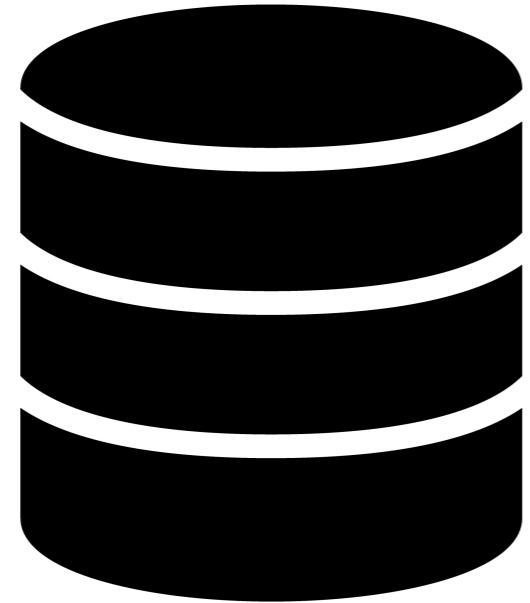


# Welcome

INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor



s tructured  
Q uery  
L anguage

# SQL Server & Transact-SQL

- SQL Server - relational database system developed by Microsoft
- Transact-SQL (T-SQL) - Microsoft's implementation of SQL, with additional functionality
- In this course: Master the fundamentals of T-SQL
- Learn how to write queries

LET'S GET  
STARTED

# Querying 101

- SQL-Server: the *store* containing databases and tables
- Queries: how we *pick* different items, from different aisles, and load up our cart
- **SELECT** : key term for retrieving data



```
SELECT description  
FROM grid;
```

description
Severe Weather Thunderstorms
Severe Weather Thunderstorms
Severe Weather Thunderstorms
Fuel Supply Emergency Coal
Physical Attack Vandalism
Physical Attack Vandalism
Physical Attack Vandalism
Severe Weather Thunderstorms
Severe Weather Thunderstorms
Suspected Physical Attack
Physical Attack Vandalism
...

# Selecting more than one column

**SELECT**

```
artist_id,  
artist_name
```

**FROM**

```
artist;
```

artist_id	artist_name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains
6	Antônio Carlos Jobim
7	Apocalyptica
8	Audioslave
9	BackBeat
10	Billy Cobham

# Query formatting

```
SELECT description, event_year, event_date  
FROM grid;
```

```
SELECT  
    description,  
    event_year,  
    event_date  
FROM  
    grid;
```

# Select TOP ()

```
-- Return 5 rows  
SELECT TOP(5) artist  
FROM artists;
```

```
-- Return top 5% of rows  
SELECT TOP(5) PERCENT artist  
FROM artists;
```

artist
AC/DC
Accept
Aerosmith
Alanis Morissette
Alice in Chains

# Select DISTINCT

```
-- Return all rows in the table  
SELECT nerc_region  
FROM grid;
```

```
+-----+  
| nerc_region |  
+-----+  
| RFC          |  
| RFC          |  
| MRO          |  
| MRO          |  
| ....         |  
+-----+
```

```
-- Return unique rows  
SELECT DISTINCT nerc_region  
FROM grid;
```

```
+-----+  
| nerc_region |  
+-----+  
| NPCC        |  
| NPCC RFC    |  
| RFC          |  
| ERCOT        |  
| ...          |  
+-----+
```

# Select \*

```
-- Return all rows  
SELECT *  
FROM grid;
```

- NOT suitable for large tables

# Aliasing column names with AS

```
SELECT demand_loss_mw AS lost_demand  
FROM grid;
```

```
+-----+  
| lost_demand |  
+-----+  
| 424 |  
| 217 |  
| 494 |  
| 338 |  
| 3900 |  
| 3300 |  
+-----+
```

```
SELECT description AS cause_of_outage  
FROM grid;
```

```
+-----+  
| cause_of_outage |  
+-----+  
| Severe Weather Thunderstorms |  
| Fuel Supply Emergency Coal |  
| Physical Attack Vandalism |  
| Suspected Physical Attack |  
| Electrical System Islanding |  
+-----+
```

# **Let's write some T-SQL!**

**INTRODUCTION TO SQL SERVER**

# Ordering and Filtering

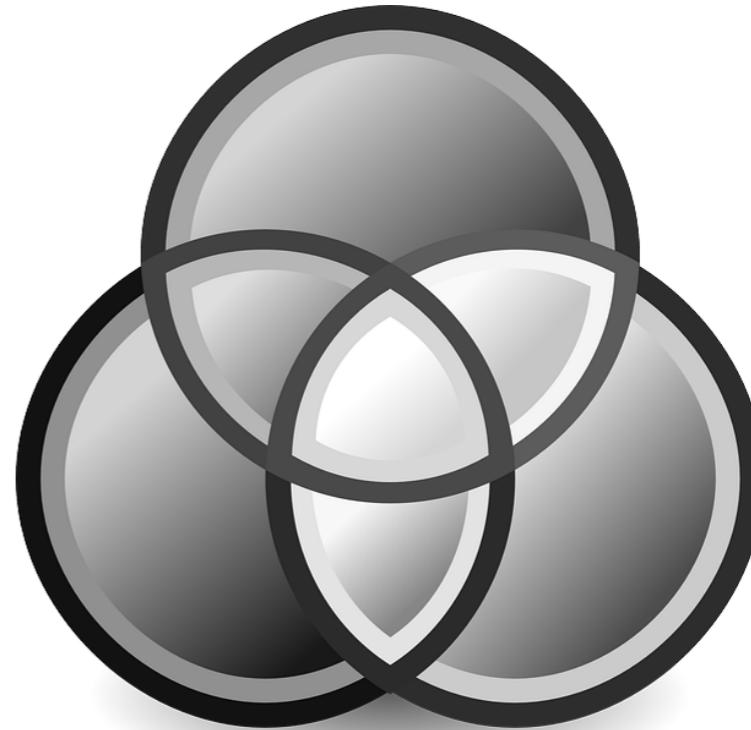
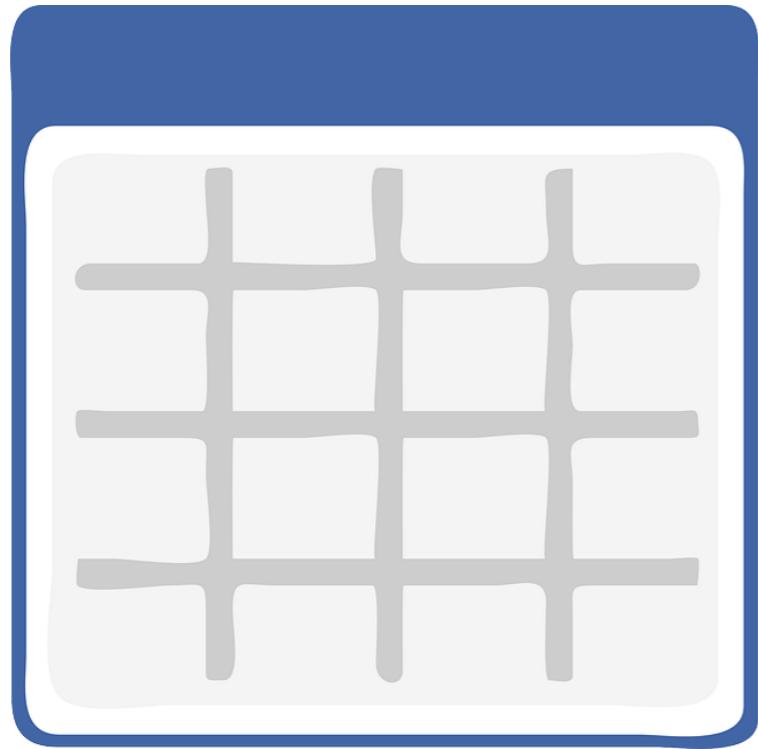
INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# Order! Order!

