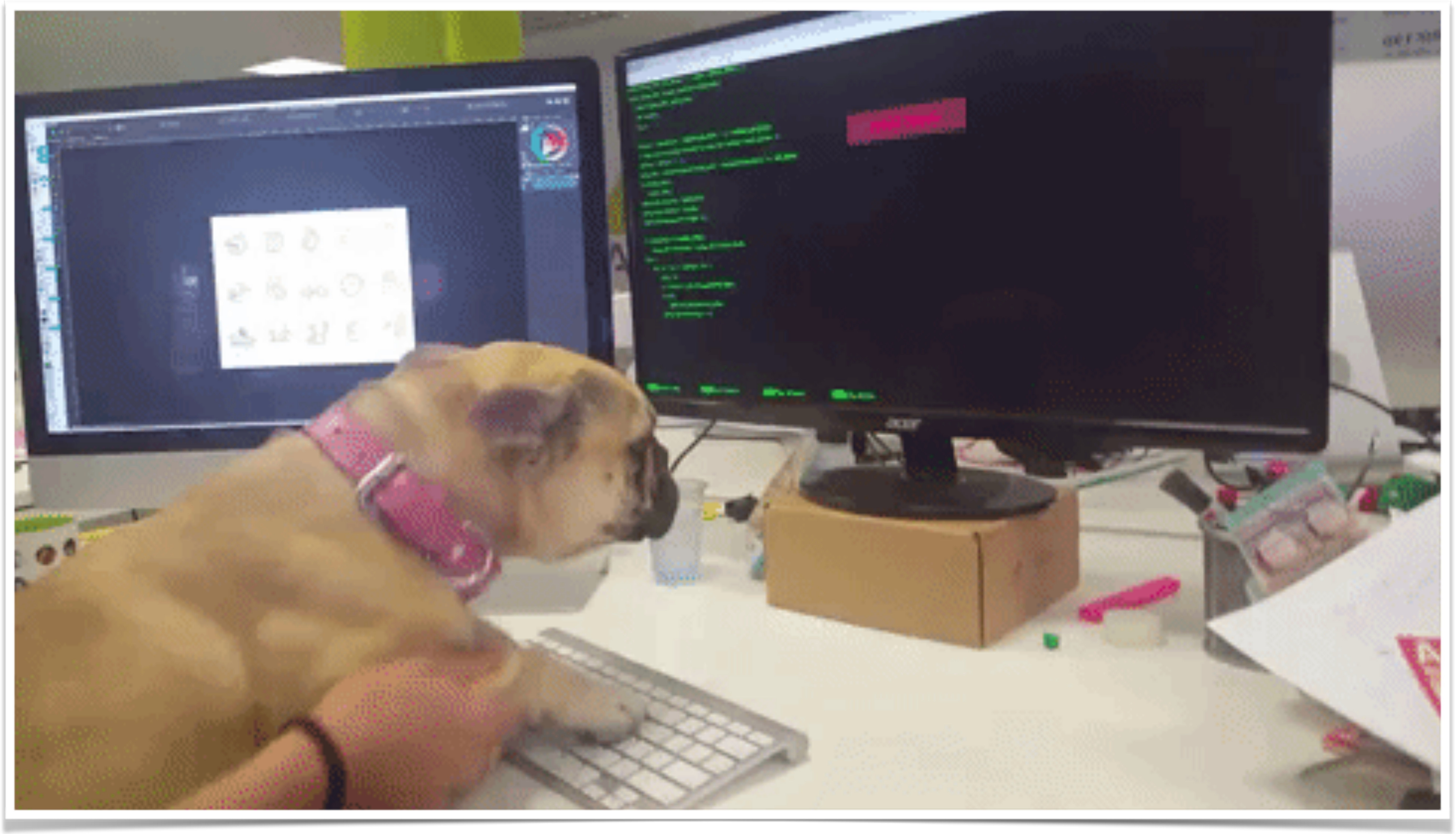
Let's SQL



Workshop Plan

PART 1

What is SQL

PART 2

Installation + Hello SQL

PART 3

Simple SQL Queries

PART 4

Advanced Queries / Filters

Workshop #2

SQL Joins

Working with multiple tables

What is SQL?

- Structured Query Language
- A programming language used to maintain and query Databases - DBMS
- Specially RDBMS

Ok! WTF is a RDBMS?

- Relational Database Management System
- A system of storing, organizing & managing data.
- Software service that runs on a server
- MySQL, Oracle, MS SQL Server, PostgreSQL, etc.

