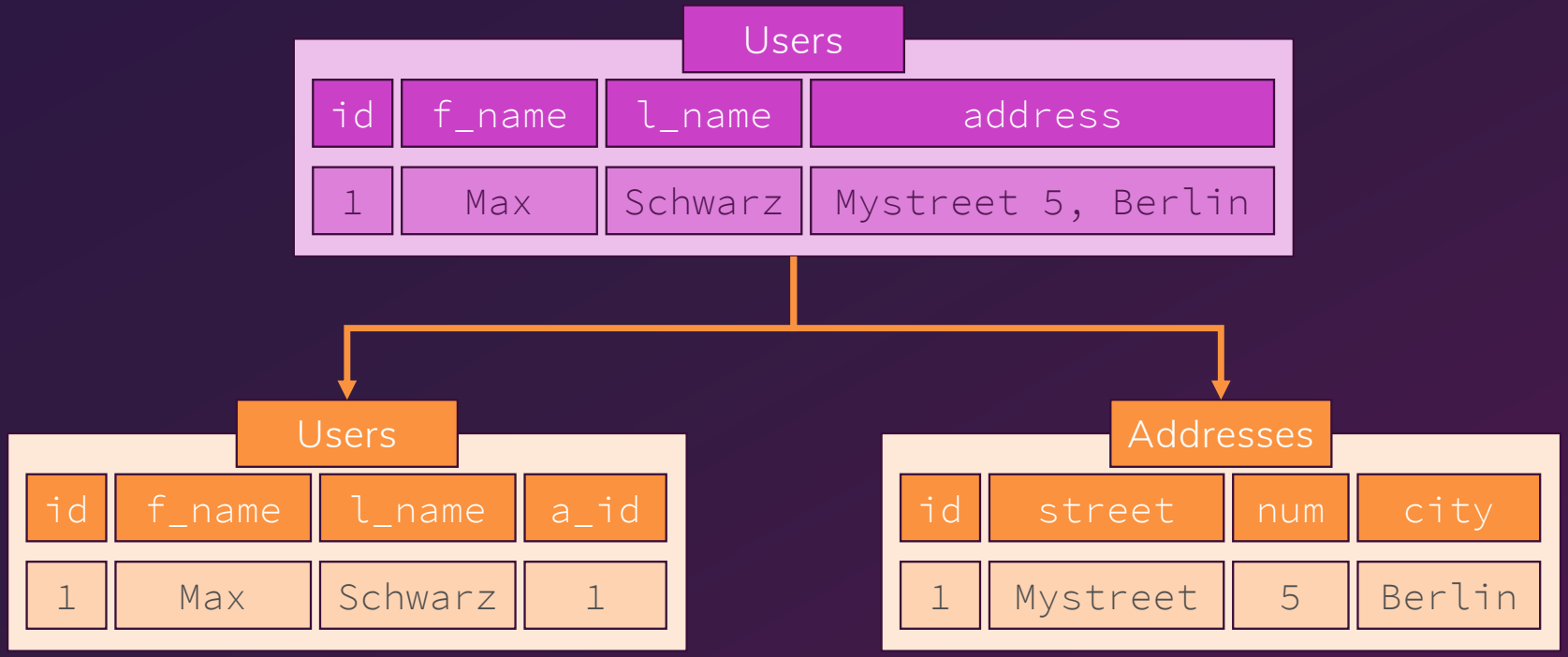
Understanding Data Normalization

A concept that reduces data redundancy and increases data maintainability

Goal: Split compound and grouped data into multiple, standalone values

Full Name
Schwarzmüller Maximilian

Data Normalization: Splitting Data Entities Across Tables



Normalization Helps Us Avoid Data Redundancy

id	first_name	last_name	address_street	address_num	address_city
1	Max	Schwarz	Teststreet	5	Munich
2	Manuel	Lorenz	Mystreet	12	Berlin
3	Julie	Barnes	Teststreet	5	Munich

Normalization Helps Us Avoid Data Redundancy

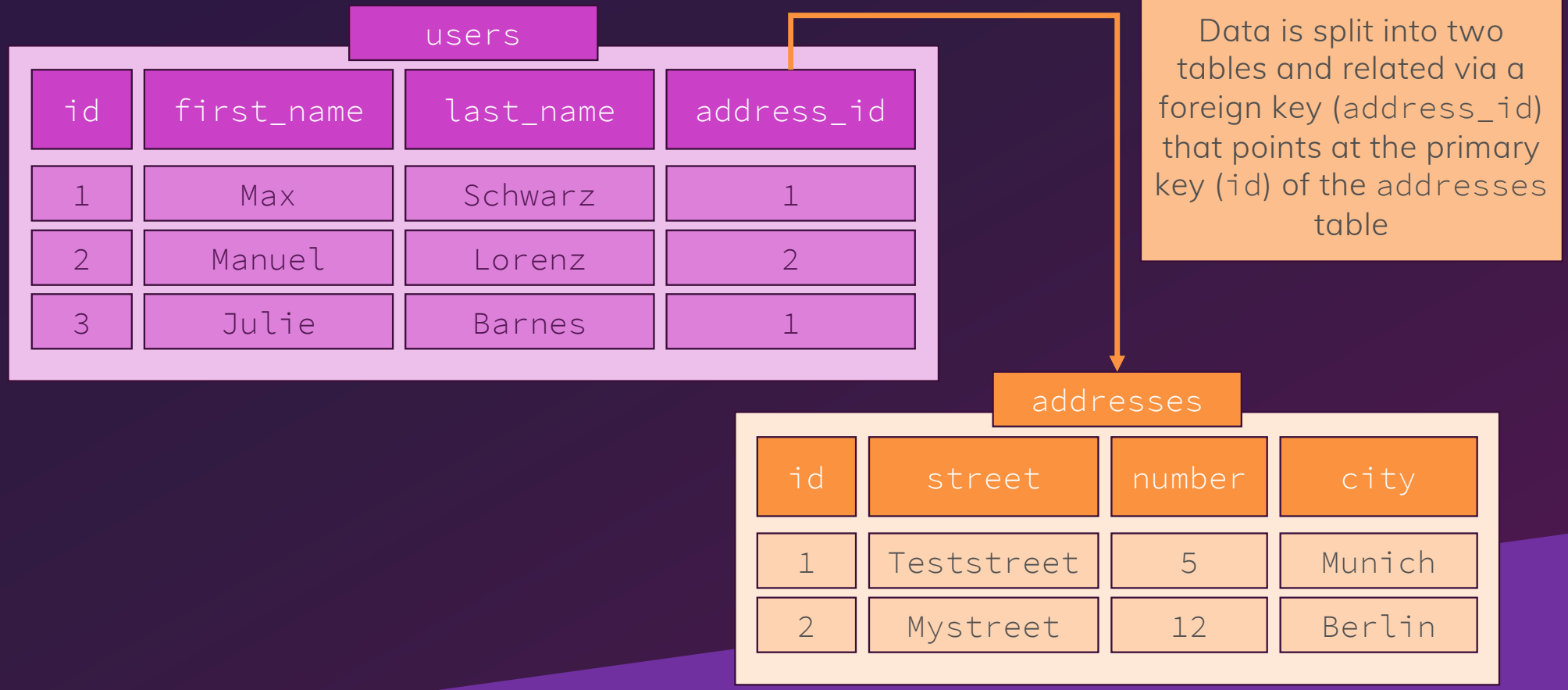
Coming up with related column names like address_xyz is another indicator of suboptimal normalization

