FSU | RESEARCH COMPUTING CENTER
INFORMATION TECHNOLOGY SERVICES

Intermediate databases and SQL

February 17, 2026 □ Casey McLaughlin

Overview □ Questions that we will cover today*

- Review of Intro to SQL materials (quickly!)
- How do I model a many-to-many relationship?
- What is database normalization?
    - First normal form
    - Second normal form
    - Third normal form
- ER Diagrams
- Multi-table joins in SQL
- Aggregate results in SQL
- Virtual columns in SQL

What we covered in the First Workshop

Overview □ Questions that we covered in the intro workshop

- What is a relational database?
- What is a relational database management system?
- How do I model a basic database?
- What are tables and keys?
- What are primary, surrogate, and foreign keys?
- What is SQL?
- How do I write SQL to query data in a database?
- How do I write SQL to query data from multiple tables?
- How do I write SQL to add and update data in a database?

What is a Relational Database?

- “A database based on the relational model of data, as proposed by E.F. Codd in 1970.” —Wikipedia

- “A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data.” — Also Wikipedia

What is SQL?

- SQL = “Structured Query Language”
- The primary way for querying relational databases

RDBMS □ Relational Database Management System

Servers:

- $$ - Microsoft SQL Server
- $$ - Microsoft Access
- $$ - Oracle Database
- Free - MySQL and MariaDB
- Free - Postgres
- Free - DuckDB
- *Many more...*

Clients:

- $$ - Microsoft Access
- $$ - JetBrains DataGrip
- Free - HeidiSQL (Windows/Linux)
- Free - Sequel Pro (Mac)
- Free - Terminal access
- Free - Programming libraries
- *Many more...*

Let’s build a relational database!

XYZ Restaurant

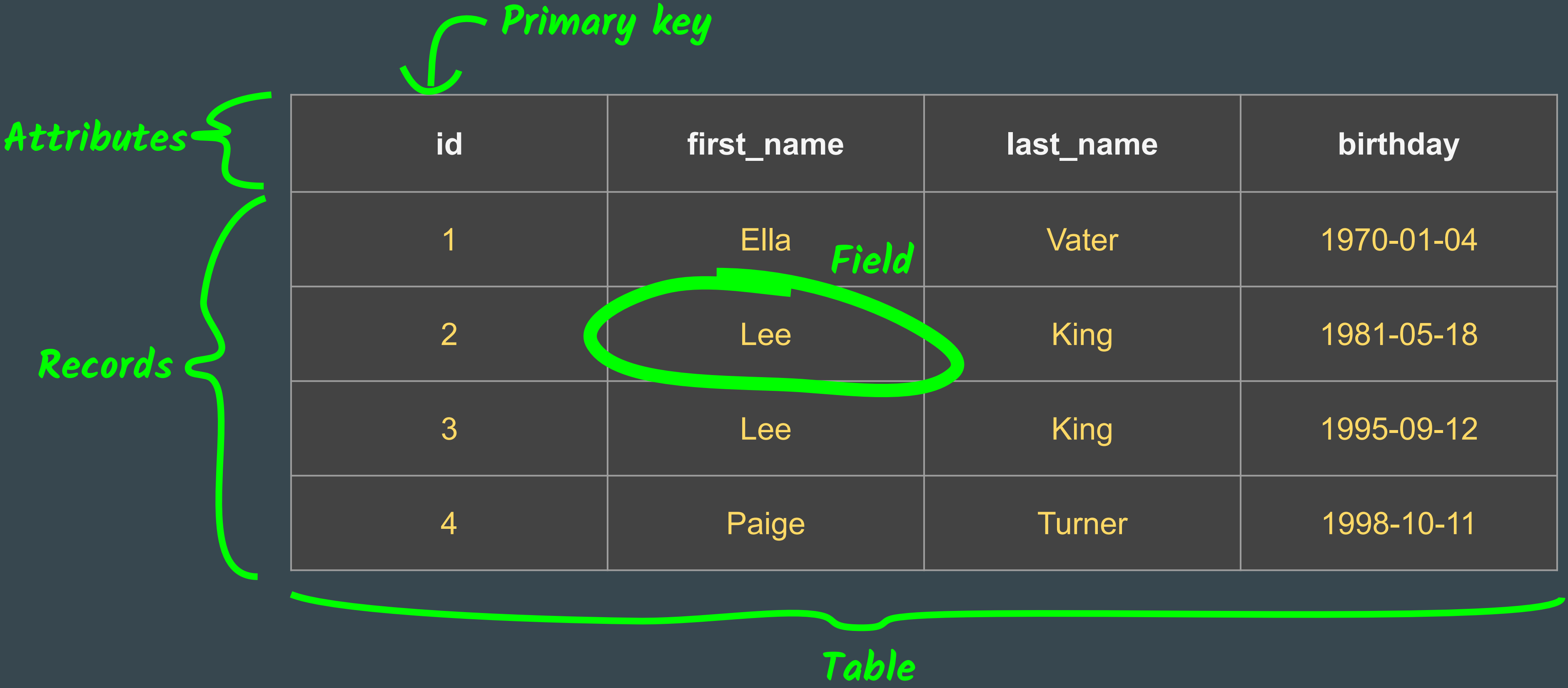
“XYZ Restaurant has a need to track customers’ names, email addresses, birthdays, and favorite dishes.”

Spreadsheet vs. Relational Data Terminology

Spreadsheet	**Relational Database**
Workbook	**Schema**
Worksheet	**Table**
Row	**Record** or **Tuple**
Column	**Attribute**
Cell	**Field**

Surrogate/synthetic keys should be arbitrary values

Relational Data Terminology Review

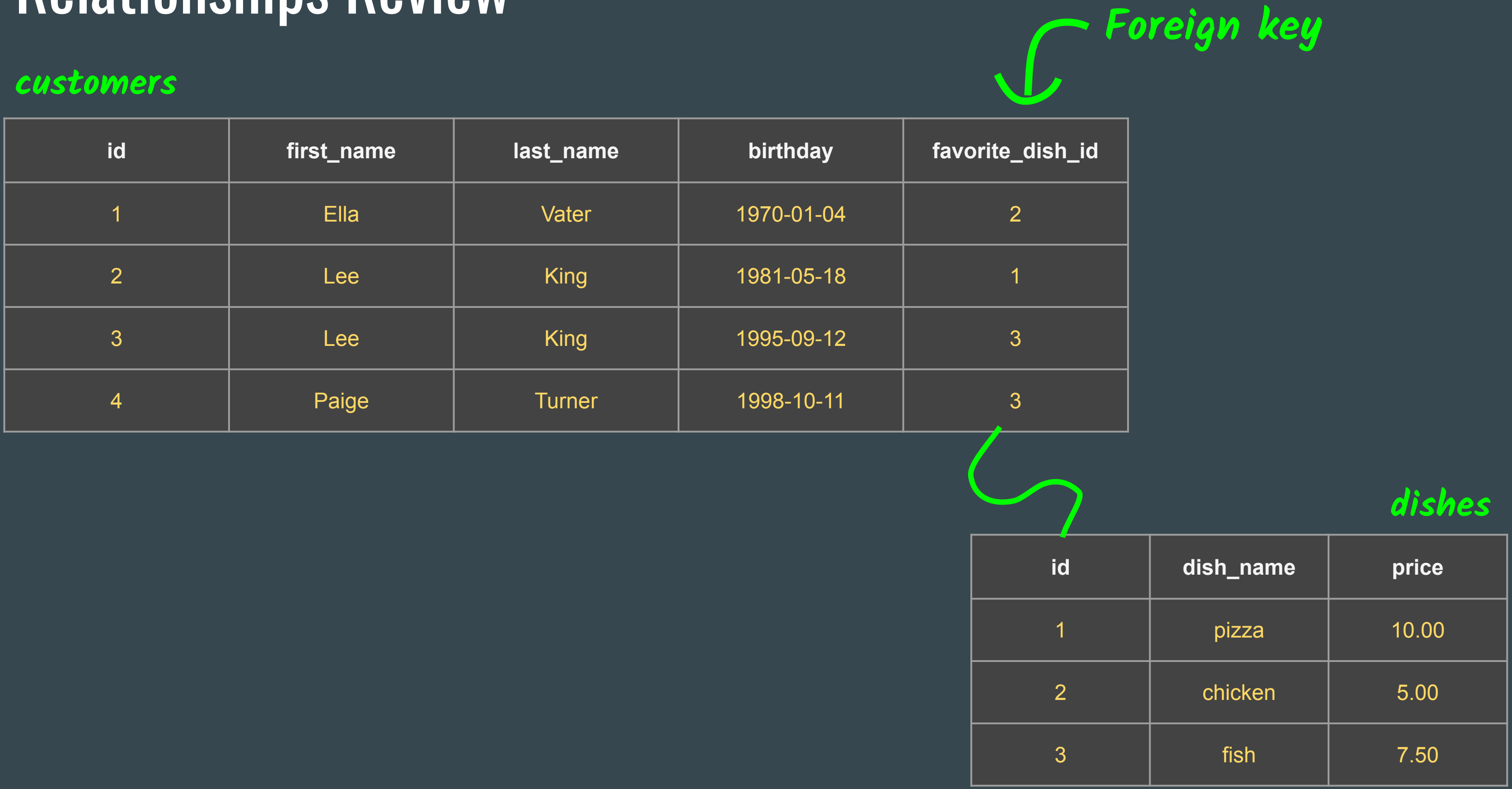


id	first_name	last_name	birthday
1	Ella	Vater	1970-01-04
2	Lee	King	1981-05-18
3	Lee	King	1995-09-12
4	Paige	Turner	1998-10-11

Normalization

the process of organizing the columns (attributes) and tables of a relational database to minimize data redundancy and improve data integrity

Relationships Review



customers

Foreign key

id	first_name	last_name	birthday	favorite_dish_id
1	Ella	Vater	1970-01-04	2
2	Lee	King	1981-05-18	1
3	Lee	King	1995-09-12	3
4	Paige	Turner	1998-10-11	3

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

Common RDMS Data Types (non-comprehensive list)

Type	Description
char	Fixed-length string of characters (e.g., “abc”)
varchar	Variable-length string of characters (e.g. “lorum ipsum”)
text	Long string of characters (difficult to index)
int	A whole number (positive or negative) (e.g., -235, 100)
float	A number with a decimal (e.g., -240.24, 93.12)
datetime	A point in time (e.g., 2025-01-05 09:03:23)
tinyint	Either a one or a zero; typically used for true/false values