Storing data RDBMS style

- Stores data in tables

Albums Table

Id	year	album	artist	genre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
2	1975	Born to Run	Bruce Springsteen	Rock
3	1971	What's Going On	Marvin Gaye	Funk / Soul
4	1990	The Complete Recordings	Robert Johnson	Blues

Storing data RDBMS style

- Stores data in tables
- A table is organised in columns and rows

Albums Table

COLUMN

Id	year	album	artist	genre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
2	1975	Born to Run	Bruce Springsteen	Rock
3	1971	What's Going On	Marvin Gaye	Funk / Soul
4	1990	The Complete Recordings	Robert Johnson	Blues

ROW

Storing data RDBMS style

- Stores data in tables
- A table is organised in columns and rows
- Columns have types

Albums Table

Numeric

Text

Id	year	album	artist	genre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
2	1975	Born to Run	Bruce Springsteen	Rock
3	1971	What's Going On	Marvin Gaye	Funk / Soul
4	1990	The Complete Recordings	Robert Johnson	Blues

Storing data RDBMS style

- Stores data in tables
- A table is organised in columns and rows
- Columns have types
- Each row represents one record in the table

Albums Table

	Id	year	album	artist	genre
ROW	1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
ROW	2	1975	Born to Run	Bruce Springsteen	Rock
ROW	3	1971	What's Going On	Marvin Gaye	Funk / Soul
ROW	4	1990	The Complete Recordings	Robert Johnson	Blues

Storing data RDBMS style

- Stores data in tables
- A table is organised in columns and rows
- Columns have types
- Each row represents one record in the table
- Each row/record is uniquely identified using a primary key, usually called id

Albums Table

Id	year	album	artist	genre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
2	1975	Born to Run	Bruce Springsteen	Rock
3	1971	What's Going On	Marvin Gaye	Funk / Soul
4	1990	The Complete Recordings	Robert Johnson	Blues

Relationships!

About Relationships

- Rows in one table can be connected to row(s) in other tables via relationships

Albums

Id	year	album	artist	genre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock
2	1975	Born to Run	Bruce Springsteen	Rock
3	1971	What's Going On	Marvin Gaye	Funk / Soul
4	1990	The Complete Recordings	Robert Johnson	Blues

Albums

Id	year	album	artist	genreId
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	4
2	1975	Born to Run	Bruce Springsteen	4
3	1971	What's Going On	Marvin Gaye	2
4	1990	The Complete Recordings	Robert Johnson	1

Genres

Id	genre
1	Blues
2	Funk / Soul
3	Pop
4	Rock

About Relationships

- Rows in one table can be connected to row(s) in other tables via relationships

- To refer to values in another table the primary key (id) of that foreign row is stored instead of the entire value.

Albums

Id	year	album	artist	genreId
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	4
2	1975	Born to Run	Bruce Springsteen	4
3	1971	What's Going On	Marvin Gaye	2
4	1990	The Complete Recordings	Robert Johnson	1

Genres

Id	genre
1	Blues
2	Funk / Soul
3	Pop
4	Rock

About Relationships

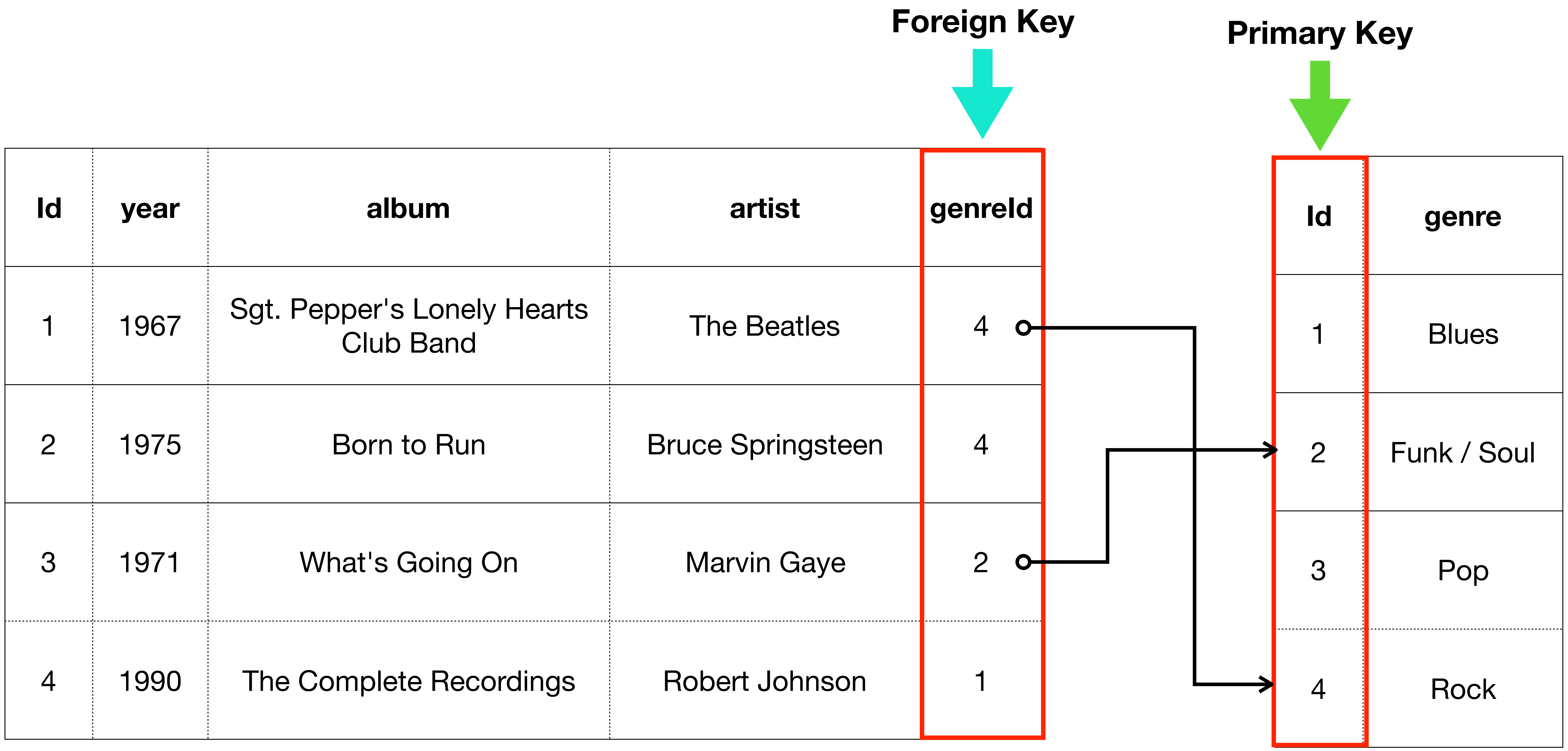
- Rows in one table can be connected to row(s) in other tables via relationships