id	first_name	last_name	address_street	address_num	address_city
1	Max	Schwarz	Teststreet	5	Munich
2	Manuel	Lorenz	Mystreet	12	Berlin
3	Julie	Barnes	Teststreet	5	Munich

Multiple rows with duplicate data (in a single table) should be avoided!

Having duplicate data in some columns in multiple rows is a clear indicator of suboptimal data normalization

Normalization Helps Us Avoid Data Redundancy



Why Normalization?

Goal: Avoid CUD *(create, update, delete)* anomalies

Creation Anomaly

Inserting incomplete data

id	course	teacher
1	SQL 1	Max
2	Web Dev	Manuel

Update Anomaly

Many rows must be updated when a single entity changes

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Deletion Anomaly

Deleting too much data

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Why Normalization?

Goal: Avoid CUD *(create, update, delete)* anomalies

Creation Anomaly

Inserting incomplete data

id	course	teacher
1	SQL 1	Max
2	Web Dev	Manuel
3		Julie

Incomplete data might be inserted

Update Anomaly

Many rows must be updated when a single entity changes

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Deletion Anomaly

Deleting too much data

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Why Normalization?

Goal: Avoid CUD *(create, update, delete)* anomalies

Creation Anomaly

Inserting incomplete data

id	course	teacher
1	SQL 1	Max
2	Web Dev	Manuel
3	MongoDB	

Incomplete data might be inserted

Update Anomaly

Many rows must be updated when a single entity changes

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

If “Max” is changed to “Maximilian”, multiple rows must be adjusted

Deletion Anomaly

Deleting too much data

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Why Normalization?