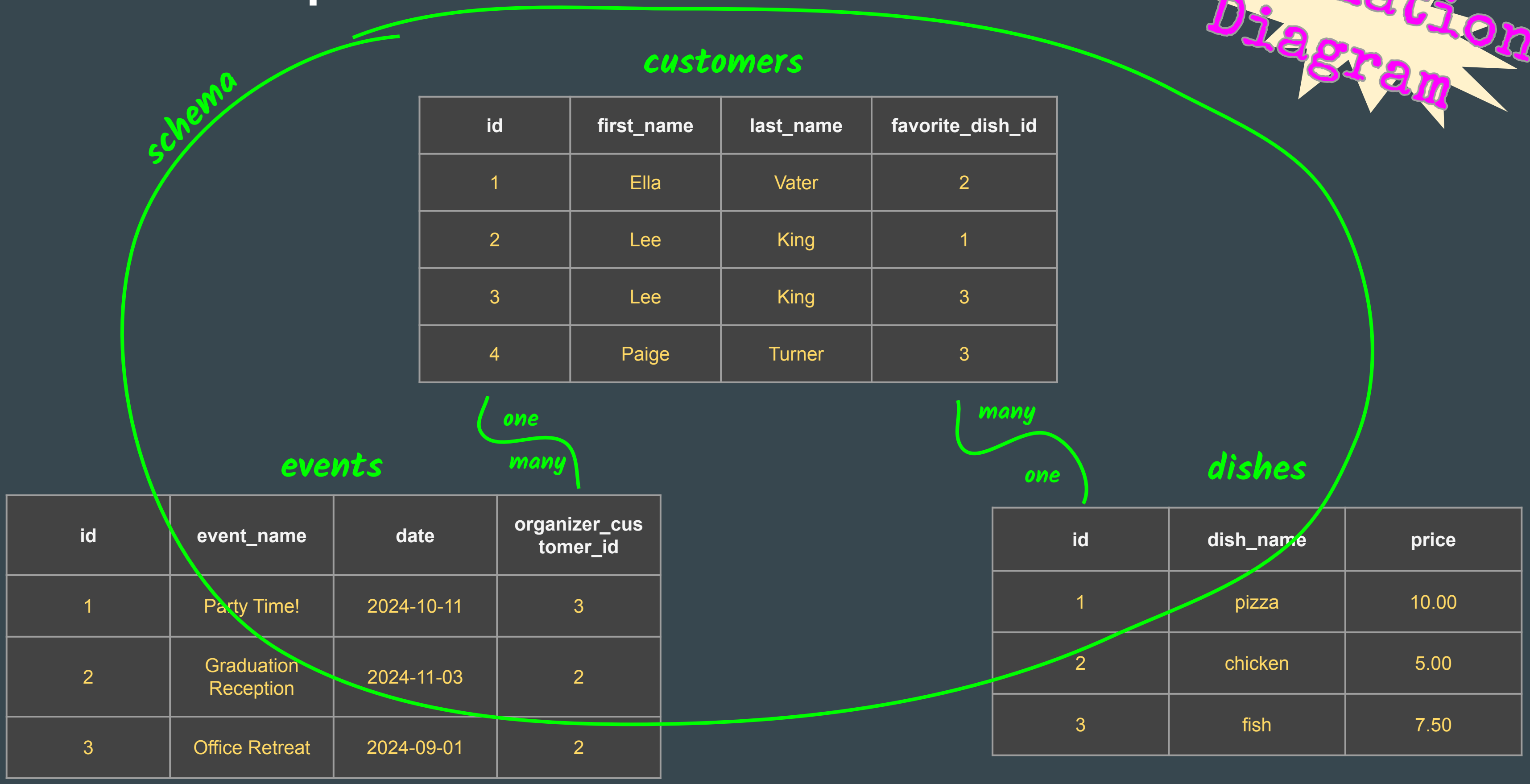
XYZ Restaurant

“XYZ Restaurant has a need to track customers’ names, email addresses, birthdays, and favorite dishes.”

“XYZ Restaurant also hosts events, and would like to keep track of names, date/times, and contacts for events.”

Relationships Review

Entity Relation (ER) Diagram



schema

customers

id	first_name	last_name	favorite_dish_id
1	Ella	Vater	2
2	Lee	King	1
3	Lee	King	3
4	Paige	Turner	3

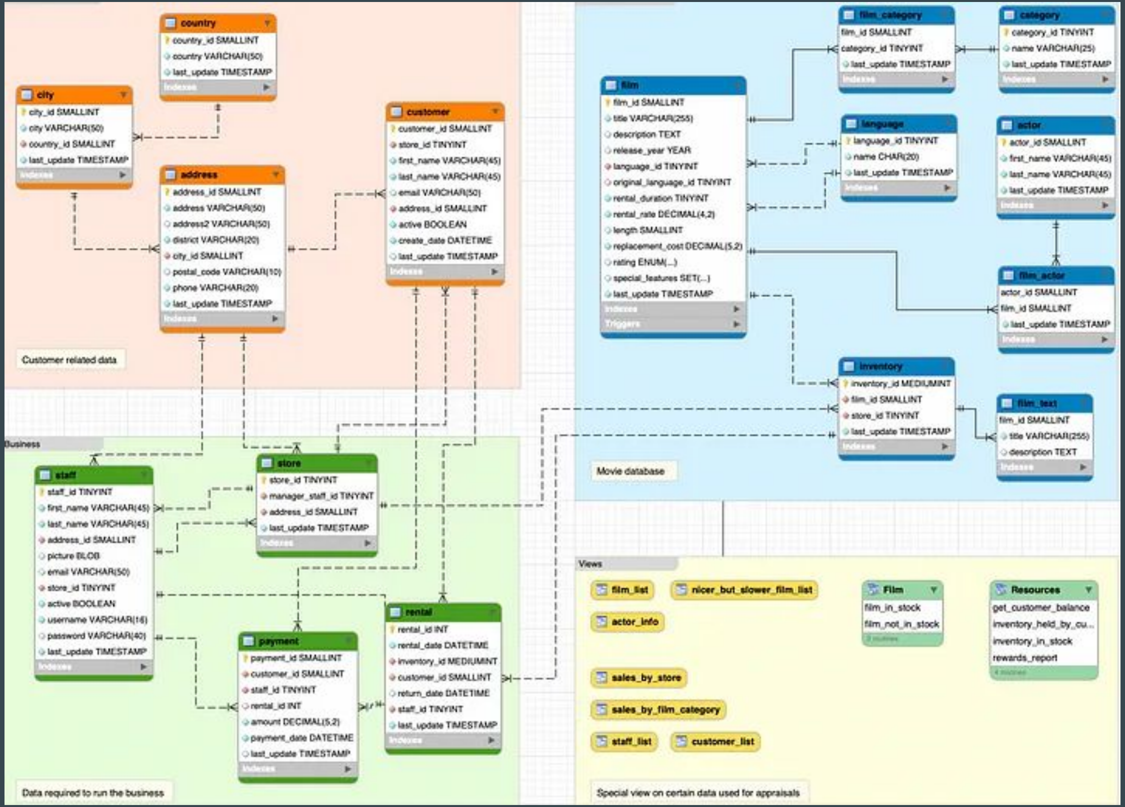
events

id	event_name	date	organizer_customer_id
1	Party Time!	2024-10-11	3
2	Graduation Reception	2024-11-03	2
3	Office Retreat	2024-09-01	2

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

Example Entity Relation Diagram



country
country_id SMALLINT
country VARCHAR(50)
last_update TIMESTAMP

city
city_id SMALLINT
city VARCHAR(50)
country_id SMALLINT
last_update TIMESTAMP

address
address_id SMALLINT
address VARCHAR(50)
address2 VARCHAR(50)
district VARCHAR(20)
city_id SMALLINT
postal_code VARCHAR(10)
phone VARCHAR(20)
last_update TIMESTAMP

customer
customer_id SMALLINT
store_id TINYINT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(50)
address_id SMALLINT
active BOOLEAN
create_date DATETIME
last_update TIMESTAMP

Customer related data

Business

staff
staff_id TINYINT
first_name VARCHAR(45)
last_name VARCHAR(45)
address_id SMALLINT
picture BLOB
email VARCHAR(50)
store_id TINYINT
active BOOLEAN
username VARCHAR(16)
password VARCHAR(40)
last_update TIMESTAMP

store
store_id TINYINT
manager_staff_id TINYINT
address_id SMALLINT
last_update TIMESTAMP

payment
payment_id SMALLINT
customer_id SMALLINT
staff_id TINYINT
rental_id INT
amount DECIMAL(5,2)
payment_date DATETIME
last_update TIMESTAMP

rental
rental_id INT
rental_date DATETIME
inventory_id MEDIUMINT
customer_id SMALLINT
return_date DATETIME
staff_id TINYINT
last_update TIMESTAMP

Data required to run the business

film
film_id SMALLINT
title VARCHAR(255)
description TEXT
release_year YEAR
language_id TINYINT
original_language_id TINYINT
rental_duration TINYINT
rental_rate DECIMAL(4,2)
length SMALLINT
replacement_cost DECIMAL(5,2)
rating ENUM(...)
special_features SET(...)
last_update TIMESTAMP

film_category
film_id SMALLINT
category_id TINYINT
last_update TIMESTAMP

category
category_id TINYINT
name VARCHAR(25)
last_update TIMESTAMP

language
language_id TINYINT
name CHAR(20)
last_update TIMESTAMP

actor
actor_id SMALLINT
first_name VARCHAR(45)
last_name VARCHAR(45)
last_update TIMESTAMP

film_actor
actor_id SMALLINT
film_id SMALLINT
last_update TIMESTAMP

inventory
inventory_id MEDIUMINT
film_id SMALLINT
store_id TINYINT
last_update TIMESTAMP

film_text
film_id SMALLINT
title VARCHAR(255)
description TEXT

Movie database

Views

film_list
nicer_but_slower_film_list
actor_info
sales_by_store
sales_by_film_category
staff_list
customer_list

Film
film_in_stock
film_not_in_stock

Resources
get_customer_balance
inventory_held_by_cu...
inventory_in_stock
rewards_report

Special view on certain data used for appraisals

Let’s write some SQL!