- To refer to values in another table the primary key (id) of that foreign row is stored instead of the entire value.

- This id is called a foreign key as it references another (a foreign) table

Albums

Genres



Id	year	album	artist	genreId
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	4
2	1975	Born to Run	Bruce Springsteen	4
3	1971	What's Going On	Marvin Gaye	2
4	1990	The Complete Recordings	Robert Johnson	1

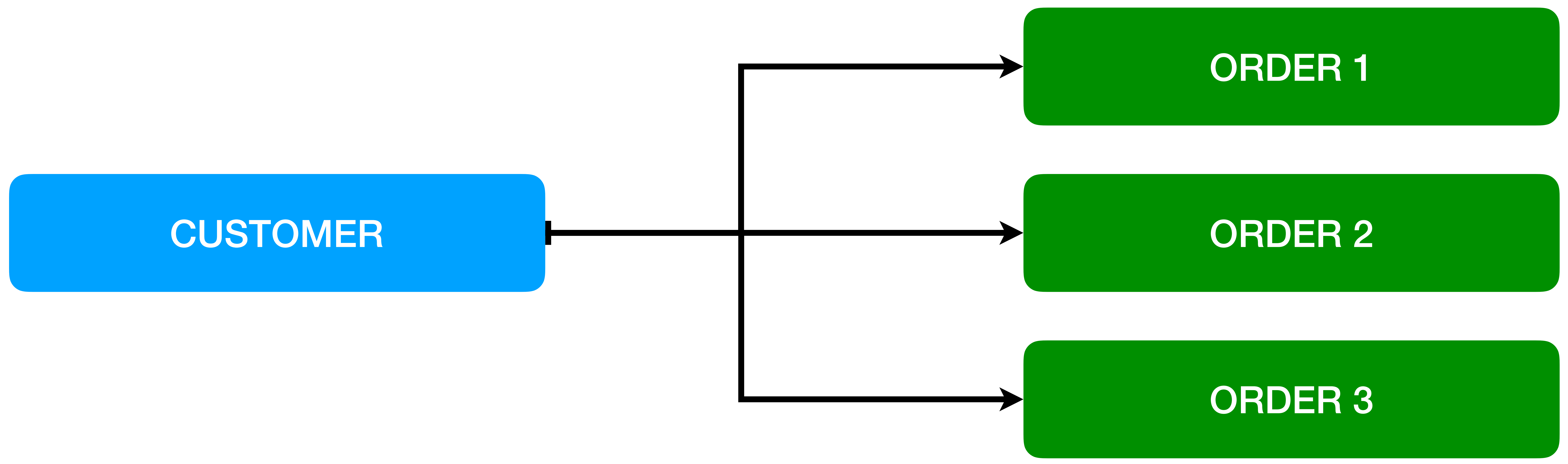
Id	genre
1	Blues
2	Funk / Soul
3	Pop
4	Rock