Goal: Avoid CUD *(create, update, delete)* anomalies

Creation Anomaly

Inserting incomplete data

id	course	teacher
1	SQL 1	Max
2	Web Dev	Manuel
3	MongoDB	

Incomplete data might be inserted

Update Anomaly

Many rows must be updated when a single entity changes

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

If “Max” is changed to “Maximilian”, multiple rows must be adjusted

Deletion Anomaly

Deleting too much data

id	student	course
1	Max	SQL
2	Manuel	SQL
3	Max	MongoDB

Multiple rows are deleted, data might be lost

ACADEMIND

Avoid unnecessary work & possible errors

ACADEMIND

There are six forms of normalization

And an easy-to-follow rule for the rest of us

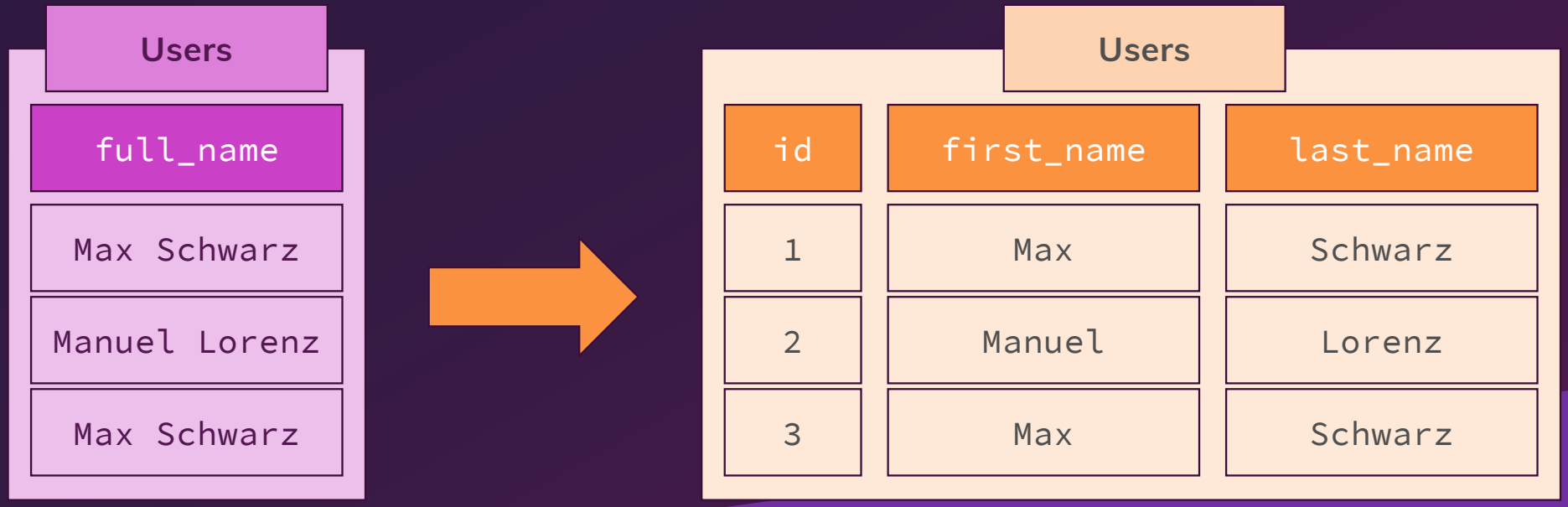
ACADEMIND

Data Normalization – The Different Forms

1NF *(First Normal Form)*	Each table cell (column + row) should contain a single value Each row (record) should be unique
2NF *(Second Normal Form)*	There are no duplicate row values because of multi-column keys (composite keys)
3NF *(Third Normal Form)*	All column values in a row are dependent on only the primary key
BCNF ***(3.5NF)*** *(Boyce-Codd Normal Form)*	There must be no conflicting unique identification criteria (i.e. column value combinations) ➔ Avoid multiple entities in one table
4NF *(Fourth Normal Form)*	All combinations of all (non-key) cell values should be possible
5NF *(Fifth Normal Form)*	There are no clashing column values because of implicit column dependencies

Normalization: First Normal Form (1NF)

Store single values & assign primary keys



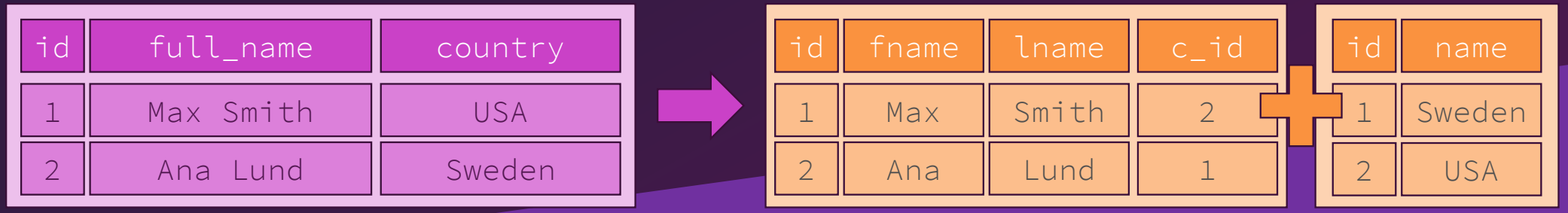
Normalization For Humans

Simple Rule

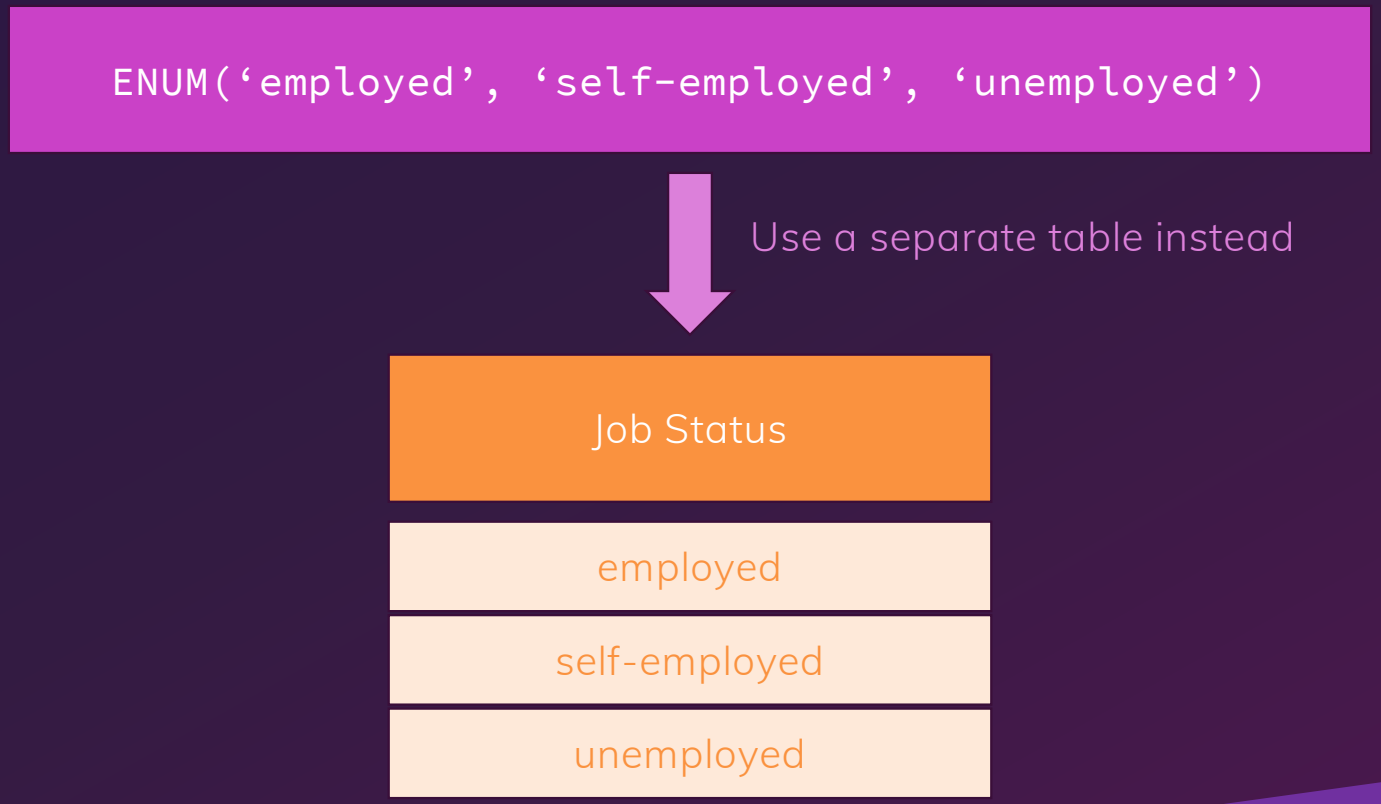
Avoid mixing data entities in the same table, avoid multiple values in a single table cell but also try to avoid splitting basic data across dozens of tables!