- Tables comprise of rows and columns
- Queries return *sets*, or *subsets*
- Sets have no inherent order
- If order is important, use `ORDER BY`



```
SELECT TOP (10) prod_id, year_intro  
FROM products  
-- Order in ascending order  
ORDER BY year_intro, product_id;
```

product_id	year_intro
36	1981
37	1982
38	1983
39	1984
40	1984
41	1984
52	1985
43	1986
44	1987
54	1987

```
SELECT TOP (10) product_id, year_intro  
FROM products  
-- Order year_intro in descending order  
ORDER BY year_intro DESC, product_id;
```

product_id	year_intro
158	2015
173	2015
170	2014
171	2014
172	2014
144	2013
146	2013
147	2013
148	2013
149	2013

```
SELECT
```

```
TOP (10) channels,  
year_intro  
FROM products  
-- Order in different directions  
ORDER BY  
    year_intro DESC,  
    channels;
```

channels	year_intro
35	2015
74	2015
29	2014
45	2014
48	2014
12	2013
13	2013
14	2013
22	2013
24	2013

```
SELECT
```

```
TOP (10) channels,  
year_intro  
FROM products  
-- Both columns in descending order  
ORDER BY  
    year_intro DESC,  
    channels DESC;
```

channels	year_intro
74	2015
35	2015
48	2014
45	2014
29	2014
837	2013
642	2013
561	2013
491	2013
198	2013

```
SELECT city_id, name_alias  
FROM invoice  
-- Ordering text (Ascending order)  
ORDER BY name_alias;
```

city_id	name_alias
48	Amsterdam
59	Bangalore
36	Berlin
38	Berlin
42	Bordeaux
23	Boston
13	Brasília
8	Brussels
45	Budapest
56	Buenos Aires

```
SELECT city_id, name_alias  
FROM invoice  
-- Ordering text (Descending order)  
ORDER BY name_alias DESC;
```

city_id	name_alias
33	Yellowknife
32	Winnipeg
49	Warsaw
7	Vienne
15	Vancouver
27	Tucson
29	Toronto
2	Stuttgart
51	Stockholm
55	Sydney

What if we only wanted to return rows that met certain criteria?

```
SELECT customer_id, total  
FROM invoice  
WHERE total > 15;
```

First 3 customers with invoice value > 15

customer_id	total
57	17.91
7	18.86
45	21.86

```
-- Rows with points greater than 10
```

```
WHERE points > 10
```

```
-- Rows with points less than 10
```

```
WHERE points < 10
```

```
-- Rows with points greater than or equal to 10
```

```
WHERE points >= 10
```

```
-- Rows with points less than or equal to 20
```

```
WHERE points <= 20
```

```
-- Character data type
```

```
WHERE country = 'Spain'
```

```
-- Date data type
```

```
WHERE event_date = '2012-01-02'
```

```
SELECT customer_id, total  
FROM invoice  
-- Testing for non-equality  
WHERE total <> 10;
```

customerid	total
2	1.98
4	3.96
8	5.94
14	8.91
23	13.86
37	0.99

# Between

```
SELECT customer_id, total  
FROM invoice  
WHERE total BETWEEN 20 AND 30;
```

customerid	total
45	21.86
46	21.86
26	23.86
6	25.86

```
SELECT customer_id, total  
FROM invoice  
WHERE total NOT BETWEEN 20 AND 30;
```

customerid	total
2	1.98
4	3.96
8	5.94
14	8.91

# What is NULL?

- NULL indicates there is no value for that record
- NULLs help highlight gaps in our data

**SELECT**

```
TOP (6) total,  
      billing_state  
FROM invoice  
WHERE billing_state IS NULL;
```

total	billing_state
1.98	NULL
3.96	NULL
5.94	NULL
0.99	NULL
1.98	NULL
1.98	NULL

**SELECT**

```
TOP (6) total,  
      billing_state  
FROM invoice  
WHERE billing_state IS NOT NULL;
```

total	billing_state
8.91	AB
13.96	MA
5.94	Dublin
0.99	CA
1.98	WA
1.98	CA

# **Let's sort it!**

**INTRODUCTION TO SQL SERVER**

# WHERE the wild things are

INTRODUCTION TO SQL SERVER

SQL

John MacKintosh  
Instructor



```
SELECT song, artist  
FROM songlist  
WHERE  
    artist = 'AC/DC';
```

song	artist
Baby, Please Don't Go	AC/DC
Back In Black	AC/DC
Big Gun	AC/DC
CAN'T STOP ROCK'N'ROLL	AC/DC
Girls Got Rhythm	AC/DC
Hard As A Rock	AC/DC
Have a Drink On Me	AC/DC
Hells Bells	AC/DC

```
SELECT song, artist  
FROM songlist  
WHERE  
    artist = 'AC/DC'  
    AND release_year < 1980;
```

song	artist
Dirty Deeds Done Dirt Cheap	AC/DC
Highway To Hell	AC/DC
It's A Long Way To The Top	AC/DC
Let There Be Rock	AC/DC
Night Prowler	AC/DC
T.N.T.	AC/DC
Touch Too Much	AC/DC
Whole Lotta Rosie	AC/DC

# AND again

- Returns 3 rows:

```
SELECT *
FROM songlist
WHERE
    release_year = 1994
    AND artist = 'Green Day';
```

- Returns 1 row:

```
SELECT *
FROM songlist
WHERE
    release_year = 1994
    AND artist = 'Green Day'
    AND song = 'Basket Case';
```

**SELECT**

```
song,  
artist,  
release_year  
FROM songlist  
WHERE release_year = 1994;
```

song	artist	release_year
Black Hole Sun	Soundgarden	1994
Fell On Black Days	Soundgarden	1994
Spoonman	Soundgarden	1994
Big Empty	Stone Temple Pilots	1994
Interstate Love Song	Stone Temple Pilots	1994
Vaseline	Stone Temple Pilots	1994

```
SELECT
```

```
    song,  
    artist,  
    release_year
```

```
FROM songlist
```

```
WHERE
```

```
    release_year = 1994
```

```
    OR release_year > 2000;
```

song	artist	release_year
Doom And Gloom	Rolling Stones	2012
Remedy	Seether	2005
45	Shinedown	2003
Black Hole Sun	Soundgarden	1994
Fell On Black Days	Soundgarden	1994
Spoonman	Soundgarden	1994
It's Been Awhile	Staind	2001
Big Empty	Stone Temple Pilots	1994
Interstate Love Song	Stone Temple Pilots	1994
Vaseline	Stone Temple Pilots	1994

```
SELECT song  
FROM songlist  
WHERE  
    artist = 'Green Day'  
AND release_year = 1994;
```

```
+-----+  
| song |  
+-----+  
| Basket Case |  
| Longview |  
| When I Come Around |  
+-----+
```

```
SELECT song  
FROM songlist  
WHERE  
    artist = 'Green Day'  
AND release_year > 2000;
```

```
+-----+  
| song |  
+-----+  
| Boulevard Of Broken Dreams |  
| Holiday (Live) |  
| Holiday |  
+-----+
```

```
SELECT song
FROM songlist
WHERE
    artist = 'Green Day'
    AND release_year = 1994
    OR release_year > 2000;
```

song
Doom And Gloom
Remedy
45
It's Been Awhile
Goodbye Daughters of the Revolution
Gold On The Ceiling
Lonely Boy
Seven Nation Army
Get Together
Vertigo
When I'm Gone
...
...

# What went wrong?

```
SELECT *
FROM songlist
WHERE
    artist = 'Green Day'
    AND release_year = 1994
    OR release_year > 2000;
```

```
SELECT *
FROM songlist
WHERE
    artist = 'Green Day'
    AND release_year = 1994;
```

OR

```
SELECT *
FROM songlist
WHERE
    release_year > 2000;
```

```
SELECT song
FROM songlist
WHERE
    artist = 'Green Day'
    AND (
        release_year = 1994
        OR release_year > 2000
    );
```

Another way of writing the query:

```
SELECT song
FROM songlist
WHERE
(
    artist = 'Green Day'
    AND release_year = 1994
)
OR (
    artist = 'Green Day'
    AND release_year > 2000
);
```

song
Basket Case
Boulevard Of Broken Dreams
Holiday (Live)
Holiday / Boulevard of Broken Dreams
Longview
When I Come Around

```
SELECT song, artist  
FROM songlist  
WHERE  
    artist IN ('Van Halen', 'ZZ Top')  
ORDER BY song;
```

song	artist
(Oh) Pretty Woman	Van Halen
1984/jump	Van Halen
A Fool for Your Stockings	ZZ Top
Ain't Talkin' 'bout Love	Van Halen
And the Cradle Will Rock...	Van Halen
Arrested For Driving While Blind	ZZ Top
Atomic Punk	Van Halen

```
SELECT song, release_year  
FROM songlist  
WHERE  
    release_year IN (1985, 1991, 1992);
```

song	release_year
Addicted to Love	1985
Don't You	1985
Come As You Are	1991
Money for Nothing	1985
Walk of Life	1985
Man On the Moon	1992
Breaking the Girl	1992
You Belong to the City	1985
Enter Sandman	1991
In Bloom	1991

```
SELECT song  
FROM songlist  
WHERE song LIKE 'a%';
```

```
+-----+  
| song |  
+-----+  
| Addicted to Love |  
| Ain't Too Proud to Beg |  
+-----+
```

```
SELECT artist  
FROM songlist  
WHERE artist LIKE 'f%';
```

```
+-----+  
| artist |  
+-----+  
| Faces |  
| Faith No More |  
+-----+
```

# **Let's practice!**

## **INTRODUCTION TO SQL SERVER**

# Aggregating Data

## INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# SUM - single column

Calculate the total amount of a column value with `SUM()`

```
SELECT
```

```
    SUM(affected_customers) AS total_affected
```

```
FROM grid;
```

total_affected
70143996

# SUM - two or more columns

```
SELECT
```

```
    SUM(affected_customers) AS total_affected  
FROM grid;
```

```
SELECT
```

```
    SUM (affected_customers) AS total_affected,  
    SUM (demand_loss_mw) AS total_loss  
FROM grid;
```

total_affected	total_loss
70143996	177888

# The wrong way...

```
SELECT
```

```
    SUM (affected_customers) AS total_affected,  
    (demand_loss_mw) AS total_loss  
FROM grid;
```

```
Msg 8120, Level 16, State 1, Line 6
```

```
Column 'grid_demand_loss_mw' is invalid in the select list because  
it is not contained in either an aggregate function or the GROUP BY clause.
```