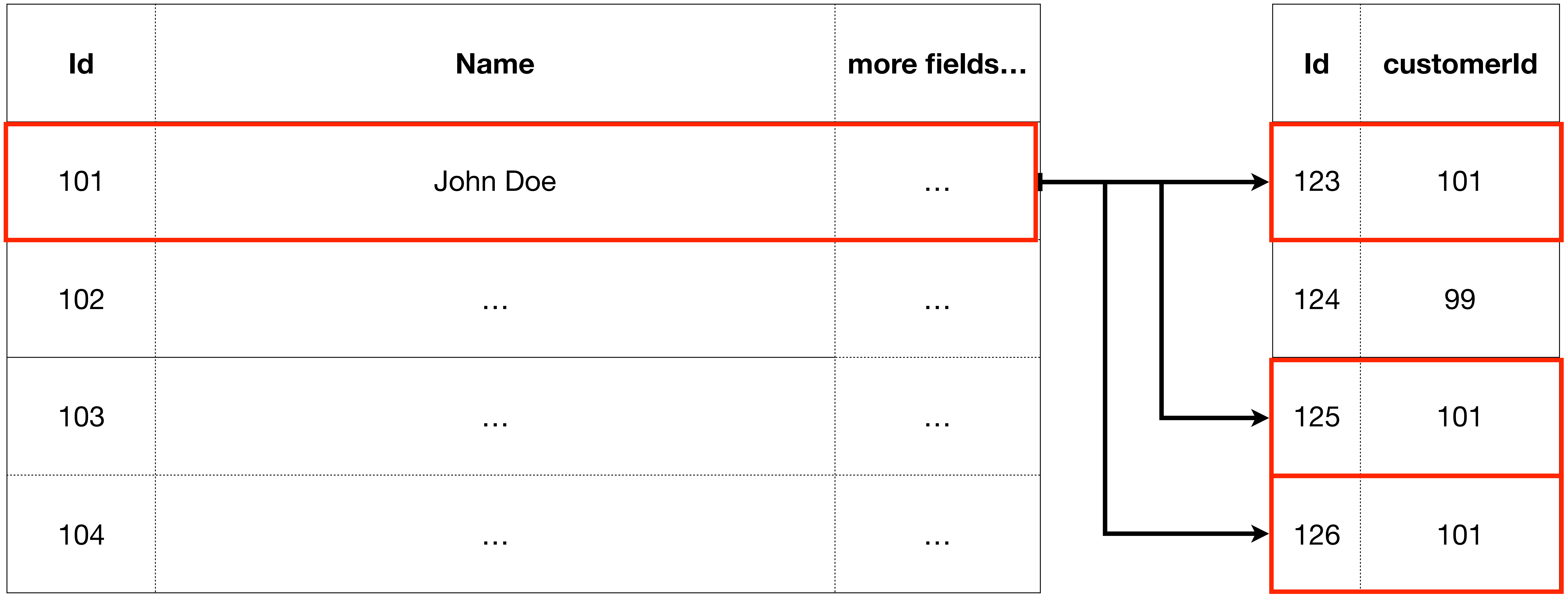
Types of Relationships

Complex relationships are built using this approach

- One to Many

One to Many





Customer

Id	Name	more fields…
101	John Doe	…
102	…	…
103	…	…
104	…	…

Order

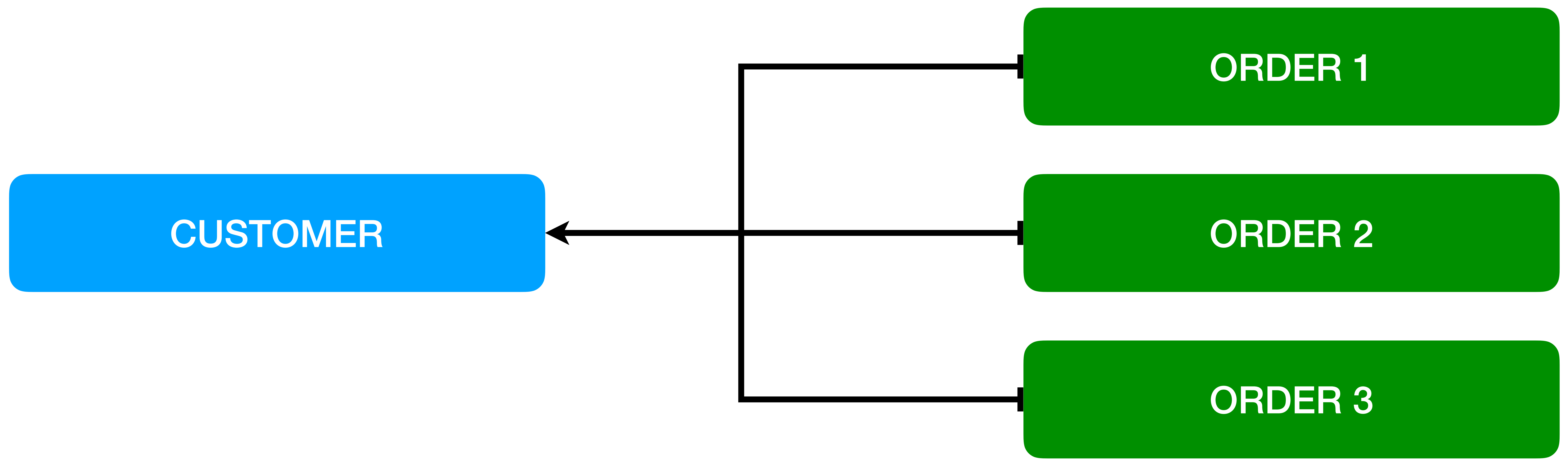
Id	customerId
123	101
124	99
125	101
126	101