One Table = One Data Entity
One Cell = One Value

And there might be more data entities than it might first look like!



Normalization: Consider Avoiding Enums

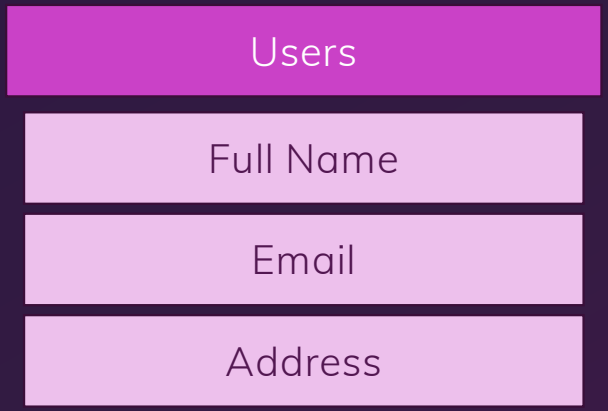


Normalization: Consider Standardizing Values

Some (but not all) values should probably be standardized
(organized as a separate data entity)

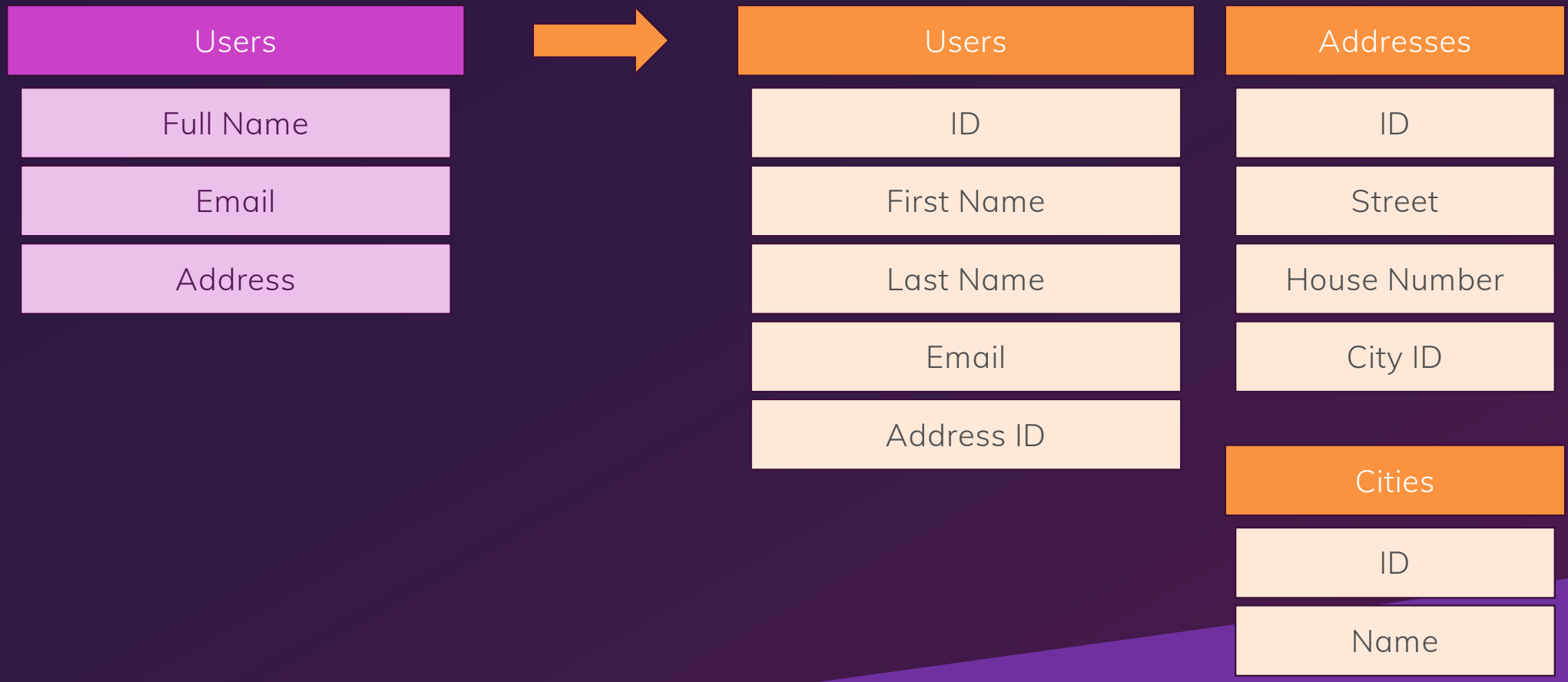
ACADEMIND

Example Time!



ACADEMIND

Example Time!