# Use aliases

```
SELECT
```

```
    SUM (affected_customers),  
    SUM (demand_loss_mw)  
FROM grid;
```

```
+-----+-----+  
| (No column name) | (No column name) |  
|-----+-----+  
| 70143996        | 177888       |  
+-----+-----+
```

```
SELECT
```

```
    SUM (affected_customers) AS total_affected,  
    SUM (demand_loss_mw) AS total_loss  
FROM grid;
```

```
+-----+-----+  
| total_affected | total_loss |  
|-----+-----|  
| 70143996      | 177888     |  
+-----+-----+
```

# COUNT

**SELECT**

```
COUNT(affected_customers) AS count_affected  
FROM grid;
```

count_affected
807

# COUNT Distinct

```
SELECT
```

```
    COUNT(DISTINCT affected_customers) AS unique_count_affected  
FROM grid;
```

```
+-----+  
| unique_count_affected |  
+-----+  
| 280 |  
+-----+
```

# MIN

**SELECT**

```
MIN(affected_customers) AS min_affected_customers  
FROM grid;
```

```
+-----+  
| min_affected_customers |  
+-----+  
| 0 |  
+-----+
```

**SELECT**

```
MIN(affected_customers) AS min_affected_customers  
FROM grid  
WHERE affected_customers > 0;
```

```
+-----+  
| min_affected_customers |  
+-----+  
| 1 |  
+-----+
```

# MAX

**SELECT**

```
MAX(affected_customers) AS max_affected_customers  
FROM grid;
```

max_affected_customers
4645572

# Average

```
SELECT
```

```
    AVG(affected_customers) AS avg_affected_customers  
FROM grid;
```

```
+-----+  
| avg_affected_customers |  
+-----+  
| 86919 |  
+-----+
```

# **Let's practice!**

## **INTRODUCTION TO SQL SERVER**

# Strings

## INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

**SELECT**

```
description,  
LEN(description) AS description_length  
FROM grid;
```

description	description_length
Severe Weather Thunderstorms	29
Severe Weather Thunderstorms	29
Severe Weather Thunderstorms	29
Fuel Supply Emergency Coal	27
Physical Attack Vandalism	26

```
SELECT
```

```
    description,  
    LEFT(description, 20) AS first_20_left  
FROM grid;
```

description	first_20_left
Severe Weather Thunderstorms	Severe Weather Thun
Severe Weather Thunderstorms	Severe Weather Thun
Severe Weather Thunderstorms	Severe Weather Thun
Fuel Supply Emergency Coal	Fuel Supply Emergenc
Physical Attack Vandalism	Physical Attack Van

```
SELECT
```

```
    description,  
    RIGHT(description, 20) AS last_20  
FROM grid;
```

description	last_20
Severe Weather Thunderstorms	ather Thunderstorms
Severe Weather Thunderstorms	ather Thunderstorms
Severe Weather Thunderstorms	ather Thunderstorms
Fuel Supply Emergency Coal	pply Emergency Coal
Physical Attack Vandalism	al Attack Vandalism

**SELECT**

```
CHARINDEX ('_', url) AS char_location,  
url  
FROM courses;
```

char_location	url
34	datacamp.com/courses/introduction_
34	datacamp.com/courses/intermediate_
29	datacamp.com/courses/writing_
29	datacamp.com/courses/joining_
27	datacamp.com/courses/intro_

```
SELECT
```

```
    SUBSTRING(url, 12, 12) AS target_section,
```

```
    url
```

```
FROM courses;
```

target_section	url
datacamp.com	https://www.datacamp.com/courses

# REPLACE

**SELECT**

```
TOP(5) REPLACE(url, '_', '-') AS replace_with_hyphen  
FROM courses;
```

```
+-----+  
| replace_with_hyphen |  
+-----|  
| datacamp.com/courses/introduction- |  
| datacamp.com/courses/intermediate- |  
| datacamp.com/courses/writing- |  
| datacamp.com/courses/joining- |  
| datacamp.com/courses/intro- |  
+-----+
```

# **Let's practice!**

## **INTRODUCTION TO SQL SERVER**

# Grouping and Having

INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# A simple SELECT

```
SELECT
```

```
    SUM(demand_loss_mw) AS lost_demand
```

```
FROM grid;
```

```
+-----+
| lost_demand |
+-----+
| 177888      |
+-----+
```

# Grouping error

Can we break this down by adding an additional column?

```
SELECT  
    SUM(demand_loss_mw) AS lost_demand,  
    description  
FROM grid;
```

```
Msg 8120, Level 16, State 1, Line 1  
Column 'grid.description' is invalid in the select list because it is not contained in  
either an aggregate function or the GROUP BY clause.
```

**SELECT**

```
SUM(demand_loss_mw) AS lost_demand,  
description  
FROM grid  
GROUP BY description;
```

lost_demand	description
NULL	Actual Physical Attack
NULL	Cold Weather Event
NULL	Cyber Event with Potential to Cause Impact
40	Distribution Interruption
2	Distribution System Interruption
NULL	Earthquake
NULL	Electrical Fault at Generator
338	Electrical System Islanding
24514	Electrical System Separation Islanding
15	Electrical System Separation Islanding Severe Weather

```
SELECT
    SUM(demand_loss_mw) AS lost_demand,
    description
FROM grid
WHERE
    description LIKE '%storm'
    AND demand_loss_mw IS NOT NULL
GROUP BY description;
```

lost_demand	description
996	Ice Storm
420	Load Shed Severe Weather    Lightning Storm
332	Major Storm
3	Severe Weather    Thunderstorm
413	Severe Weather    Wind Storm
4171	Severe Weather    Winter Storm
1352	Winter Storm

# HAVING

- Can use aggregate functions in `SELECT`
- Filter data using `WHERE`
- Split data into groups using `GROUP BY`
- What if we want to sum values based on groups?
- ... and then filter on those sums?

```
SELECT
```

```
    SUM(demand_loss_mw) AS lost_demand,  
    description
```

```
FROM grid
```

```
WHERE
```

```
    description LIKE '%storm'
```

```
    AND demand_loss_mw IS NOT NULL
```

```
GROUP BY description;
```

lost_demand	description
996	Ice Storm
420	Load Shed Severe Weather
332	Lightning Storm
3	Major Storm
413	Severe Weather Thunderstorm
4171	Wind Storm
1352	Winter Storm

```
SELECT
    SUM(demand_loss_mw) AS lost_demand,
    description
FROM grid
WHERE
    description LIKE '%storm'
    AND demand_loss_mw IS NOT NULL
GROUP BY description
HAVING SUM(demand_loss_mw) > 1000;
```

lost_demand	description
4171	Severe Weather Winter Storm
1352	Winter Storm

# Summary

- GROUP BY splits the data up into combinations of one or more values
- WHERE filters on row values
- HAVING appears after the GROUP BY clause and filters on groups or aggregates