SQL Query Syntax Review

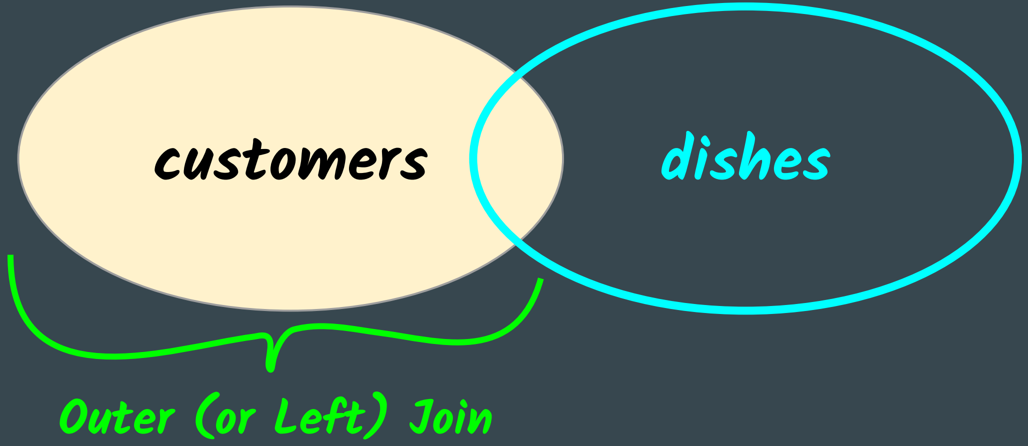
Condition (predicate)

```
SELECT first_name, last_name FROM customers WHERE state = 'FL';
```

- 🟨 Keyword
- ⬜ Attributes
- 🟩 Table
- 🟥 Operator
- 🟧 Literal

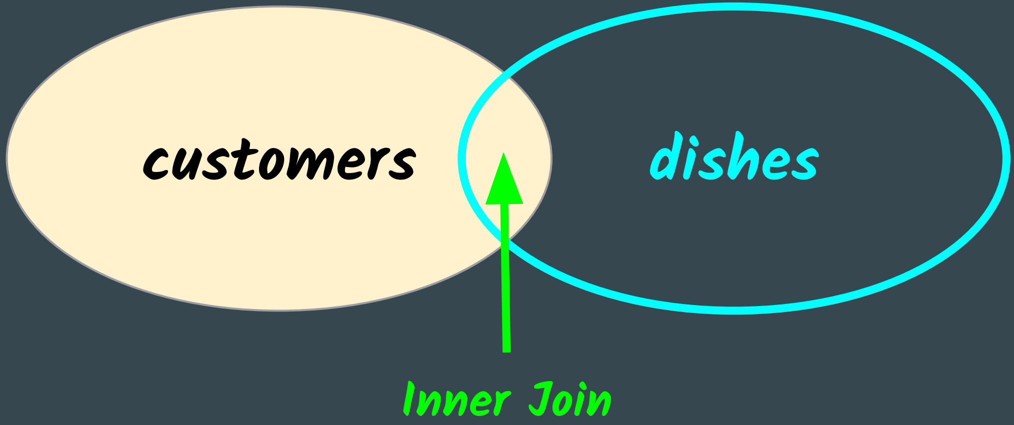
Outer/Left Joins

```
SELECT first_name, last_name

FROM customers LEFT JOIN dishes

ON customers.favorite_dish_id = dishes.id;
```



Inner Joins

```
SELECT first_name, last_name

FROM customers c INNER JOIN dishes d ON c.favorite_dish_id = d.id;
```



Aggregate Functions

```
SELECT COUNT(id) FROM customers;
```

```
SELECT SUM(price) AS total FROM dishes;
```

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM customers;
```

Modifying Data: Adding a record

```sql
INSERT INTO customers (first_name, last_name, email)

VALUES ('Bob', 'Jones', 'bjones@example.org');
```

Modifying Data: Updating a record

```sql
UPDATE customers SET first_name = 'Robert'

WHERE first_name = 'Bob' AND last_name = 'Jones';
```

```sql
UPDATE customers SET first_name = 'Robert' WHERE id = 26;
```

Modifying Data: Deleting a record