INNER JOIN

addresses

id	street	house_number
1	Teststreet	10A
2	Some Street	5
3	My Street	18

users

id	first_name	…	address_id
1	Max	…	1
2	Manuel	…	3

```
SELECT u.first_name, a.street, a.house_number
FROM users AS u
INNER JOIN addresses AS a ON u.address_id = a.id
```

INNER JOIN

addresses

id	street	house_number
1	Teststreet	10A
2	Some Street	5
3	My Street	18

users

id	first_name	…	address_id
1	Max	…	1
2	Manuel	…	3

```
SELECT u.first_name, a.street, a.house_number
FROM users AS u
INNER JOIN addresses AS a ON u.address_id = a.id
```

LEFT JOIN

addresses

id	street	house_number
1	Teststreet	10A
2	Some Street	5
3	My Street	18

users

id	first_name	…	address_id
1	Max	…	1
2	Manuel	…	3

```
SELECT u.first_name, a.street, a.house_number
FROM addresses AS a
LEFT JOIN users AS u ON u.address_id = a.id
```

LEFT JOIN

addresses

id	street	house_number
1	Teststreet	10A
2	Some Street	5
3	My Street	18

users

id	first_name	…	address_id
1	Max	…	1
2	Manuel	…	3

```
SELECT u.first_name, a.street, a.house_number
FROM addresses AS a
LEFT JOIN users AS u ON u.address_id = a.id
```

LEFT JOIN

addresses

id	street	house_number
1	Teststreet	10A
2	Some Street	5
3	My Street	18

users

id	first_name	…	address_id
1	Max	…	1
2	Manuel	…	3
3	Julie	…	NULL

```
SELECT u.first_name, a.street, a.house_number
FROM addresses AS a
LEFT JOIN users AS u ON u.address_id = a.id
```

ACADEMIND

What About RIGHT JOIN?

What's Happening Behind The Scenes

cities

id	name
1	Berlin
2	New York
…	…
7	Washington

addresses

id	street	num	city_id
1	Teststreet	8A	3
2	Some Street	10	1
…	…	…	…
10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

What's Happening Behind The Scenes

cities

id	name
1	Berlin
2	New York
…	…
7	Washington

addresses

id	street	num	city_id
1	Teststreet	8A	3
2	Some Street	10	1
…	…	…	…
10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
INNER JOIN users ON …
```

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
INNER JOIN users ON …
```

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
INNER JOIN users ON …
```

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
INNER JOIN users ON …
```

Final Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…

1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

What's Happening Behind The Scenes

```
SELECT name FROM cities LEFT JOIN addresses ON …
```

Intermediate Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…
7	Washington	10	Smallstreet	11	3

Not related to any user

users

id	f_name	…	address_id
1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

```
SELECT name FROM cities LEFT JOIN addresses ON …
INNER JOIN users ON …
```

Final Result Set

1	Berlin	1	Teststreet	8A	3
2	New York	2	Some Street	10	1
…	…	…	…	…	…

1	Max	…	2
2	Manuel	…	4
…	…	…	…
7	Michael	…	8

False Friend: UNION

`UNION` is a clause that combines multiple result sets into one result set by appending rows

`JOIN` clauses merge multiple tables into one result set by appending columns

id	first_name	last_name
1	Max	Schwarz
2	Manuel	Lorenz

id	first_name	last_name
3	Julie	Barnes

False Friend: UNION

`UNION` is a clause that combines multiple result sets into one result set by appending rows

`JOIN` clauses merge multiple tables into one result set by appending columns

id	first_name	last_name
1	Max	Schwarz
2	Manuel	Lorenz
3	Julie	Barnes

```
SELECT * FROM users WHERE id < 3
UNION
SELECT * FROM users WHERE id = 3
```

False Friend: UNION

UNION is a clause that combines multiple result sets into one result set by appending rows

id	first_name	last_name
1	Max	Schwarz
2	Manuel	Lorenz
3	Julie	Barnes

```
SELECT * FROM users WHERE id < 3
UNION
SELECT * FROM users WHERE id = 3
```

JOIN clauses merge multiple tables into one result set by appending columns

id	name	address_id
1	Max	1
2	Manu	2

id	street	city
1	Teststreet	Munich

False Friend: UNION

UNION is a clause that combines multiple result sets into one result set by appending rows

id	first_name	last_name
1	Max	Schwarz
2	Manuel	Lorenz
3	Julie	Barnes

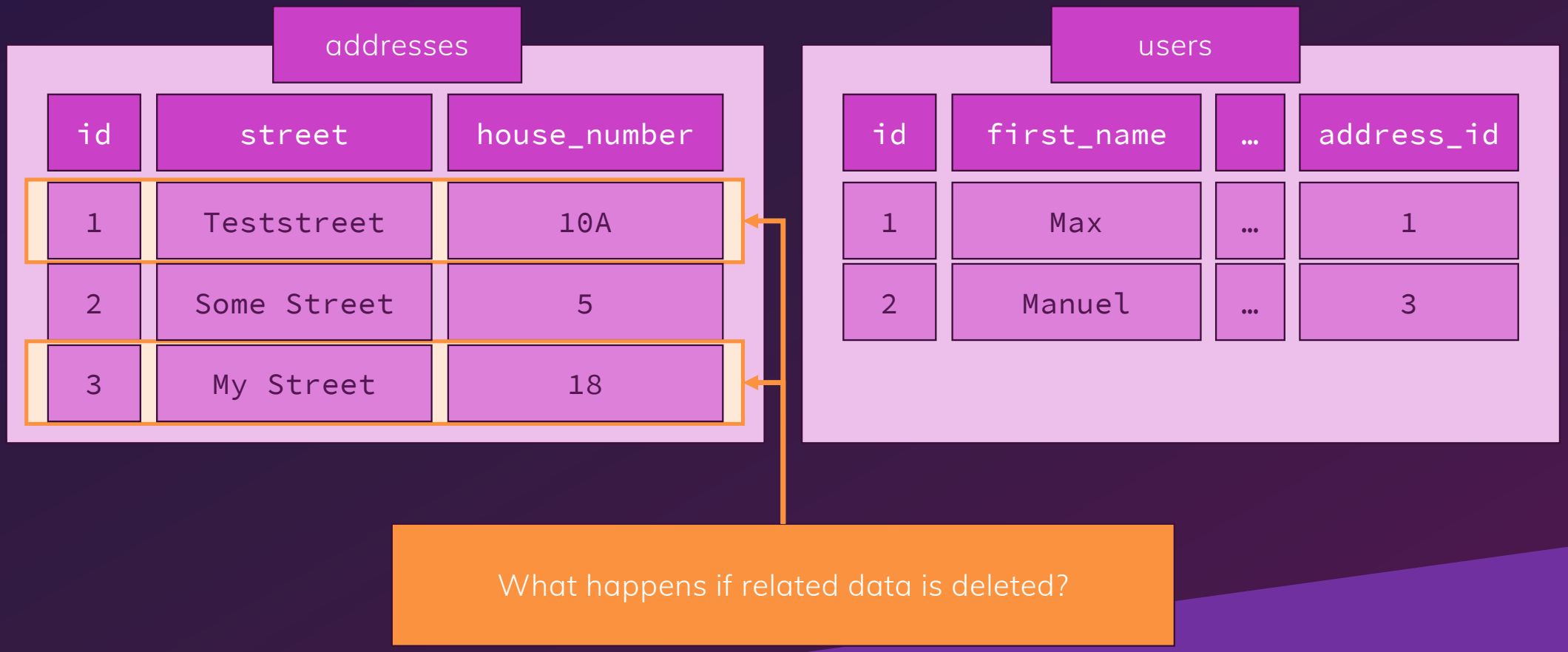
```
SELECT * FROM users WHERE id < 3
UNION
SELECT * FROM users WHERE id = 3
```