**Let's put our skills to  
the test!**

**INTRODUCTION TO SQL SERVER**

# Joining tables

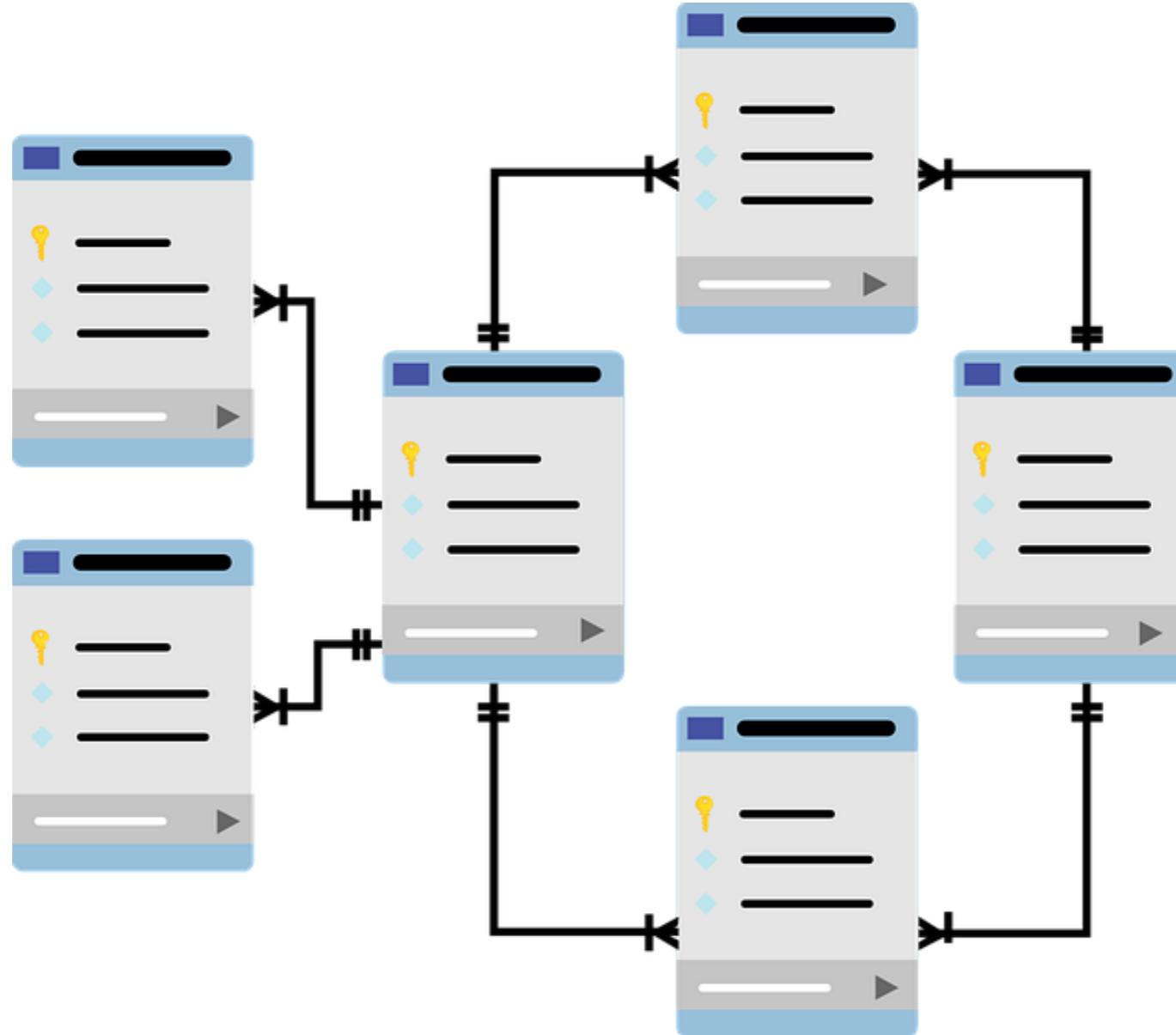
INTRODUCTION TO SQL SERVER



**John MacKintosh**

Instructor

# Relational Databases



# Primary Keys

- Primary keys: Uniquely identify each row in a table

artist_id	name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains

- Primary key: `artist_id`

album_id	title	artist_id
1	For Those About To Rock	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

- Primary key: `album_id`
- What about `artist_id` ?

# Foreign keys

- artist table

artist_id	name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains

- album table

album_id	title	artist_id
1	For Those About To Rock	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

- artist\_id : Foreign key to artist

# Joining album and artist

- `artist` table

artist_id	name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains

- `album` table

album_id	title	artist_id
1	For Those About To Rock	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

- AC/DC has `artist_id` = 1

- Rows 1 and 4 have `artist_id` = 1

# Joining album and artist

album_id	title	artist_id	artist_name
1	For Those About To Rock	1	AC/DC
4	Let There Be Rock	1	AC/DC

- Return album details from `album` table
- Return corresponding artist details from `artist` table
- Joined using `artist_id` column

# INNER JOIN

**SELECT**

```
album_id,  
title,  
album.artist_id,  
name AS artist_name  
FROM album  
INNER JOIN artist ON artist.artist_id = album.artist_id  
WHERE album.artist_id = 1;
```

album_id	title	artist_id	artist_name
1	For Those About To Rock	1	AC/DC
4	Let There Be Rock	1	AC/DC

# INNER JOIN syntax

**SELECT**

```
table_A.columnX,  
table_A.columnY,  
table_B.columnZ
```

**FROM** table\_A

**INNER JOIN** table\_B **ON** table\_A.foreign\_key = table\_B.primary\_key;

**SELECT**

```
album_id,  
title,  
album.artist_id,  
name AS artist_name  
FROM album  
INNER JOIN artist on artist.artist_id = album.artist_id;
```

album_id	title	artist_id	artist_name
1	For Those About To Rock	1	AC/DC
4	Let There Be Rock	1	AC/DC
2	Balls To The Wall	2	Accept
3	Restless and Wild	2	Accept

- Returns **all** combinations of **all** matches between `album` and `artist`

# Multiple INNER JOINS

**SELECT**

```
table_A.columnX,  
table_A.columnY,  
table_B.columnZ, table_C columnW  
FROM table_A  
INNER JOIN table_B ON table_B.foreign_key = table_A.primary_key  
INNER JOIN table_C ON table_C.foreign_key = table_B.primary_key;
```

# **Let's join some tables!**

**INTRODUCTION TO SQL SERVER**

# Mix n match - LEFT & RIGHT joins

INTRODUCTION TO SQL SERVER

A dark blue circular icon containing the letters "SQL" in white.

**John MacKintosh**  
Instructor

# The rationale for LEFT and RIGHT joins

- Why do we need LEFT and RIGHT joins?
- One table may not have an exact match in another:
  - Customer order history for marketing campaign
  - Product list and returns history
  - Patients admitted but not yet discharged

# The rationale for LEFT and RIGHT joins

- Why do we need LEFT and RIGHT joins?
- One table may not have an exact match in another:
  - Customer order history for marketing campaign
  - Product list and returns history
  - **Patients admitted but not yet discharged**

## Admissions table

Patient_ID	Admitted
1	1
2	1
3	1
4	1
5	1

## Discharges table

Patient_ID	Discharged
1	1
3	1
4	1

INNER JOIN:

Patient_ID	Admitted	Discharged
1	1	1
3	1	1
4	1	1

LEFT JOIN:

Patient_ID	Admitted	Discharged
1	1	1
2	1	NULL
3	1	1
4	1	1
5	1	NULL

# LEFT JOIN SYNTAX

**SELECT**

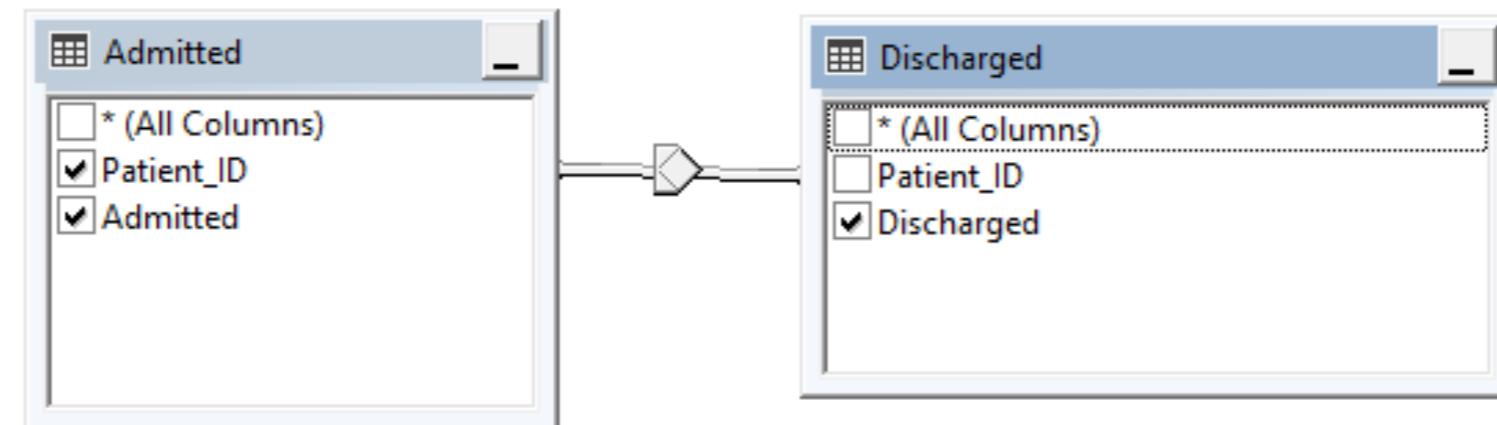
Admitted.Patient\_ID,

Admitted,

Discharged

**FROM** Admitted

**LEFT JOIN** Discharged **ON** Discharged.Patient\_ID = Admitted.Patient\_ID;



**SELECT**

Admitted.Patient\_ID,

Admitted,

Discharged

**FROM** Admitted

**LEFT JOIN** Discharged **ON** Discharged.Patient\_ID = Admitted.Patient\_ID;

Patient_ID	Admitted	Discharged
1	1	1
2	1	NULL
3	1	1
4	1	1
5	1	NULL

# RIGHT JOIN

**SELECT**

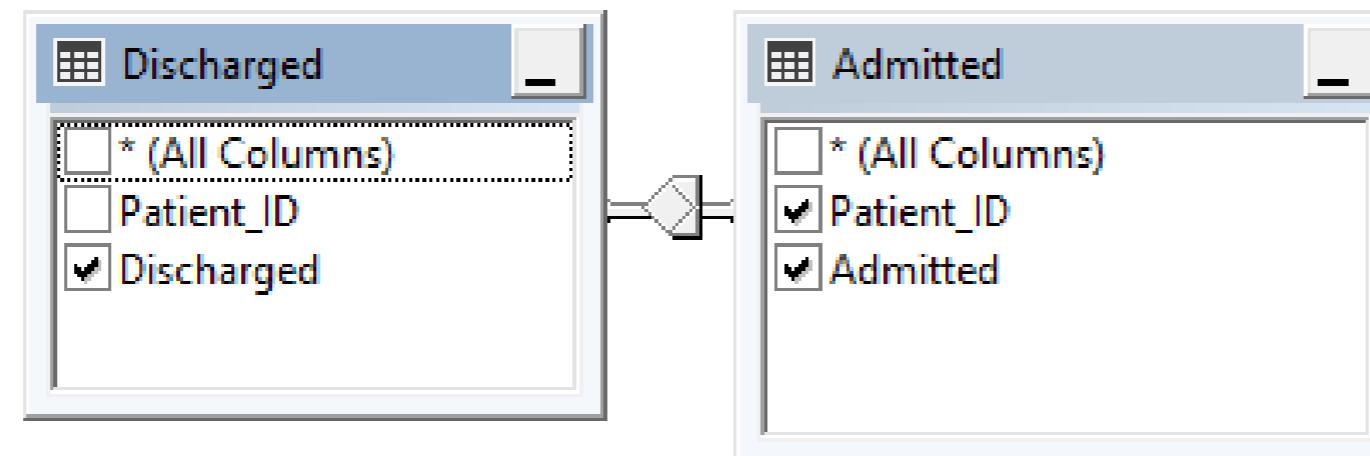
Admitted.Patient\_ID,

Admitted,

Discharged

**FROM** Discharged

**RIGHT JOIN** Admitted **ON** Admitted.Patient\_ID = Discharged.Patient\_ID;



# RIGHT JOIN results

SELECT

```
Admitted.Patient_ID,  
Admitted,  
Discharged  
FROM Discharged  
RIGHT JOIN Admitted ON Admitted.Patient_ID = Discharged.Patient_ID;
```