```
DELETE FROM customers WHERE id = 26;
```

Moving on…

What fields in the *orders* table?

- “XYZ Restaurant has a need to track customers’ names, email addresses, birthdays, and favorite dishes.”
- “XYZ Restaurant also hosts events, and would like to keep track of names, date/times, and contacts for events.”
- “XYZ Restaurant needs to track orders. Orders can include multiple dishes.”

What fields in the *orders* table?

- id
- customer_id
- datetime
- dishes

There can be many dishes in one order

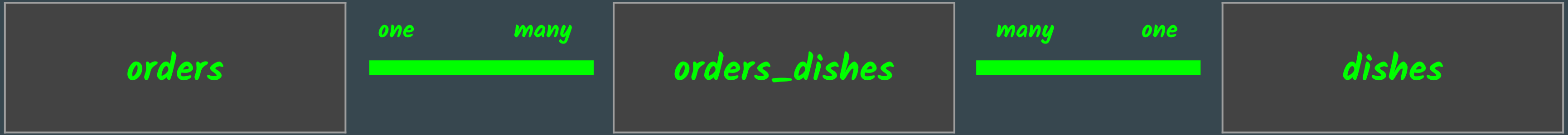
~~dish_1~~	~~dish_2~~	~~dish_3~~
~~pizza~~	~~fish~~	~~fish~~

~~dishes~~
~~pizza, fish, fish~~

What fields in the *orders* table?

- Orders can have multiple (or “many”) dishes
- Dishes are part of multiple (or “many”) orders

This is a **many-to-many** relationship.



orders

one

many

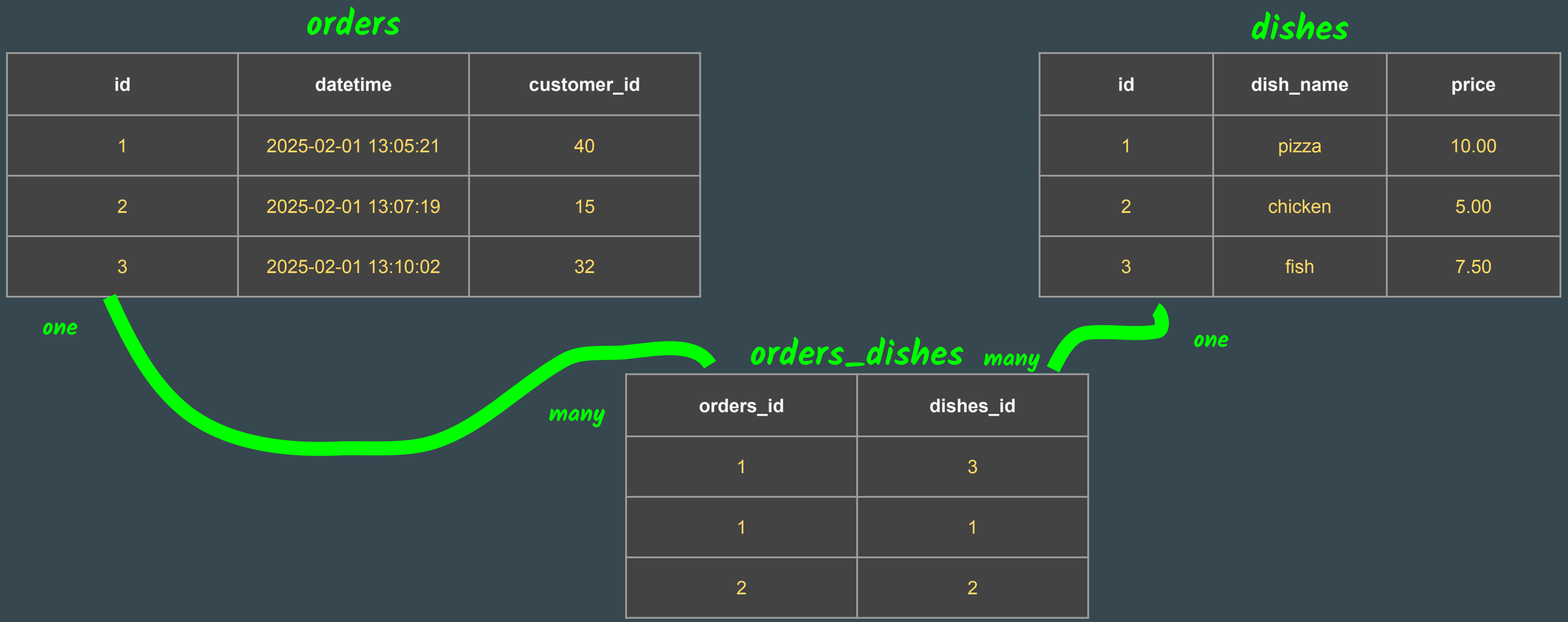
orders_dishes

many

one

dishes

Many-to-Many Relationships



orders

id	datetime	customer_id
1	2025-02-01 13:05:21	40
2	2025-02-01 13:07:19	15
3	2025-02-01 13:10:02	32

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

orders_dishes

orders_id	dishes_id
1	3
1	1
2	2

Let’s talk about normalization

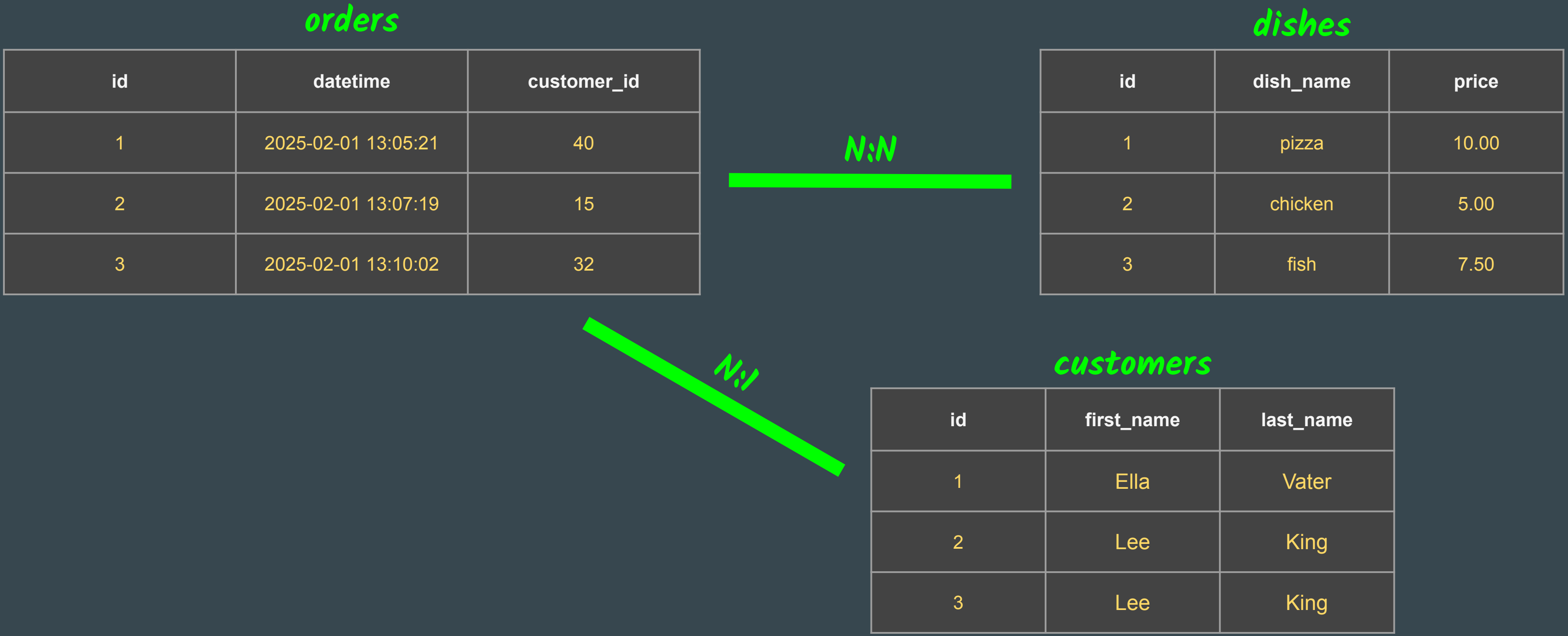
Let’s talk about normalization

- Normalization is a systematic process of organizing data in a relational database to reduce data redundancy and improve data integrity.

- It involves decomposing tables into smaller, well-structured tables and defining relationships between them according to specific rules.

- We are going to discuss three levels today: **first, second, and third normal forms** .

Many-to-Many Relationships ER diagram representation



orders

id	datetime	customer_id
1	2025-02-01 13:05:21	40
2	2025-02-01 13:07:19	15
3	2025-02-01 13:10:02	32

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

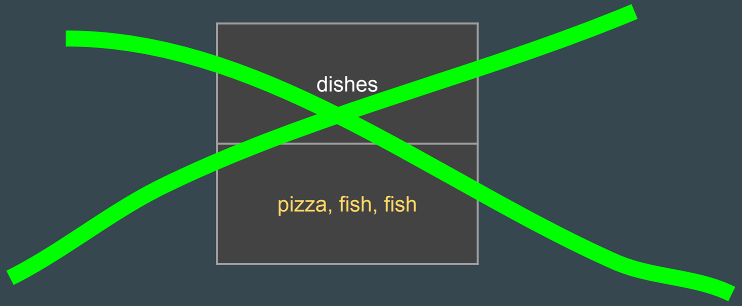
customers

id	first_name	last_name
1	Ella	Vater
2	Lee	King
3	Lee	King

Let’s talk about normalization

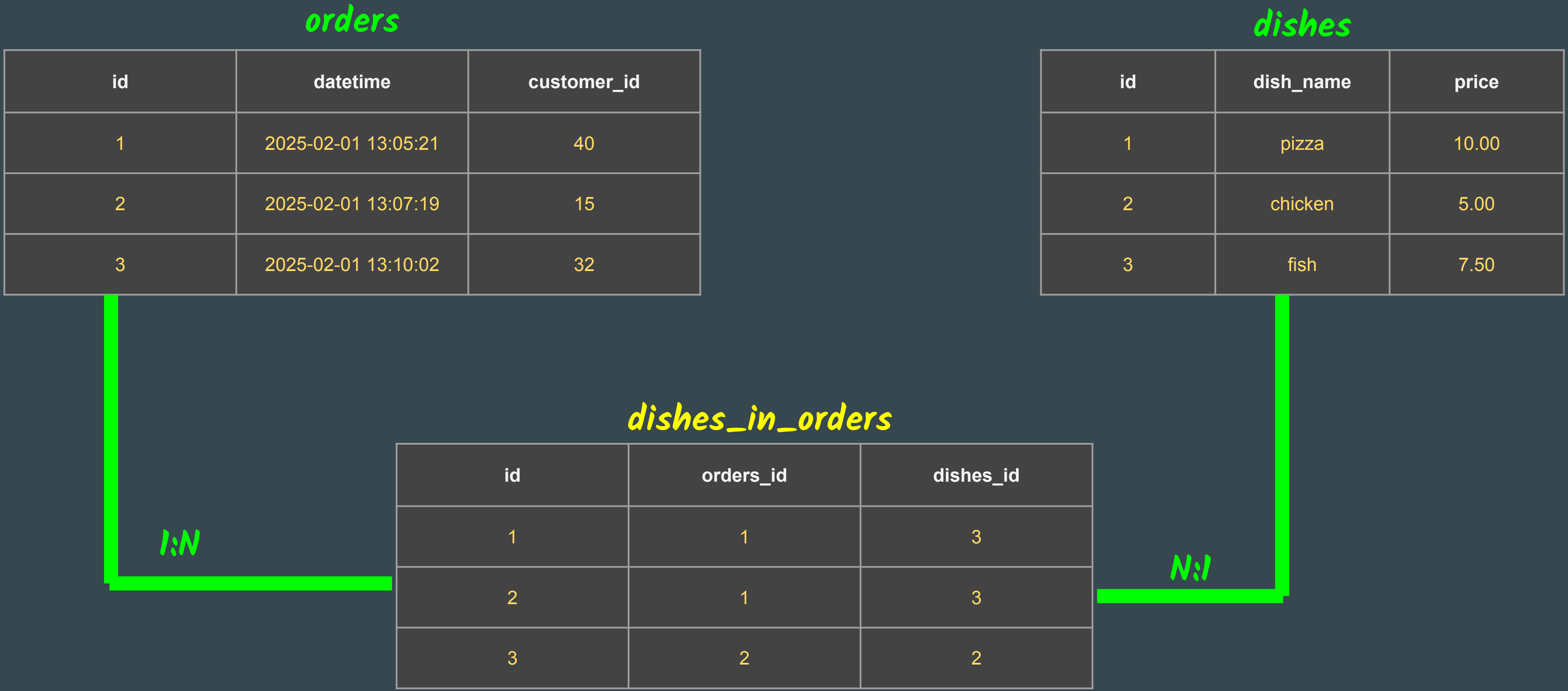
First Normal Form (1NF)

- Values in each cell should be atomic and tables should have no repeating groups.
- Row order should not matter.



dishes
pizza, fish, fish

1NF Violations and Fixes



orders

id	datetime	customer_id
1	2025-02-01 13:05:21	40
2	2025-02-01 13:07:19	15
3	2025-02-01 13:10:02	32

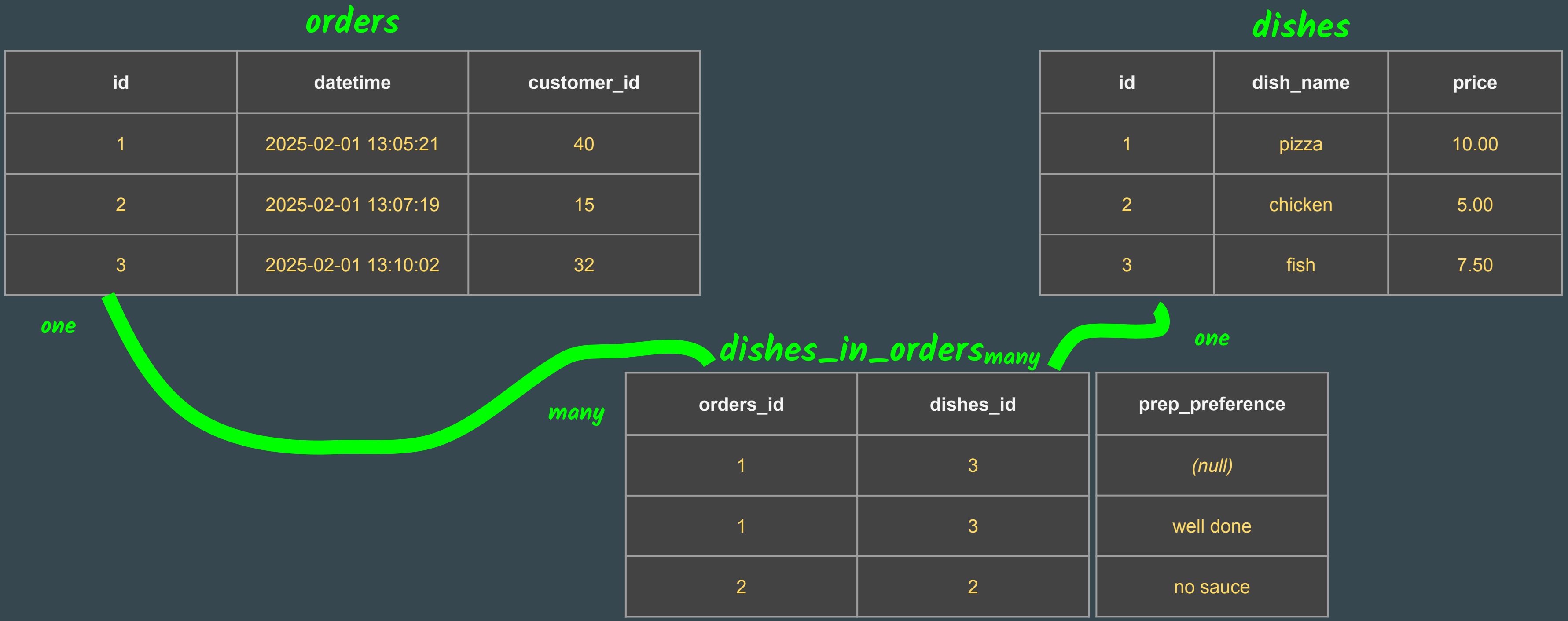
dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

dishes_in_orders

id	orders_id	dishes_id
1	1	3
2	1	3
3	2	2

Many-to-Many Relationships



orders

id	datetime	customer_id
1	2025-02-01 13:05:21	40
2	2025-02-01 13:07:19	15
3	2025-02-01 13:10:02	32

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

dishes_in_orders