JOIN clauses merge multiple tables into one result set by appending columns

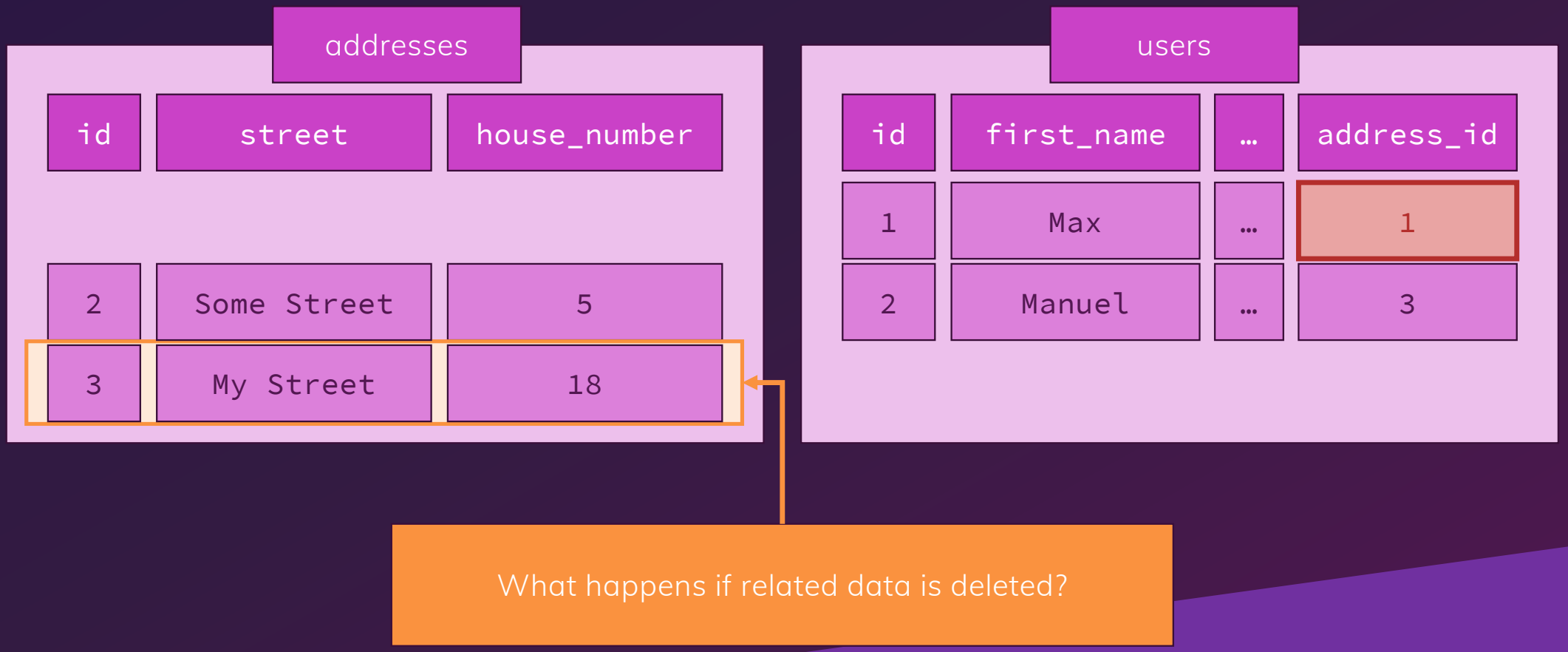
id	name	street	city
1	Max	Teststreet	Munich

```
SELECT u.id, name, street, city
FROM users AS u
INNER JOIN addresses AS a
ON a.id = u.address_id
```

Related Data & Data Integrity



Related Data & Data Integrity



Foreign Key Constraints

```
CREATE TABLE users (
  …
  address_id INT REFERENCES addresses (id) ON DELETE CASCADE
  …
);
```

REFERENCES

Defines the **related table + column**

ON DELETE

ON UPDATE

Defines the **action** that should be executed if a related row is deleted or updated

ACADEMIND

ON DELETE & ON UPDATE

RESTRICT	Prevent the intended action (e.g. deleting a related row)
NO ACTION *(default)*	Prevent the intended action (e.g. deleting a related row) *Check can be deferred, e.g. as part of a transaction*
CASCADE	Perform the same action (e.g. deleting a related row) on the row with the foreign key
SET NULL	Set the foreign key value to NULL if the related row was deleted
SET DEFAULT	Set the foreign key value to its DEFAULT value if the related row was deleted

ACADE MIND

Different Kinds Of Data Relationships

One-to-Many
(1:n)

One record in table A has one or many related records in table B

e.g. an employee belongs to one company but a company has many employees

Many-to-Many
(n:n)

One record in table A has one or many related tables in table B – and vice versa

e.g. an employee is part of multiple projects and every project has multiple employees assigned to it

One-to-One
(1:1)

One record in table A belongs to exactly one record in table B – and vice versa

e.g. an employee has exactly one intranet account and every intranet account belongs to exactly one employee

Example Time!