Patient_ID	Admitted	Discharged
1	1	1
2	1	NULL
3	1	1
4	1	1
5	1	NULL

# Summary

- `INNER JOIN` : Only returns matching rows
- `LEFT JOIN` (or `RIGHT JOIN`): All rows from the main table plus matches from the joining table
- `NULL` : Displayed if no match is found
- `LEFT JOIN` and `RIGHT JOIN` can be interchangeable

## INNER JOIN

LEFT TABLE	RIGHT TABLE
MATCHES RETURNED, NON MATCHES DISCARDED	MATCHES RETURNED, NON MATCHES DISCARDED

## LEFT JOIN

LEFT - MAIN TABLE	RIGHT - JOINING TABLE
ALL ROWS RETURNED	MATCHES RETURNED, NON MATCHES RETURN NULL

## RIGHT JOIN

LEFT - JOINING TABLE	RIGHT - MAIN TABLE
MATCHES RETURNED, NON MATCHES RETURN NULL	ALL ROWS RETURNED

# **Let's Practice!**

**INTRODUCTION TO SQL SERVER**

# UNION & UNION ALL

INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

**SELECT**

```
album_id,  
title,  
artist_id  
FROM album  
WHERE artist_id IN (1, 3)
```

album_id	title	artist_id
1	For Those About To Rock	1
4	Let There Be Rock	1
5	Big Ones	3

**SELECT**

```
album_id,  
title,  
artist_id  
FROM album  
WHERE artist_id IN (1, 4, 5)
```

album_id	title	artist_id
1	For Those About To Rock	1
4	Let There Be Rock	1
6	Jagged Little Pill	4
7	Facelift	5

# Combining results

```
SELECT  
    album_id,  
    title,  
    artist_id  
FROM album  
WHERE artist_id IN (1, 3)  
  
UNION  
SELECT  
    album_id,  
    title,  
    artist_id  
FROM album  
WHERE artist_id IN (1, 4, 5);
```

album_id	title	artist_id
1	For Those About To Rock	1
4	Let There Be Rock	1
5	Big Ones	3
6	Jagged Little Pill	4
7	Facelift	5

- Duplicate rows are excluded

# UNION ALL

**SELECT**

```
album_id,  
title,  
artist_id  
FROM album  
WHERE artist_id IN (1, 3)  
UNION ALL  
SELECT  
album_id,  
title,  
artist_id  
FROM album  
WHERE artist_id IN (1, 4, 5);
```

album_id	title	artist_id
1	For Those About To Rock	1
4	Let There Be Rock	1
5	Big Ones	3
1	For Those About To Rock	1
4	Let There Be Rock	1
6	Jagged Little Pill	4
7	Facelift	5

- Includes duplicate rows

# Creating new column names for final results

```
SELECT
```

```
    album_id AS ALBUM_ID,  
    title AS ALBUM_TITLE,  
    artist_id AS ARTIST_ID  
FROM album  
WHERE artist_id IN(1, 3)
```

```
UNION ALL
```

```
SELECT
```

```
    album_id AS ALBUM_ID,  
    title AS ALBUM_TITLE,  
    artist_id AS ARTIST_ID  
FROM album  
WHERE artist_id IN(1, 4, 5)
```

ALBUM_ID	ALBUM_TITLE	ARTIST_ID
1	For Those About To Rock	1
4	Let There Be Rock	1
5	Big Ones	3
1	For Those About To Rock	1
4	Let There Be Rock	1
6	Jagged Little Pill	4
7	Facelift	5

# Summary

- UNION or UNION ALL : Combines queries from the same table or different tables

If combining data from different tables:

- Select the same number of columns in the same order
- Columns should have the same data types

If source tables have different column names

- Alias the column names

UNION : Discards duplicates (slower to run)

UNION ALL : Includes duplicates (faster to run)

# **Let's practice!**

## **INTRODUCTION TO SQL SERVER**

# You've got the power

INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# CRUD operations

## CREATE

- Databases, Tables or views
- Users, permissions, and security groups

## READ

- Example: `SELECT` statements

## UPDATE

- Amend existing database records

## DELETE

# CREATE

- `CREATE TABLE` unique table name
- (column name, data type, size)

```
CREATE TABLE test_table(  
    test_date date,  
    test_name varchar(20),  
    test_int int  
)
```

# A few considerations when creating a table

- Table and column names
- Type of data each column will store
- Size or amount of data stored in the column

# Data types

Dates:

- date ( YYYY-MM-DD ), datetime ( YYYY-MM-DD hh:mm:ss )
- time

Numeric:

- integer, decimal, float
- bit ( 1 = TRUE , 0 = FALSE . Also accepts NULL values)

Strings:

- char , varchar , nvarchar

# **Let's create some tables!**

**INTRODUCTION TO SQL SERVER**

# Insert, Update, Delete

INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# INSERT

```
INSERT INTO table_name
```

```
INSERT INTO table_name (col1, col2, col3)
```

```
INSERT INTO table_name (col1, col2, col3)
```

```
VALUES
```

```
('value1', 'value2', value3)
```

# INSERT SELECT

```
INSERT INTO table_name (col1, col2, col3)
SELECT
    column1,
    column2,
    column3
FROM other_table
WHERE
    -- conditions apply
```

- Don't use `SELECT *`
- Be specific in case table structure changes

# UPDATE

```
UPDATE table  
SET column = value  
WHERE  
    -- Condition(s);
```

- Don't forget the WHERE clause!

```
UPDATE table  
SET  
    column1 = value1,  
    column2 = value2  
WHERE  
    -- Condition(s);
```

# DELETE

```
DELETE  
FROM table  
WHERE  
    -- Conditions
```

- Test beforehand!

```
TRUNCATE TABLE table_name
```

- Clears the entire table at once



# **Let's INSERT, UPDATE, and DELETE!**

**INTRODUCTION TO SQL SERVER**

# Declare yourself

## INTRODUCTION TO SQL SERVER



**John MacKintosh**  
Instructor

# Variables

```
SELECT *
FROM artist
WHERE name = 'AC/DC';
```

Now change the query for another artist:

```
SELECT *
FROM artist
WHERE name = 'U2';
```

To avoid repetition, create a variable:

```
SELECT *
FROM artist
WHERE name = @my_artist;
```

# DECLARE

```
DECLARE @
```

Integer variable:

```
DECLARE @test_int INT
```

Varchar variable:

```
DECLARE @my_artist VARCHAR(100)
```

# SET

Integer variable:

```
DECLARE @test_int INT  
  
SET @test_int = 5
```

Assign value to `@my_artist`:

```
DECLARE @my_artist  varchar(100)  
  
SET @my_artist = 'AC/DC'
```

```
DECLARE @my_artist varchar(100)
DECLARE @my_album varchar(300);

SET @my_artist = 'AC/DC'
SET @my_album = 'Let There Be Rock' ;

SELECT --
FROM --
WHERE artist = @my_artist
AND album = @my_album;
```

```
DECLARE @my_artist varchar(100)
DECLARE @my_album varchar(300);

SET @my_artist = 'U2'
SET @my_album = 'Pop' ;

SELECT --
FROM --
WHERE artist = @my_artist
AND album = @my_album;
```

# Temporary tables

**SELECT**

```
col1,  
col2,  
col3 INTO #my_temp_table
```

**FROM** my\_existing\_table

**WHERE**

```
-- Conditions
```

- `#my_temp_table` exists until connection or session ends

```
-- Remove table manually
```

**DROP TABLE** #my\_temp\_table

# **Let's declare some variables!**

**INTRODUCTION TO SQL SERVER**

# Congratulations!

INTRODUCTION TO SQL SERVER



**John MacKintosh**

Instructor

# What we learned...

- Selecting: `SELECT`
- Ordering: `ORDER BY`
- Filtering: `WHERE` and `HAVING`
- Aggregating: `SUM` , `COUNT` , `MIN` , `MAX` and `AVG`
- Text manipulation: `LEFT` , `RIGHT` , `LEN` and `SUBSTRING`

# What we learned (II)...

- GROUP BY
- INNER JOIN , LEFT JOIN , RIGHT JOIN
- UNION and UNION ALL
- Create, Read, Update and Delete (CRUD)
- Variables
- Temporary tables