orders_id	dishes_id	prep_preference
1	3	*(null)*
1	3	well done
2	2	no sauce

Let’s talk about normalization

First Normal Form (1NF) “Smells”

- Columns with multiple values
- Join tables without primary keys

Let’s talk about normalization

Second Normal Form (2NF)

- No value in a table should depend on only part of a key that can be used to uniquely identify a row.

Let’s talk about normalization

Second Normal Form (2NF)

events

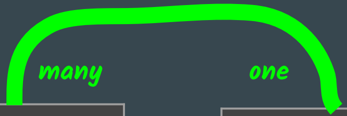
id	event_name	date	customers_id	location
1	Party Time!	2024-10-11	3	Main Dining Room Level 1
2	Graduation Reception	2024-11-03	2	Main Dining Room Level 2
3	Office Retreat	2024-09-01	2	Terrace
4	Office Retreat	2024-10-01	2	Terrace
5	Office Retreat	2024-11-01	2	Main Dining Room Level 1
6	Graduation Reception	2024-11-01	7	Terrace

Let's talk about normalization

Second Normal Form (2NF)

events

id	event_name	date	customer_id	location_id
1	Party Time!	2024-10-11	3	1
2	Graduation Reception	2024-11-03	2	2
3	Office Retreat	2024-09-01	2	3
4	Office Retreat	2024-10-01	2	3
5	Office Retreat	2024-11-01	2	2
6	Graduation Reception	2024-11-01	7	3



event_locations

id	name	price
1	Main Dining Room Level 1	200
2	Main Dining Room Level 2	250
3	Terrace	400

Let’s talk about normalization

Second Normal Form (2NF) “Smells”

- Lots of repeating data in tables

Let’s talk about normalization

Third Normal Form (3NF)

- Values should not be stored if they can be calculated from another non-key field.

3rd Normal Form

- “XYZ Restaurant has a need to track customers’ names, email addresses, birthdays, and favorite dishes.”
- “XYZ Restaurant also hosts events, and would like to keep track of names, date/times, and contacts for events.”
- “XYZ Restaurant needs to track orders. Orders can include multiple dishes.”
- “XYZ Restaurant offers a senior discount of 50% off normal menu prices.”

3rd Normal Form

dishes

id	dish_name	price	~~senior_price~~
1	pizza	10.00	~~5.00~~
2	chicken	5.00	~~2.50~~
3	fish	7.50	~~3.75~~

Let’s talk about normalization

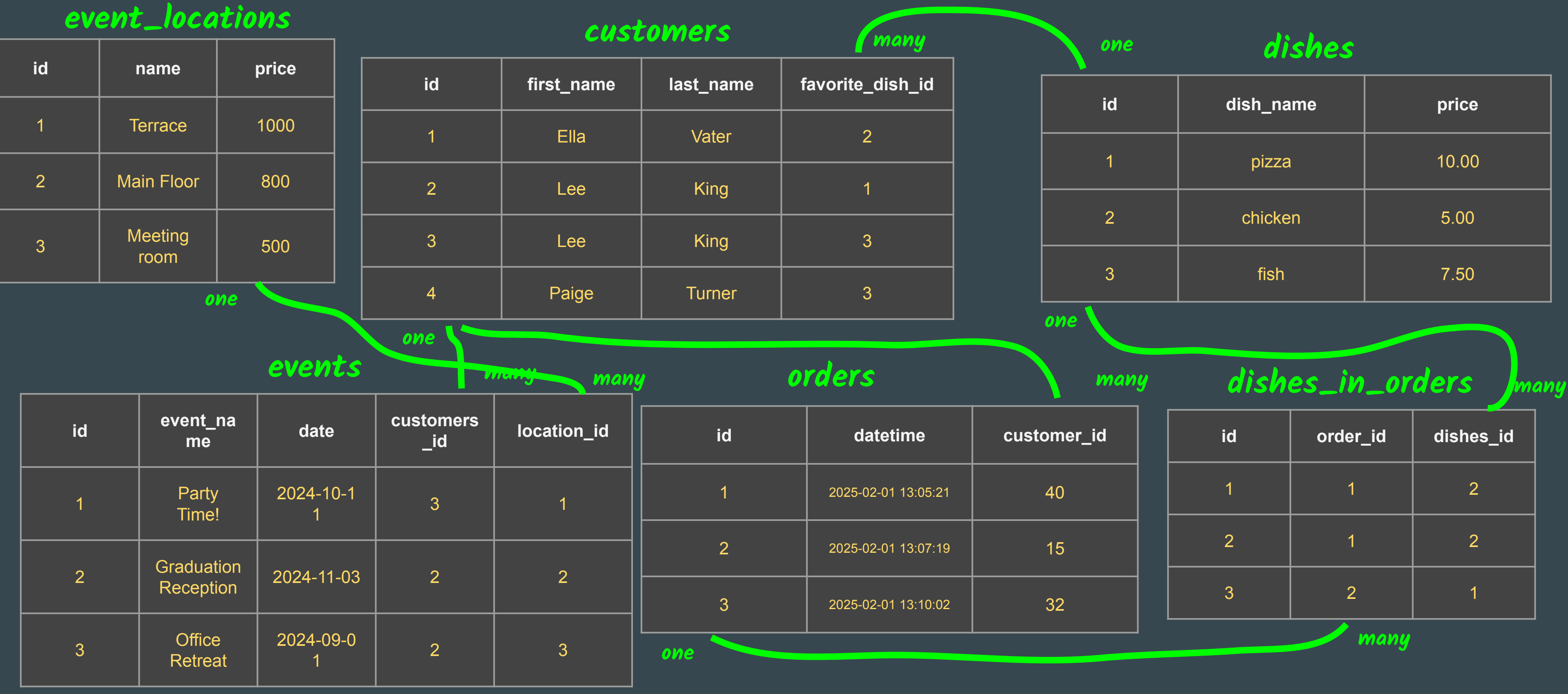
Third Normal Form (3NF) “Smells”

- Attributes that show the same information, but in different form (e.g., discounts)

More normal forms?!

- Third normal form (3NF) is usually where the schema is considered “good enough”
- 4NF - 6NF are out of scope for this workshop, but feel free to look them up.

Updated Schema in Entity-Relation (ER) Diagram



event_locations

id	name	price
1	Terrace	1000
2	Main Floor	800
3	Meeting room	500

customers

id	first_name	last_name	favorite_dish_id
1	Ella	Vater	2
2	Lee	King	1
3	Lee	King	3
4	Paige	Turner	3

dishes

id	dish_name	price
1	pizza	10.00
2	chicken	5.00
3	fish	7.50

events

id	event_name	date	customers_id	location_id