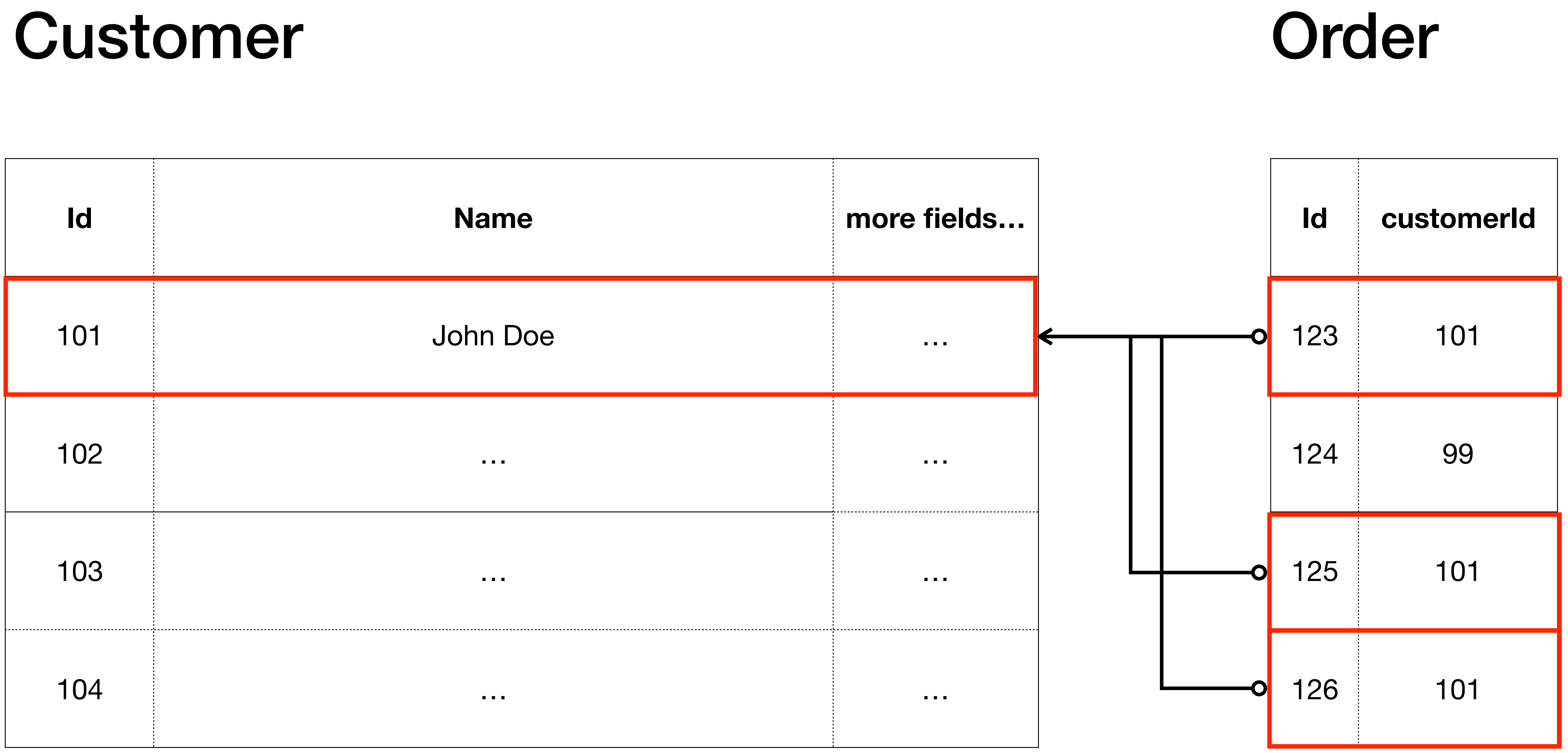
Types of Relationships

Complex relationships are built using this approach

- One to Many
- Many to One

Many to One





Customer

Id	Name	more fields…
101	John Doe	…
102	…	…
103	…	…
104	…	…

Order

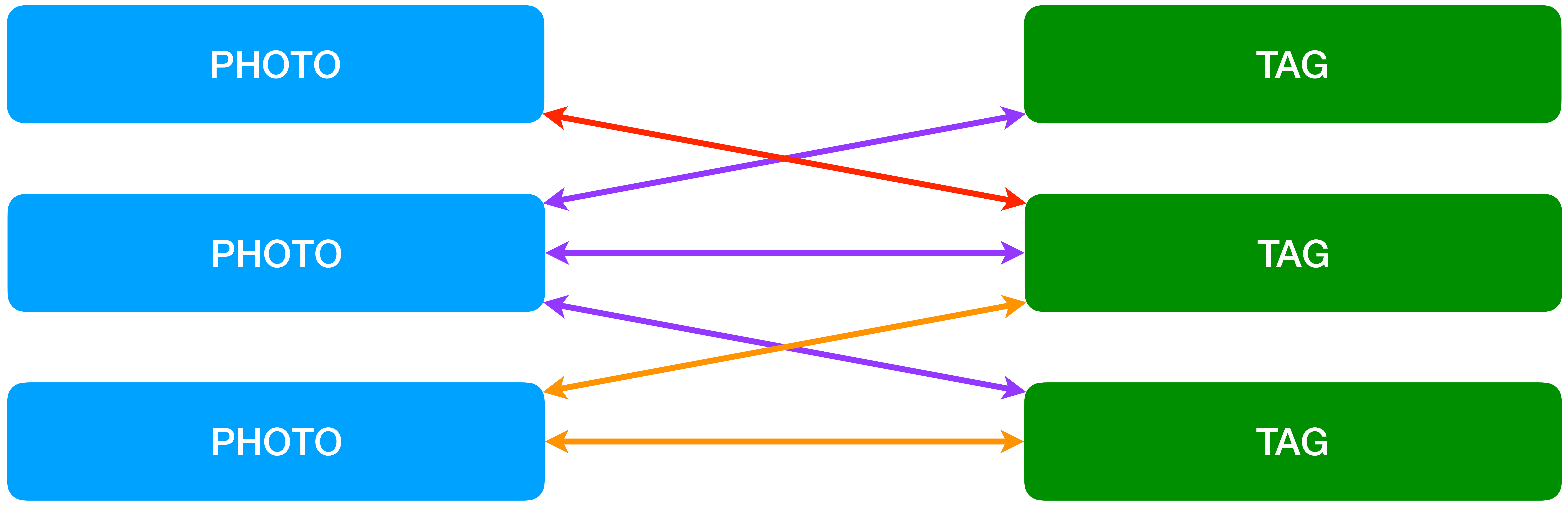
Id	customerId
123	101
124	99
125	101
126	101