# Next Steps

- Intermediate SQL Server
- Joining Data in SQL

# **Congratulations!**

**INTRODUCTION TO SQL SERVER**

# Your first database

INTRODUCTION TO RELATIONAL DATABASES IN SQL



Timo Grossenbacher

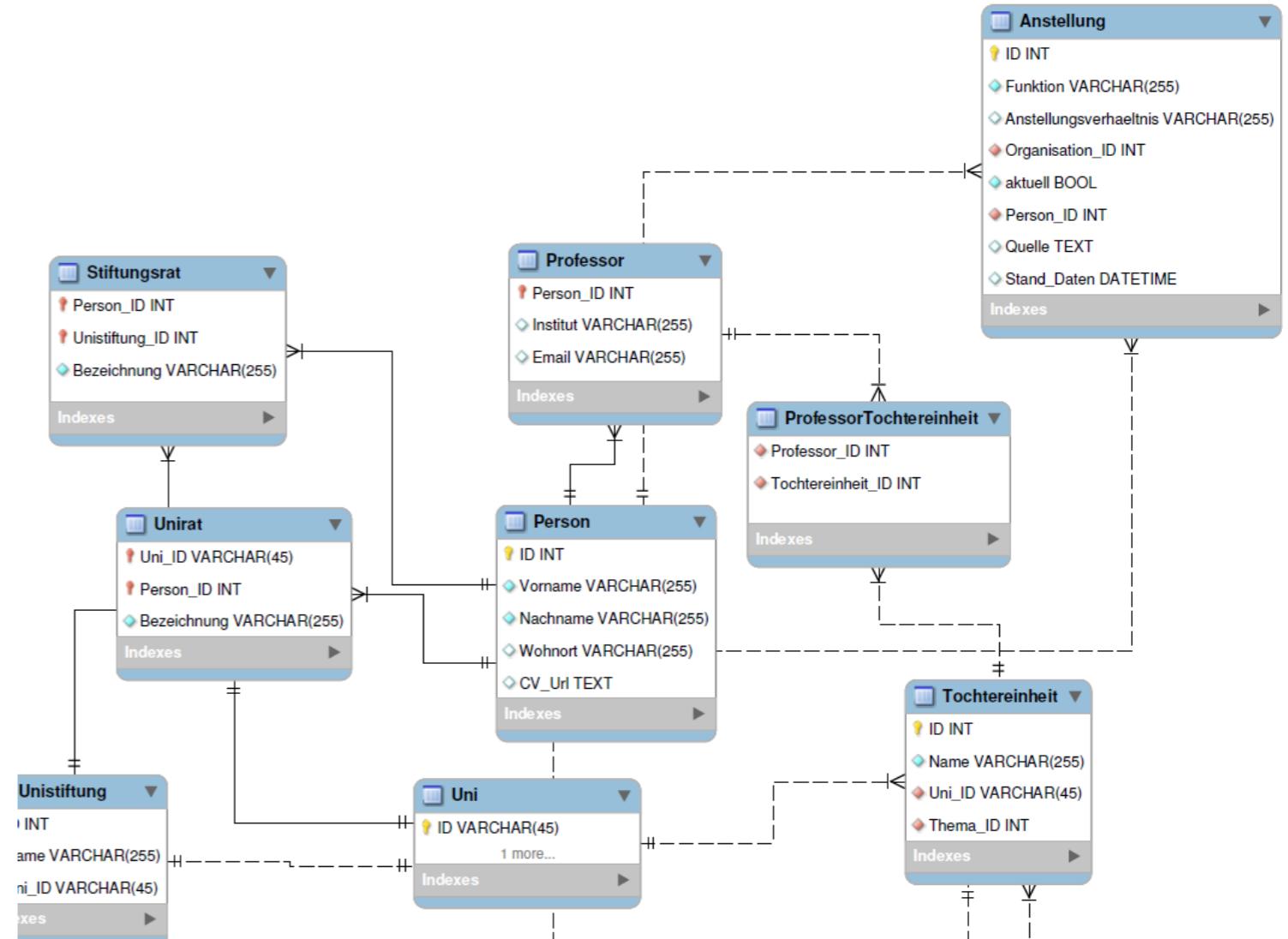
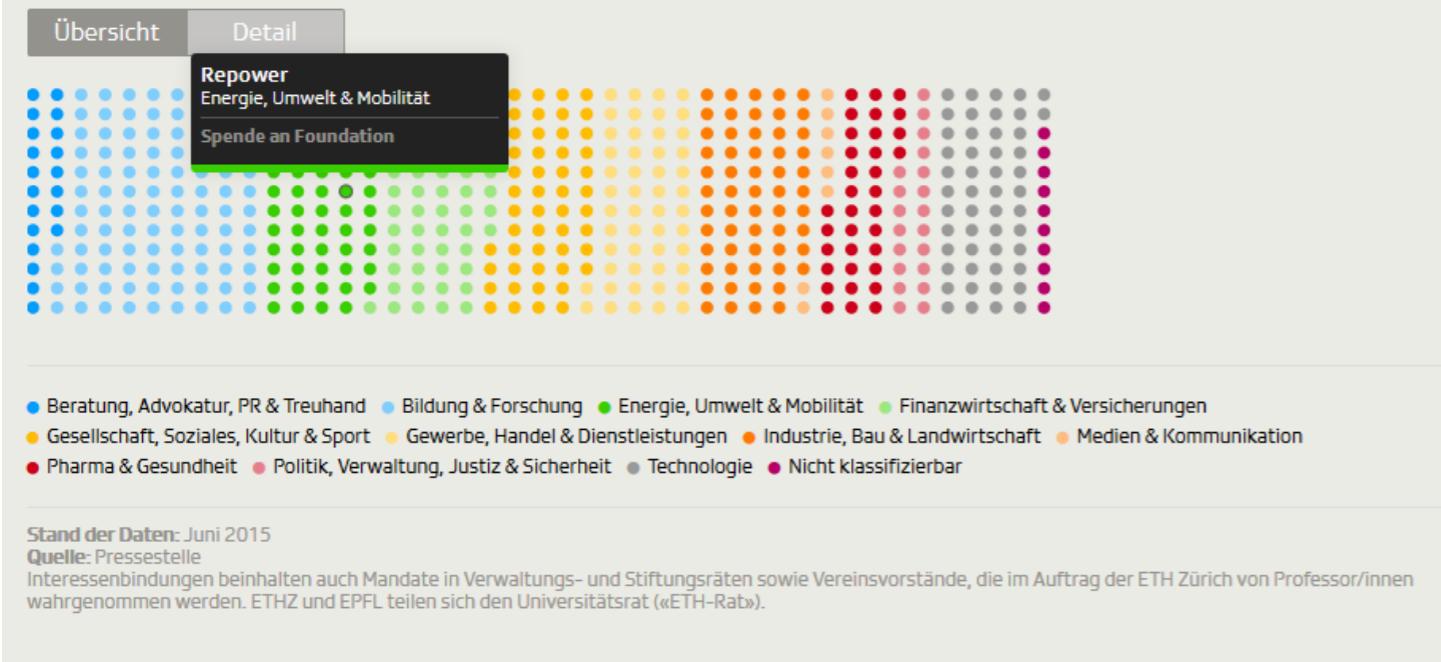
Data Journalist

# Investigating universities in Switzerland

## Eidgenössische Technische Hochschule Zürich

Zu dieser Hochschule gehören rund **18'600** Studierende und **Professor/innen**. Es besteht ein jährlicher Aufwand von rund **1.6 Mrd. Fr.**, wovon **8.8 %** aus privaten Drittmitteln stammen (BFS, 2014).

Jeder Punkt in der Grafik zeigt eine von insgesamt **516** Interessenbindungen. [f](#) [t](#)



# A relational database:

- real-life *entities* become *tables*
  - reduced redundancy
  - data integrity by *relationships*
- e.g. professors , universities , companies
  - e.g. only one entry in companies for the bank "Credit Suisse"
  - e.g. a professor can work at multiple universities and companies , a company can employ multiple professors

# Throughout this course you will:

- work with the data I used for my investigation
- create a relational database from scratch
- learn three concepts:
  - *constraints*
  - *keys*
  - *referential integrity*

You'll need: Basic understanding of SQL, as taught in [Introduction to SQL](#).

# Your first duty: Have a look at the PostgreSQL database

```
SELECT table_schema, table_name  
FROM information_schema.tables;
```

table_schema		table_name
pg_catalog		pg_statistic
pg_catalog		pg_type
pg_catalog		pg_policy
pg_catalog		pg_authid
pg_catalog		pg_shadow
public		university_professors
pg_catalog		pg_settings
...		