Some Company

Employees should be organized into teams and have one intranet account per employee. Teams sit in different company buildings, though one building can house multiple teams. In addition to teams, employees can be part of projects. Every employee should only be part of one team but may be part of multiple projects.

Example Time!

Some Company

Employees should be organized into teams and have one intranet account per employee. Teams sit in different company buildings, though one building can house multiple teams. In addition to teams, employees can be part of projects. Every employee should only be part of one team but may be part of multiple projects.

Example Time!

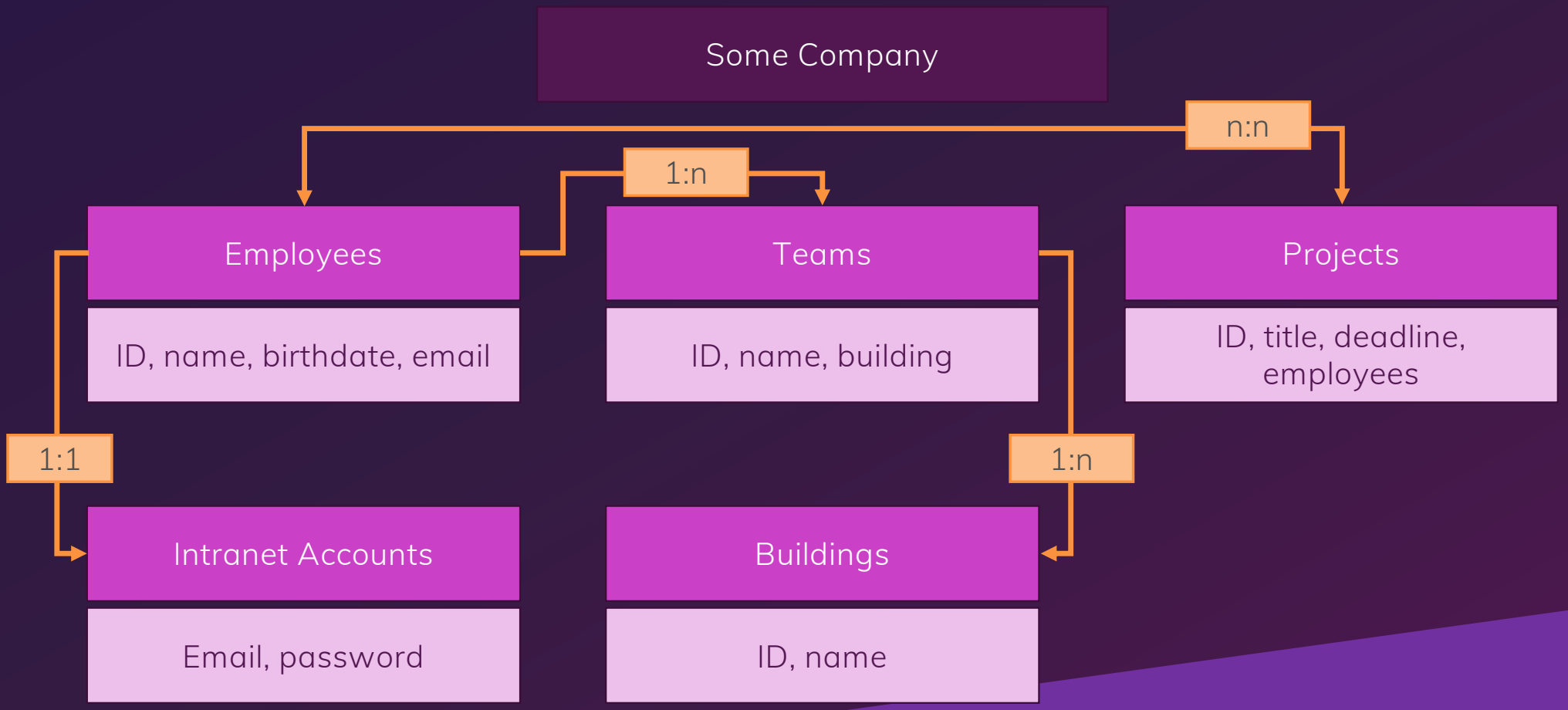
Some Company

Employees	Teams	Projects
ID, name, birthdate, email	ID, name, building	ID, title, deadline, employees