1	Party Time!	2024-10-11	3	1
2	Graduation Reception	2024-11-03	2	2
3	Office Retreat	2024-09-01	2	3

orders

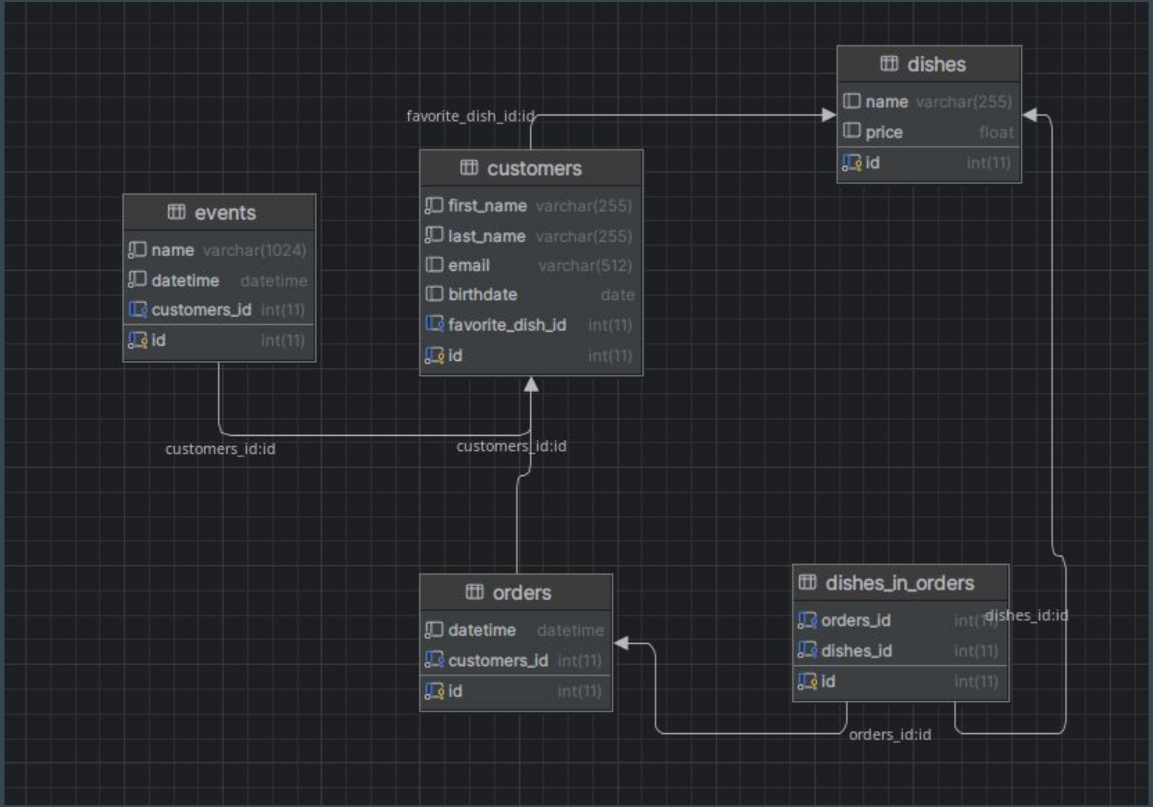
id	datetime	customer_id
1	2025-02-01 13:05:21	40
2	2025-02-01 13:07:19	15
3	2025-02-01 13:10:02	32

dishes_in_orders

id	order_id	dishes_id
1	1	2
2	1	2
3	2	1

Let’s explore some Entity Relation diagram styles

Updated Schema in Entity-Relation (ER) Diagram



events	
name	varchar(1024)
datetime	datetime
customers_id	int(11)
id	int(11)

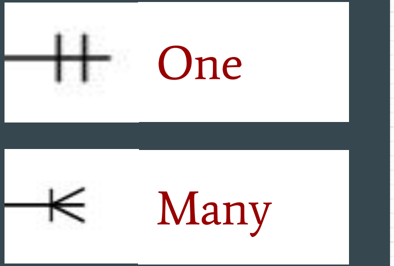
customers	
first_name	varchar(255)
last_name	varchar(255)
email	varchar(512)
birthdate	date
favorite_dish_id	int(11)
id	int(11)

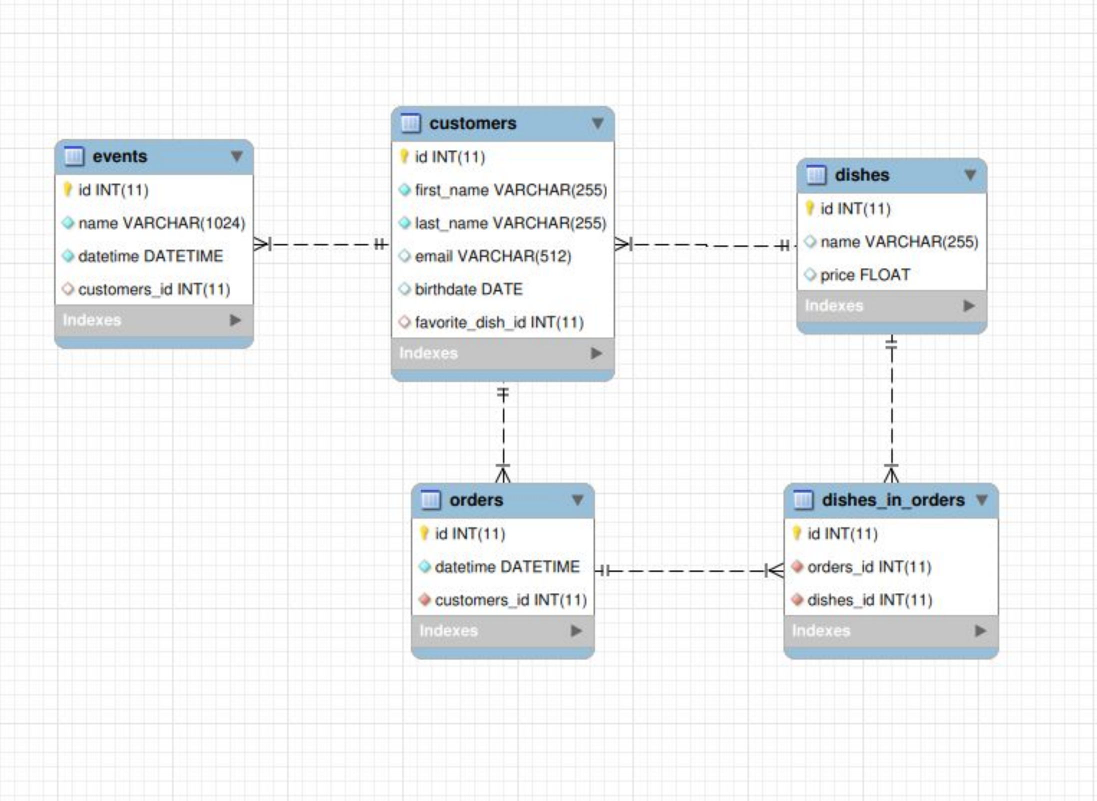
dishes	
name	varchar(255)
price	float
id	int(11)

orders	
datetime	datetime
customers_id	int(11)
id	int(11)

dishes_in_orders	
orders_id	int(11)
dishes_id	int(11)
id	int(11)

Updated Schema in Entity-Relation (ER) Diagram





events
- id INT(11)
- name VARCHAR(1024)
- datetime DATETIME
- customers_id INT(11)
- Indexes

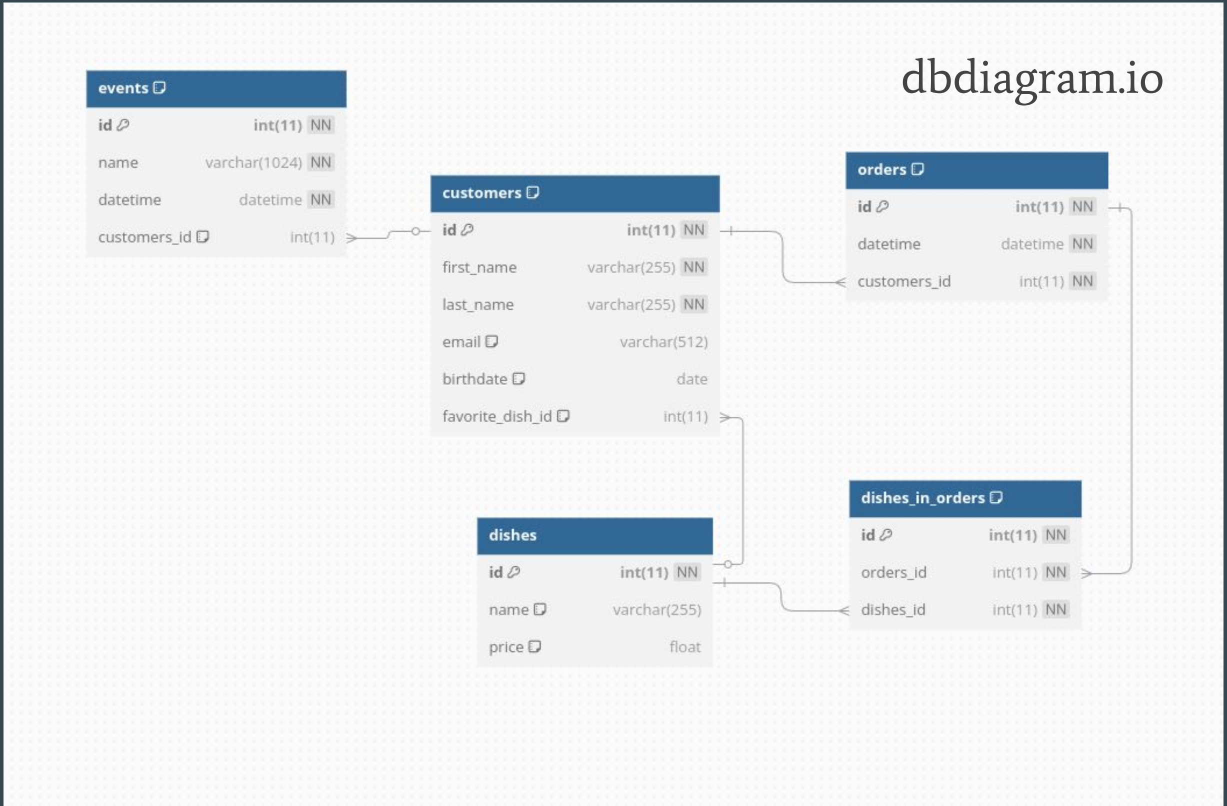
customers
- id INT(11)
- first_name VARCHAR(255)
- last_name VARCHAR(255)
- email VARCHAR(512)
- birthdate DATE
- favorite_dish_id INT(11)
- Indexes

dishes
- id INT(11)
- name VARCHAR(255)
- price FLOAT
- Indexes

orders
- id INT(11)
- datetime DATETIME
- customers_id INT(11)
- Indexes

dishes_in_orders
- id INT(11)
- orders_id INT(11)
- dishes_id INT(11)
- Indexes

Updated Schema in Entity-Relation (ER) Diagram



dbdiagram.io

events		
id	int(11)	NN
name	varchar(1024)	NN
datetime	datetime	NN
customers_id	int(11)	

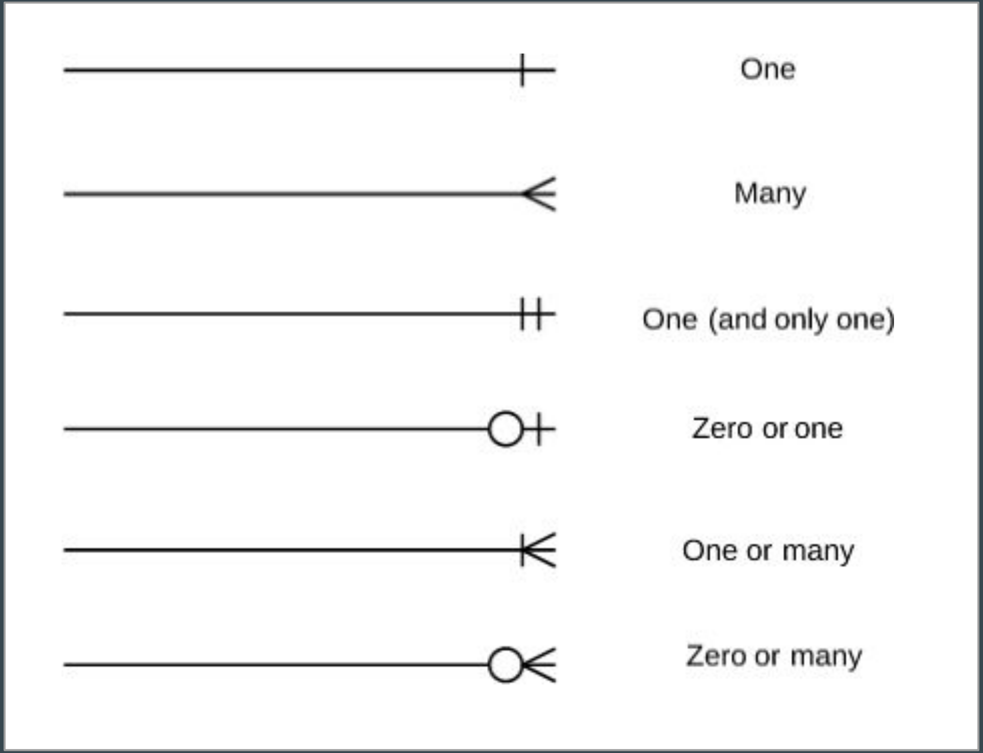
customers		
id	int(11)	NN
first_name	varchar(255)	NN
last_name	varchar(255)	NN
email	varchar(512)	
birthdate	date	
favorite_dish_id	int(11)	

orders		
id	int(11)	NN
datetime	datetime	NN
customers_id	int(11)	NN

dishes		
id	int(11)	NN
name	varchar(255)	
price	float	

dishes_in_orders		
id	int(11)	NN
orders_id	int(11)	NN
dishes_id	int(11)	NN

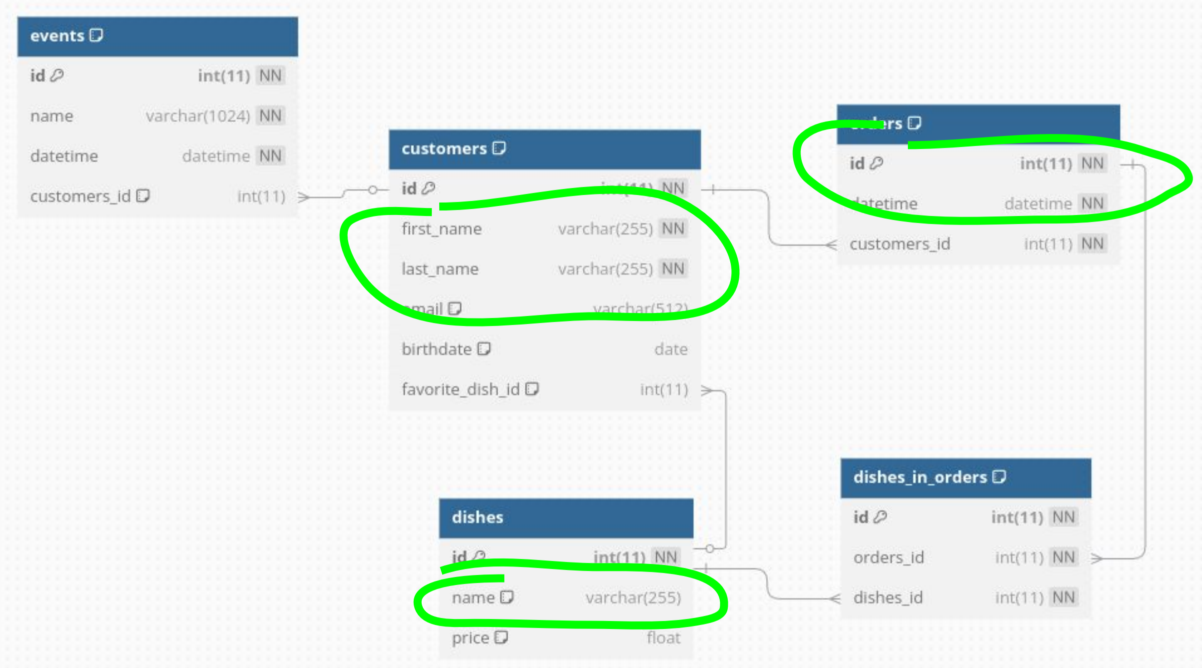
Standard Entity Relation (ER) Diagram Syntax



Let’s write some SQL (again)!

Multi-table JOIN query

- "I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025"



events		
id	int(11)	NN
name	varchar(1024)	NN
datetime	datetime	NN
customers_id	int(11)	

customers		
id	int(11)	NN
first_name	varchar(255)	NN
last_name	varchar(255)	NN
email	varchar(512)	
birthdate	date	
favorite_dish_id	int(11)	

orders		
id	int(11)	NN
datetime	datetime	NN
customers_id	int(11)	NN

dishes		
id	int(11)	NN
name	varchar(255)	
price	float	

dishes_in_orders		
id	int(11)	NN
orders_id	int(11)	NN
dishes_id	int(11)	NN

Multi-table JOIN query

- "I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025"