# Have a look at the columns of a certain table

```
SELECT table_name, column_name, data_type  
FROM information_schema.columns  
WHERE table_name = 'pg_config';
```

table_name	column_name	data_type
pg_config	name	text
pg_config	setting	text

# **Let's do this.**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Tables: At the core of every database

INTRODUCTION TO RELATIONAL DATABASES IN SQL

SQL

Timo Grossenbacher

Data Journalist

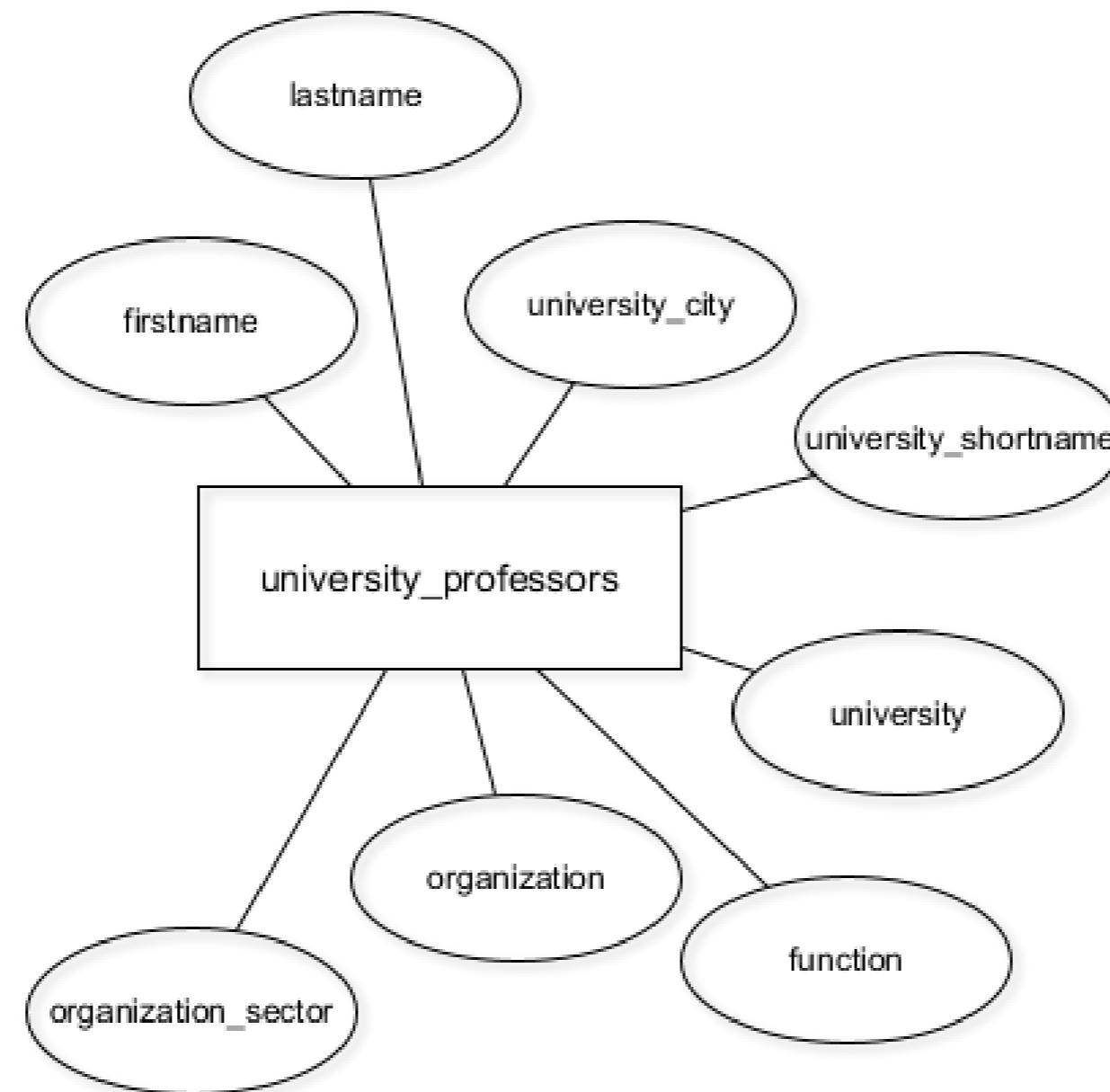
# Redundancy in the university\_professors table

```
SELECT * FROM  
FROM university_professors  
LIMIT 3;
```

```
-[ RECORD 1 ]-----+-----  
firstname          | Karl  
lastname          | Aberer  
university        | ETH Lausanne  
university_shortname | EPF  
university_city    | Lausanne  
function          | Chairman of L3S Advisory Board  
organization        | L3S Advisory Board  
organization_sector | Education & research  
-[ RECORD 2 ]-----+-----  
firstname          | Karl  
lastname          | Aberer  
university        | ETH Lausanne  
university_shortname | EPF  
university_city    | Lausanne  
function          | Member Conseil of Zeno-Karl Schindler Foundation  
organization        | Zeno-Karl Schindler Foundation  
organization_sector | Education & research  
-[ RECORD 3 ]-----+-----  
firstname          | Karl  
lastname          | Aberer  
(truncated)  
function          | Member of Conseil Fondation IDIAP  
organization        | Fondation IDIAP  
(truncated)
```

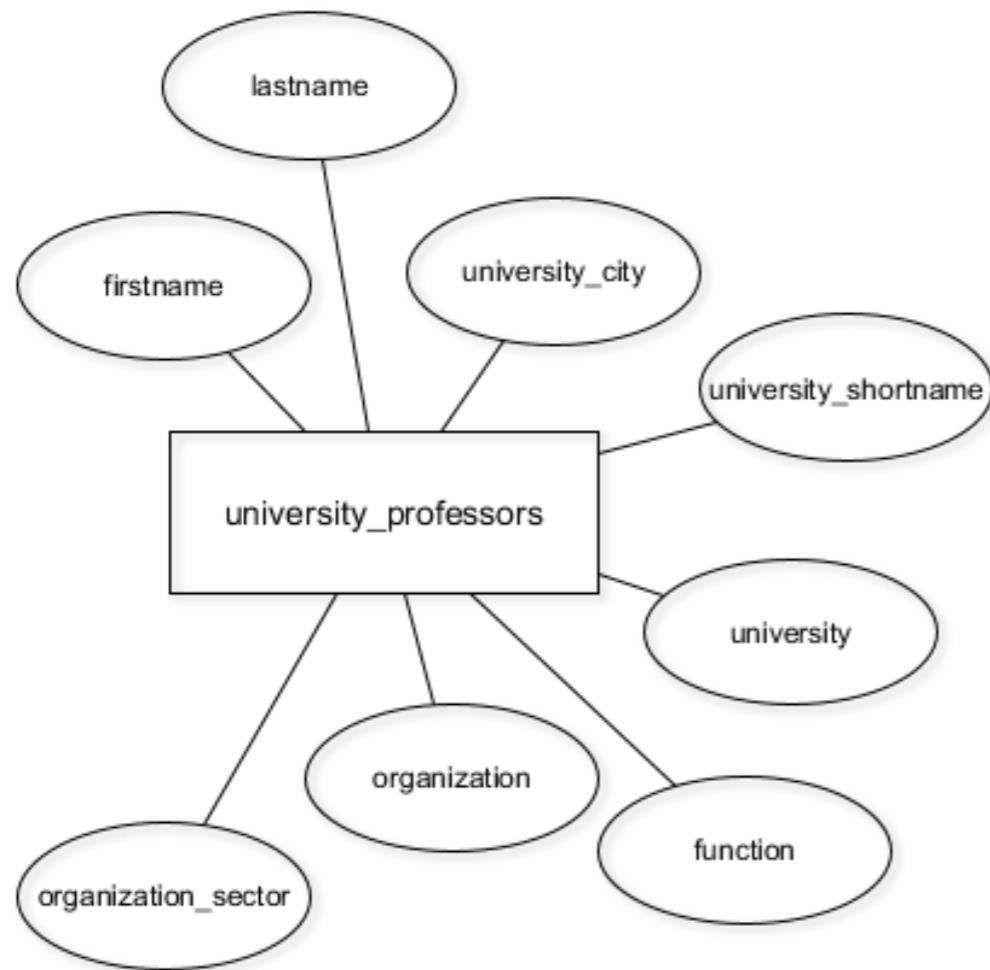
-[ RECORD 1 ]-----	
firstname	Karl
lastname	Aberer
university	ETH Lausanne
university_shortname	EPF
university_city	Lausanne
function	Chairman of L3S Advisory Board
organisation	L3S Advisory Board
organisation_sector	Education & research
-[ RECORD 2 ]-----	
firstname	Karl
lastname	Aberer
university	ETH Lausanne
university_shortname	EPF
university_city	Lausanne
function	Member Conseil of Zeno-Karl Schindler Foundation
organisation	Zeno-Karl Schindler Foundation
organisation_sector	Education & research
-[ RECORD 3 ]-----	
firstname	Karl
lastname	Aberer
(truncated)	
function	Member of Conseil Fondation IDIAP
organisation	Fondation IDIAP
(truncated)	