Types of Relationships

Complex relationships are built using this approach

- One to Many
- Many to One
- Many to Many

Many to Many



Photo

Id	more fields…
101	…
102	…
103	…
104	…

PhotoTag

photoId	tagId
101	126
102	125
101	123
101	125

Tag

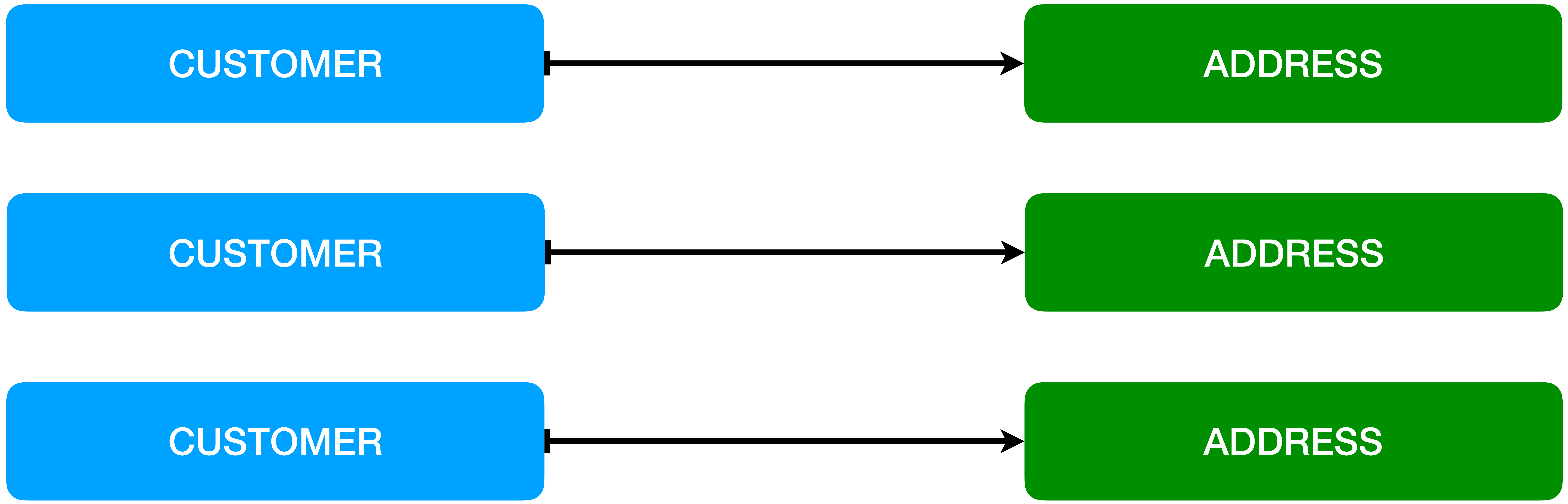
Id	Tag
123	#cool
124	#blah
125	#wearehomify
126	#yawn