```
SELECT id, first_name, last_name

FROM customers;
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

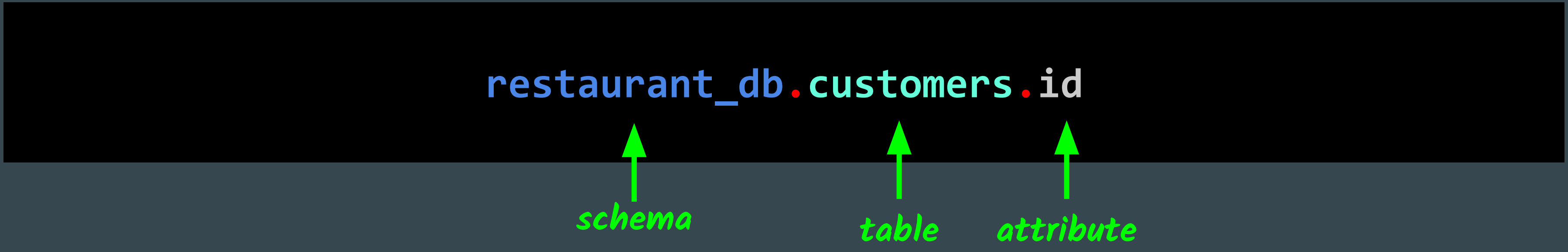
```
SELECT id, first_name, last_name

FROM customers JOIN orders ON customers.id = orders.customers_id
```

```
Column 'id' in field list is ambiguous
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT customers.first_name, customers.last_name, orders.id

FROM customers JOIN orders ON customers.id = orders.customers_id
```

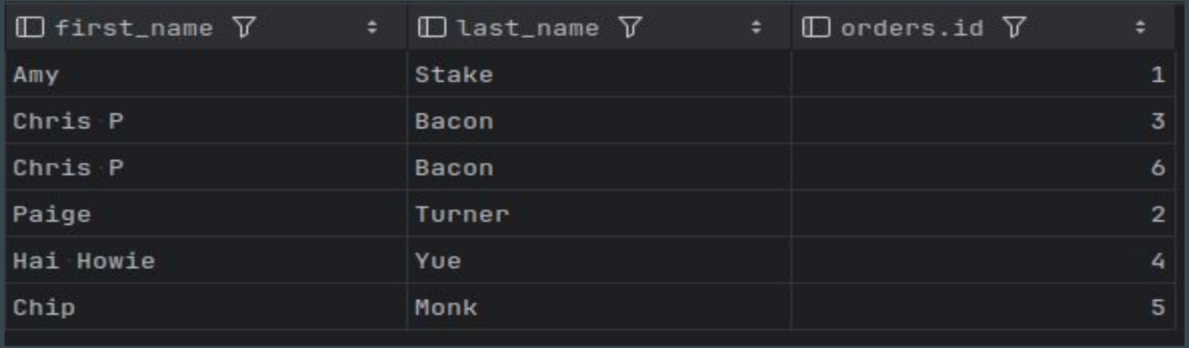


```
restaurant_db.customers.id
```

Multi-table JOIN query

- "I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025"

```sql
SELECT customers.first_name, customers.last_name, orders.id

FROM customers JOIN orders ON customers.id = orders.customers_id
```



first_name	last_name	orders.id
Amy	Stake	1
Chris P	Bacon	3
Chris P	Bacon	6
Paige	Turner	2
Hai Howie	Yue	4
Chip	Monk	5

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT customers.first_name, customers.last_name, orders.id
FROM customers JOIN orders ON customers.id = orders.customers_id
```

customers

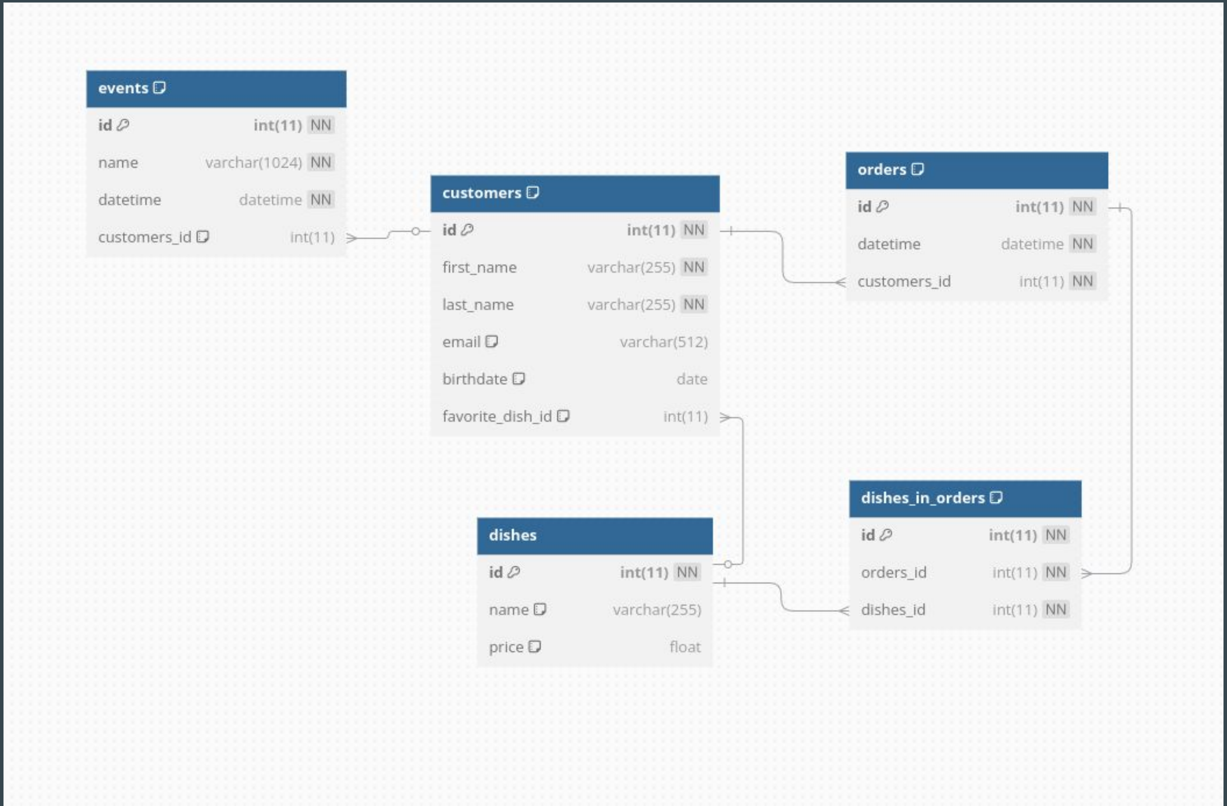
id	first_name	last_name
1	Ella	Vater
2	Lee	King

1:N

orders

id	datetime	customers_id
1	2025-02-01	40
2	2025-02-01	15

Updated Schema in Entity-Relation (ER) Diagram



events		
id	int(11)	NN
name	varchar(1024)	NN
datetime	datetime	NN
customers_id	int(11)	

customers		
id	int(11)	NN
first_name	varchar(255)	NN
last_name	varchar(255)	NN
email	varchar(512)	
birthdate	date	
favorite_dish_id	int(11)	

orders		
id	int(11)	NN
datetime	datetime	NN
customers_id	int(11)	NN

dishes		
id	int(11)	NN
name	varchar(255)	
price	float	

dishes_in_orders		
id	int(11)	NN
orders_id	int(11)	NN
dishes_id	int(11)	NN

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT customers.first_name, customers.last_name, orders.id, dishes_in_orders.dishes_id

FROM customers

    JOIN orders ON customers.id = orders.customers_id

    JOIN dishes_in_orders ON orders.id = dishes_in_orders.orders_id
```

customers

id	first_name	last_name
1	Ella	Vater
2	Lee	King

1:N

orders

id	datetime	customers_id
1	2025-02-01	40
2	2025-02-01	15

1:N

dishes_in_orders

id	order_id	dish_id
1	1	2
2	1	3

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```sql
SELECT customers.first_name, customers.last_name, orders.id, dishes_in_orders.dishes_id, dishes.name