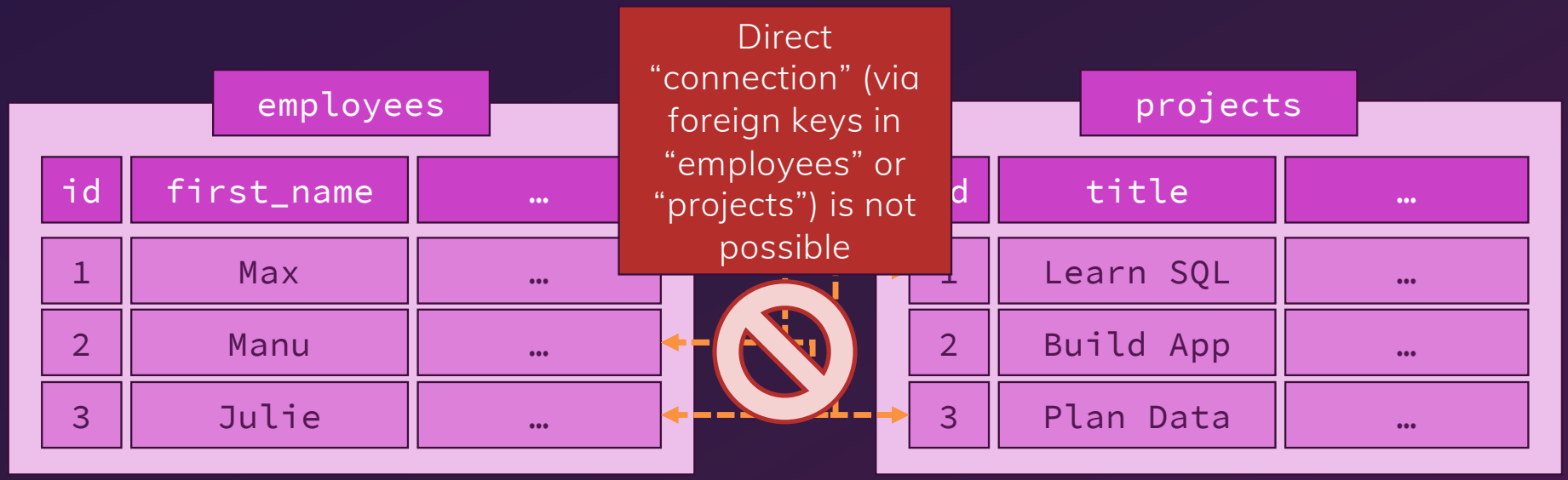
Intranet Accounts	Buildings
Email, password	ID, name

ACADEMIND

Example Time!



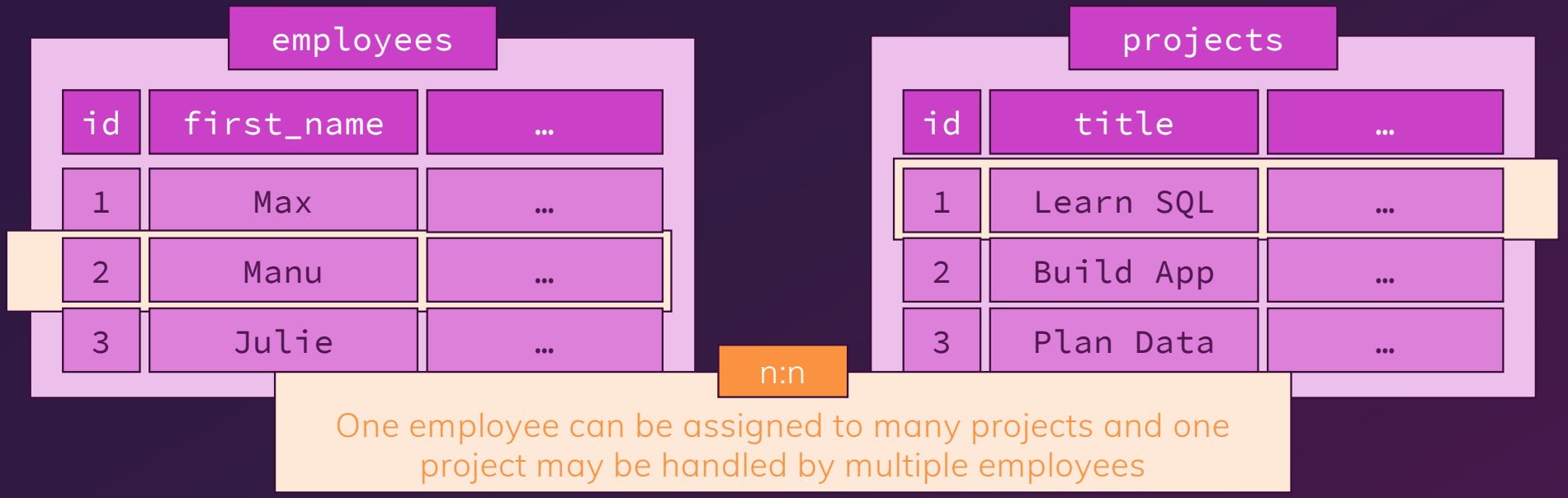
Many-To-Many Relations Need Intermediate Tables



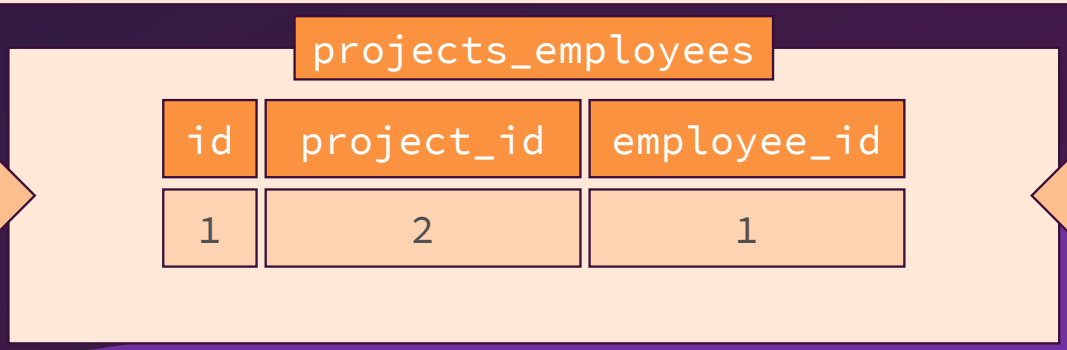
n:n

One employee can be assigned to many projects and one project may be handled by multiple employees

Many-To-Many Relations Need Intermediate Tables



An “intermediate table” is created and used to store the relations between “employees” and “projects”



One row per relation between the two “main tables”