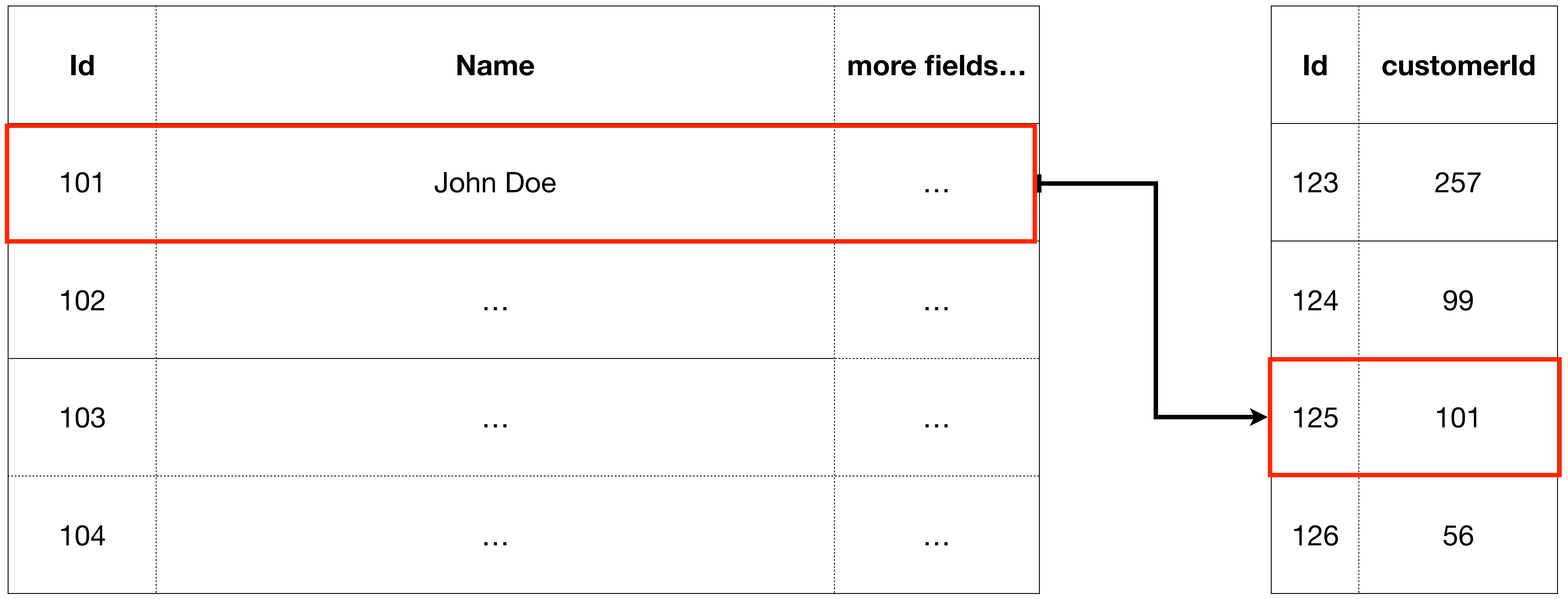
Types of Relationships

Complex relationships are built using this approach

- One to Many
- Many to One
- Many to Many
- One to One

One to One





Customer

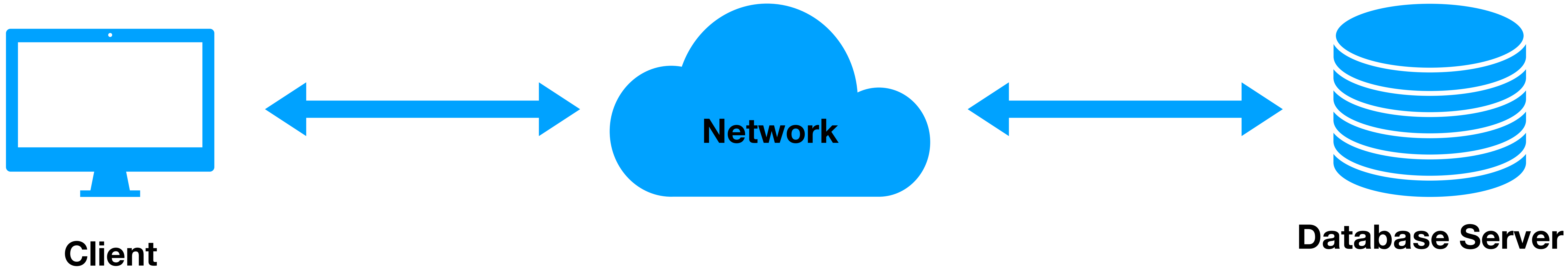
Id	Name	more fields…
101	John Doe	…
102	…	…
103	…	…
104	…	…

Address

Id	customerId
123	257
124	99
125	101
126	56

STEVIE
WANNA TAKE
A BREAK?

How to Connect



MySQL
Workbench

MySQL
Server

Let’s Install !

https://github.com/mouselangelo/sql-workshop

Connect & Verify

Import Data

HELLO SQL!

SELECT

Query database to fetch records matching the criteria specified

Syntax:

```
SELECT *
FROM <table>
```

*

Wildcard,
means “everything”

Example:

```
SELECT *
FROM albums
```

Convention:

```
SELECT *
FROM albums
```

SQL keywords in CAPS
Names of tables, columns, etc. in lowercase

Syntax:

```
SELECT <col1, col2, ...>
FROM <table>
```

Example:

```
SELECT album, artist, year
FROM albums
```

Exercise:

Get

Album, Artist, Genre

Solution:

```
SELECT album, artist, genre
FROM albums
```

GATORADE
FLOW
THIRST QUENCHER
PREDATORS HOCKEY
NHL
NHL.COM

DISTINCT

Finds rows with unique values for column / columns
(Skips duplicates)

Syntax:

```
SELECT DISTINCT<column(s)>
FROM <table>
```

Example:

```
SELECT DISTINCT artist
FROM albums
```

Exercise:

Try it with:

Genre

Solution:

```
SELECT DISTINCT genre
FROM albums
```

Example:

```
SELECT DISTINCT year, artist
FROM albums
```

For multiple columns - DISTINCT selects unique combinations of all column values

Exercise:

Try it with:

Genre, Artist

Solution:

```
SELECT DISTINCT genre, artist
FROM albums
```

WHERE

Specifying conditions / filters

Syntax:

```
SELECT <column(s)>
FROM <table>
[WHERE <conditions>]
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year = 1956
```

Example:

```
SELECT album, artist, genre
FROM albums
WHERE genre = “Rock”
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year = 1956
```

Example:

```
SELECT album, artist, genre
FROM albums
WHERE genre = “Rock”
```

Operators for WHERE

Operator	Description	Examples
=	Equal	year = 1970 genre = ‘Rock’
>	Greater than	year > 2000
<	Less than	year < 2000
>=	Greater than or Equal to	artist >= “Eagles"
<=	Less than or Equal to	artist <= “Eagles"
!=	Not Equal to	year != 1970
<>		artist <> “Eagles"

Example:

```
SELECT album, artist, genre
FROM albums
WHERE genre != “Rock”
```

LIMIT

Limit number of results

Syntax:

```
SELECT <column(s)>
FROM <table>
[WHERE <conditions>]
[LIMIT <count>]
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year > 1956
LIMIT 10
```

OFFSET

Skip some results

Syntax:

```
SELECT <column(s)>
FROM <table>
[WHERE <conditions>]
[OFFSET <count>]
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year > 1956
OFFSET 10
```

PAGINATION

Using LIMIT & OFFSET

Syntax:

```
SELECT <column(s)>
FROM <table>
[WHERE <conditions>]
[LIMIT <count>]
[OFFSET <count>]
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year > 1956
LIMIT 10
OFFSET 10
```

ORDER BY

Sorting the results in an order

Syntax:

```
SELECT <column(s)>
FROM <table>
[WHERE <conditions>]
[ORDER BY <fields [ASC|DESC]>]
```

Example:

```
SELECT album, artist, year
FROM albums
ORDER BY artist
```

Example:

```
SELECT album, artist, year
FROM albums
ORDER BY artist DESC
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE year > 1956
ORDER BY year DESC, artist ASC
LIMIT 10
```

More Operators for WHERE!

Operator	Description	Examples
BETWEEN	Value is between a certain range	year BETWEEN 1970 AND 1980
LIKE	Search for a pattern. % (percent) for multiple characters _ (underscore) for a single character	artist like “the bea%” artist like “A___” artist like “__”
IN	Match a list of values	genre IN (“Rock”, “Pop”) artist IN (SELECT …)

Example:

```
SELECT year, artist, album
FROM albums
WHERE
year BETWEEN 1950 AND 1959
```

Example:

```
SELECT DISTINCT artist
FROM albums
WHERE artist LIKE "The b%"
```

Example:

```
SELECT DISTINCT artist
FROM albums
WHERE artist LIKE "__"
```

Example:

```
SELECT DISTINCT genre
FROM albums
WHERE
genre IN
("Rock", "Pop", "Country")
```

Example:

```
SELECT DISTINCT year, artist
FROM albums
WHERE year IN
(SELECT year
FROM albums
WHERE year
BETWEEN 1950 AND 1959)
```

Aliases

expression AS name

Syntax:

```
SELECT col AS <name>, ….
FROM <table>
```

Example:

```
SELECT album, artist,
 year AS `released in`
FROM albums
WHERE
genre = "Rock"
```

AGGREGATE FUNCTIONS

Operator	Description	Examples
COUNT	Count of results	SELECT COUNT(album) FROM …
SUM	Sum of numeric values	SUM(amount)
AVG	Average of numeric values	AVG(rating)
MIN	Minimum	MIN(rating)
MAX	Maximum	MAX(rating)

Example:

```
SELECT COUNT(album)
FROM albums
WHERE year = 1980
```

Example:

```
SELECT SUM(sales)
FROM albums
WHERE artist = "The Beatles"
```

Example:

```
SELECT avg(rating)
FROM albums
WHERE artist = "The Beatles"
```

Example:

```
SELECT
min(rating), max(rating),
avg(rating)
FROM albums
WHERE artist = "The Beatles"
```

Example:

```
SELECT
min(rating) AS `min rating`,
max(rating) AS `max rating`,
avg(rating) AS `average rating`
FROM albums
WHERE artist = "The Beatles"
```

Multiple WHERE conditions

Specify more than one filter / condition using AND, OR, NOT

Example:

```
SELECT album, artist
FROM albums
WHERE
genre = "Rock"
AND
year = 1980
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE
artist LIKE "X%"
OR
artist LIKE "Z%"
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE
NOT
year BETWEEN 1950 AND 2007
```

Example:

```
SELECT album, artist, year
FROM albums
WHERE
(artist = "The Beatles" OR artist =
"Bob Dylan")
AND
NOT (year BETWEEN 1950 AND 1967)
```

GROUP BY

Grouping on values

Syntax:

```
SELECT <column(s)>
FROM <table>
[GROUP BY <fields>]
```

Example:

```
SELECT artist,
COUNT(album) AS
num_albums
FROM albums
GROUP BY artist
```

MIKASA

Workshop #2

SQL Joins

Working with multiple tables