FROM customers

    JOIN orders ON customers.id = orders.customers_id

    JOIN dishes_in_orders ON orders.id = dishes_in_orders.orders_id

    JOIN dishes ON dishes_in_orders.dishes_id = dishes.id
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT c.first_name, c.last_name, o.id, do.dishes_id, d.name

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT c.first_name, c.last_name, o.id AS order_id, d.name

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```sql
SELECT o.id AS order_id, o.datetime, c.first_name, c.last_name, d.name AS dish_name

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id

WHERE o.datetime >= '2025-01-01 00:00:00'
```

Multi-table JOIN query

- “I want to see customer names, order IDs, and dishes that were ordered since January 1, 2025”

```
SELECT o.id AS order_id, o.datetime, c.first_name, c.last_name, d.name
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
WHERE (o.datetime >= '2025-01-01 00:00:00') AND (c.last_name = 'Bacon' OR etc…)
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for Order #2”

```
SELECT c.first_name, c.last_name, d.name, d.price

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id

WHERE o.id = 2
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for Order #2”

```
SELECT c.first_name, c.last_name, d.name, d.price, SUM(d.price) AS `total_price`

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id

WHERE o.id = 2
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for Order #2”

```
SELECT c.first_name, c.last_name, d.name, d.price, SUM(d.price) AS `total_price`

FROM customers c

    JOIN orders o ON c.id = o.customers_id

    JOIN dishes_in_orders do ON o.id = do.orders_id

    JOIN dishes d ON do.dishes_id = d.id

WHERE o.id = 2
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for Order #2”

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
WHERE o.id = 2
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for all orders”

Aggregate values multi-table JOIN query

- “I want to see the full bill for all orders”

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    c.email AS `customer_email`,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
~~WHERE o.id = 2~~
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for all orders”

```sql
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
GROUP BY o.id
```

Aggregate values multi-table JOIN query

- “I want to see the full bill for all orders”

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    c.email AS `customer_email`,
    o.id AS `order_id`,
    o.datetime AS `order_datetime`,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
GROUP BY o.id, o.datetime
ORDER BY o.datetime ASC
```

Virtual columns in queries

- “XYZ Restaurant offers a senior discount of 50% off normal menu prices.”
- AARP: Senior = “55”
- Casey: Senior = “65”

dishes

id	dish_name	price	~~senior_price~~
1	pizza	10.00	~~5.00~~
2	chicken	5.00	~~2.50~~
3	fish	7.50	~~3.75~~

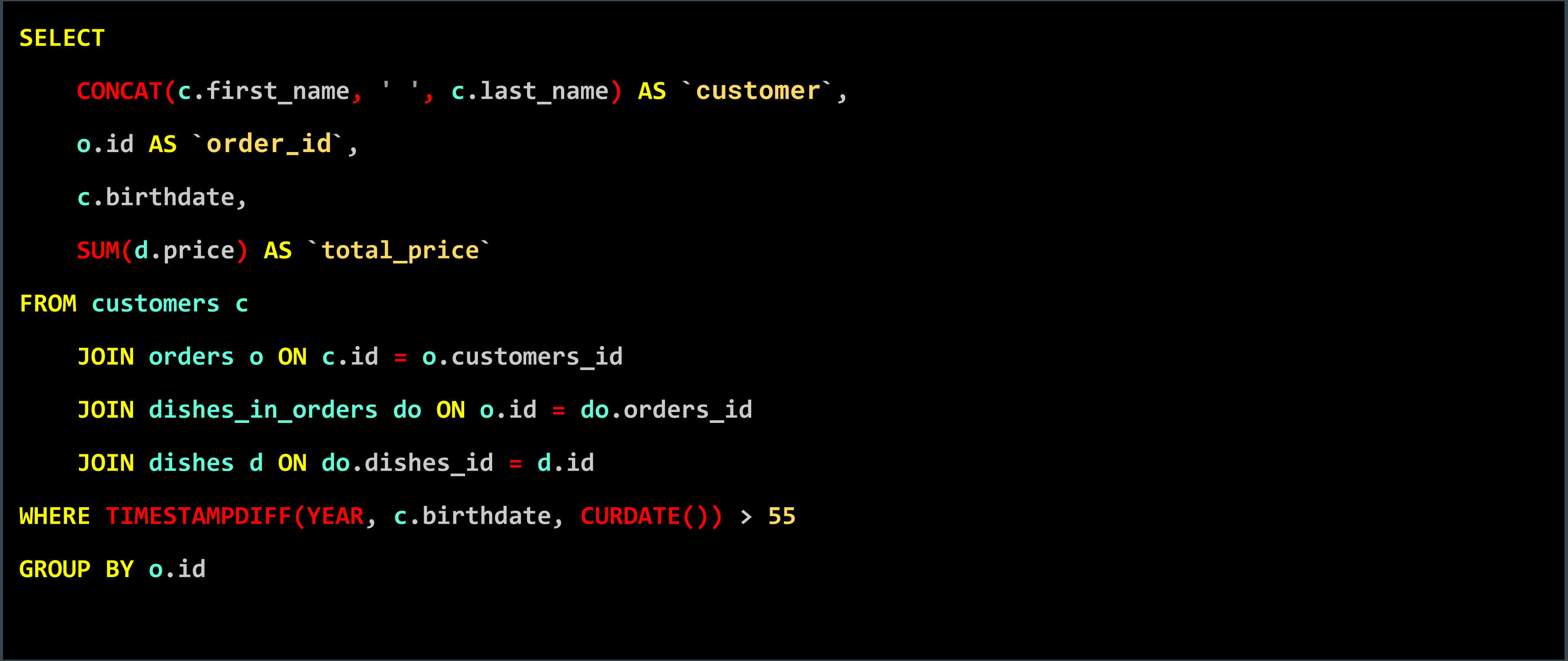
Virtual columns in queries

- “XYZ Restaurant offers a senior discount of 50% off normal menu prices.”

```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    c.birthdate,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
GROUP BY o.id
```

Virtual columns in queries

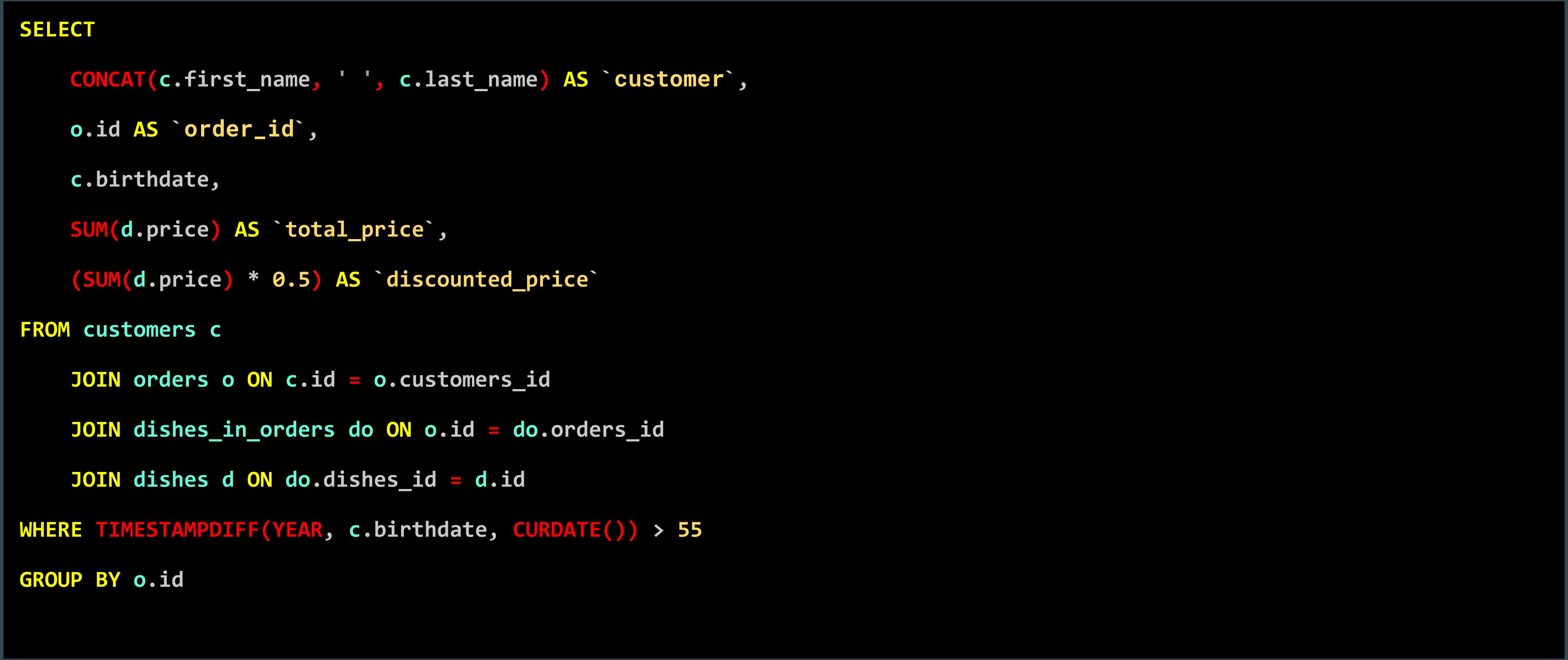
- “XYZ Restaurant offers a senior discount of 50% off normal menu prices.”



```sql
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    c.birthdate,
    SUM(d.price) AS `total_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
WHERE TIMESTAMPDIFF(YEAR, c.birthdate, CURDATE()) > 55
GROUP BY o.id
```

Virtual columns in queries

- “XYZ Restaurant offers a senior discount of 50% off normal menu prices.”



```
SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS `customer`,
    o.id AS `order_id`,
    c.birthdate,
    SUM(d.price) AS `total_price`,
    (SUM(d.price) * 0.5) AS `discounted_price`
FROM customers c
    JOIN orders o ON c.id = o.customers_id
    JOIN dishes_in_orders do ON o.id = do.orders_id
    JOIN dishes d ON do.dishes_id = d.id
WHERE TIMESTAMPDIFF(YEAR, c.birthdate, CURDATE()) > 55
GROUP BY o.id
```

Important things we didn’t cover in-full today…

- Indexes
- One-to-one and recursive relationships
- Denormalization
- All data types (each RDMS differs slightly in these)
- Access control (user accounts and permissions)
- All aggregate functions
- Transactions and stored procedures
- And so much more…

How to continue your RDMS and SQL journey

- LinkedIn Learning video series
  - Programming Foundations: Databases
  - https://bit.ly/3XNMhjQ

- Me:
  - Casey McLaughlin
  - cmclaughlin@fsu.edu

Please fill out this Survey!