# Currently: One "entity type" in the database

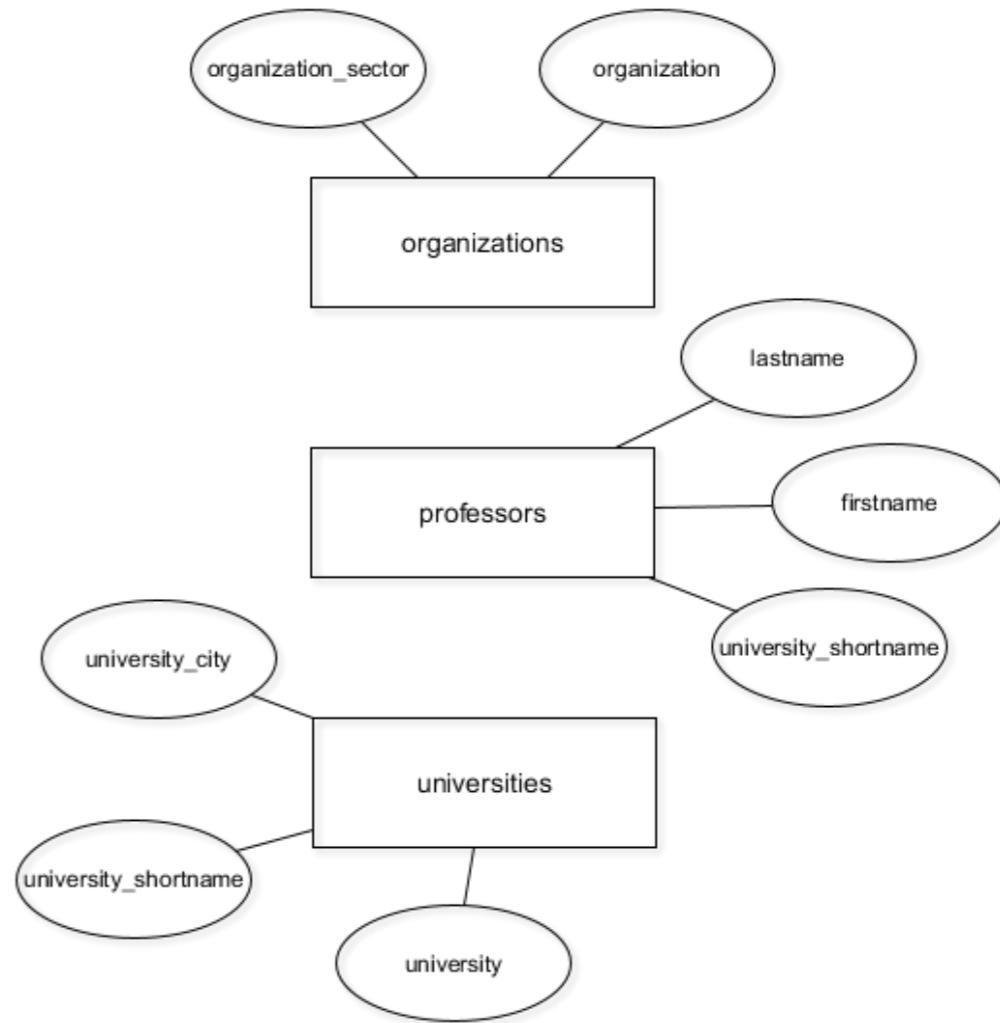


# A better database model with three entity types

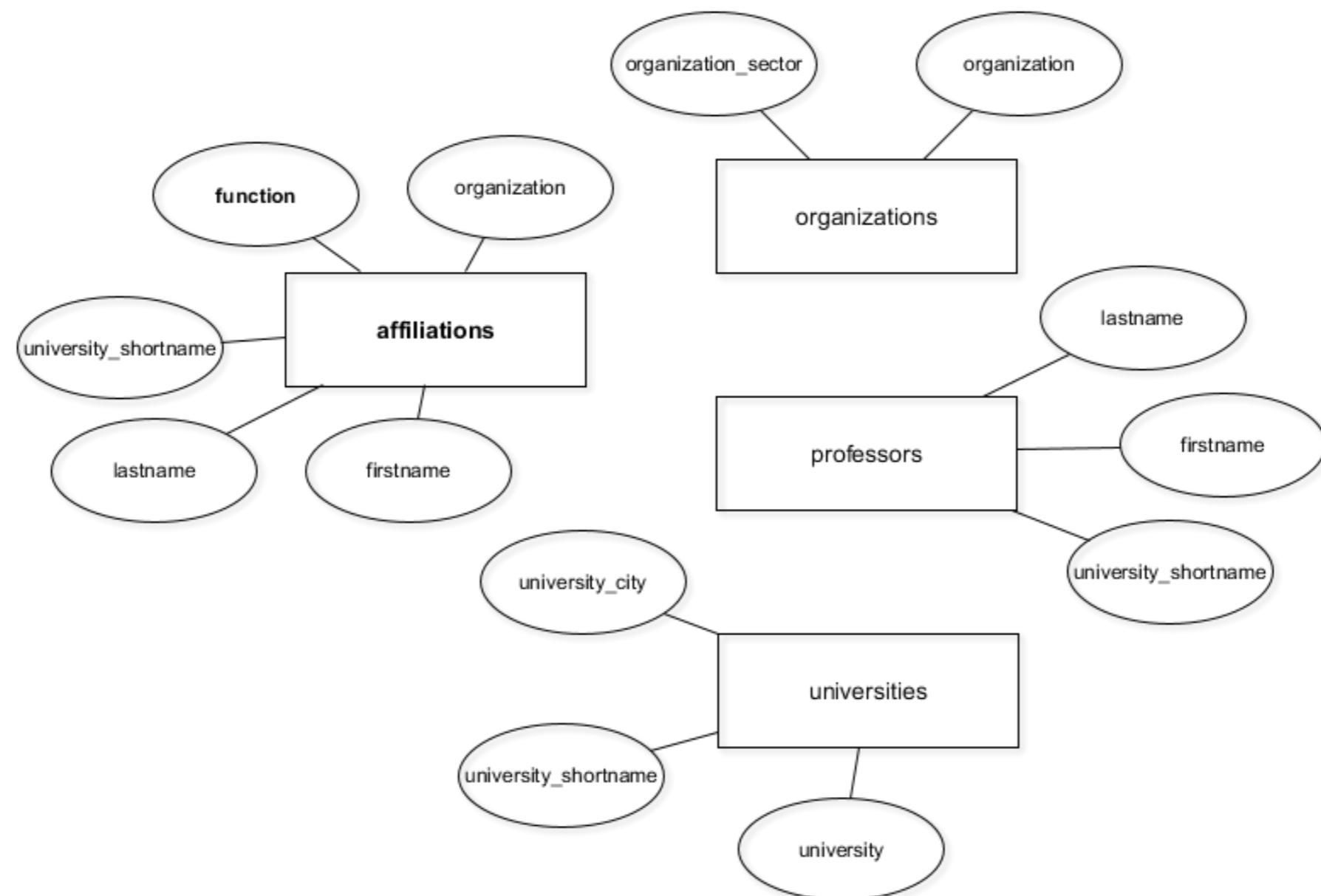
Old:



New:



# A better database model with four entity types



# Create new tables with CREATE TABLE

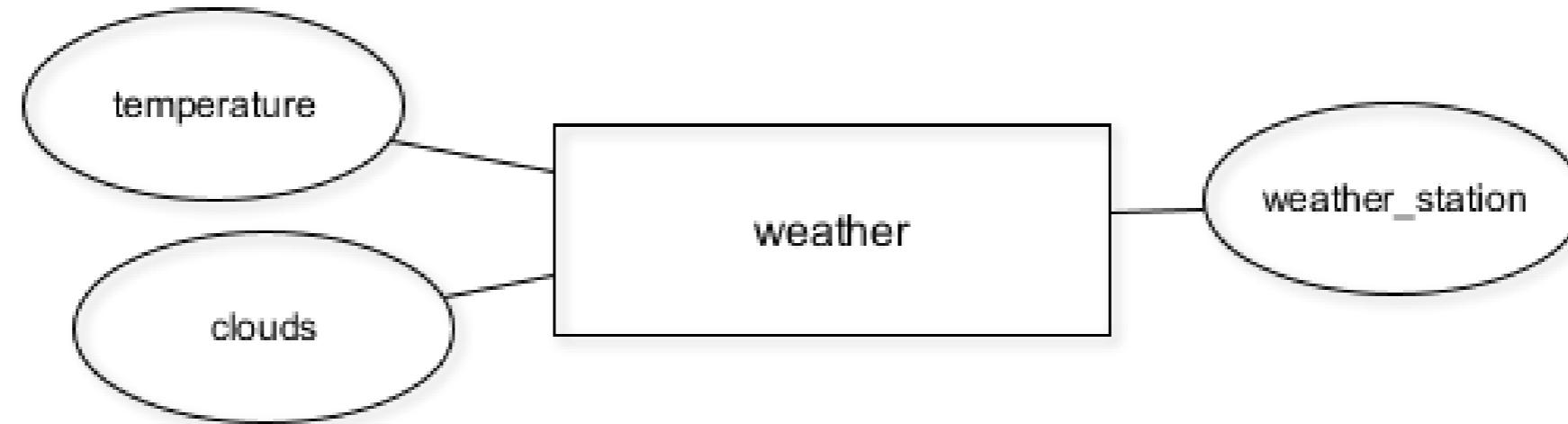
```
CREATE TABLE table_name (  
    column_a data_type,  
    column_b data_type,  
    column_c data_type  
)
```

# Create new tables with CREATE TABLE

```
CREATE TABLE weather (  
    clouds text,  
    temperature numeric,  
    weather_station char(5)  
)
```

To add columns you can use the following SQL query:

```
ALTER TABLE table_name  
ADD COLUMN column_name data_type;
```



# **Let's practice!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Update your database as the structure changes

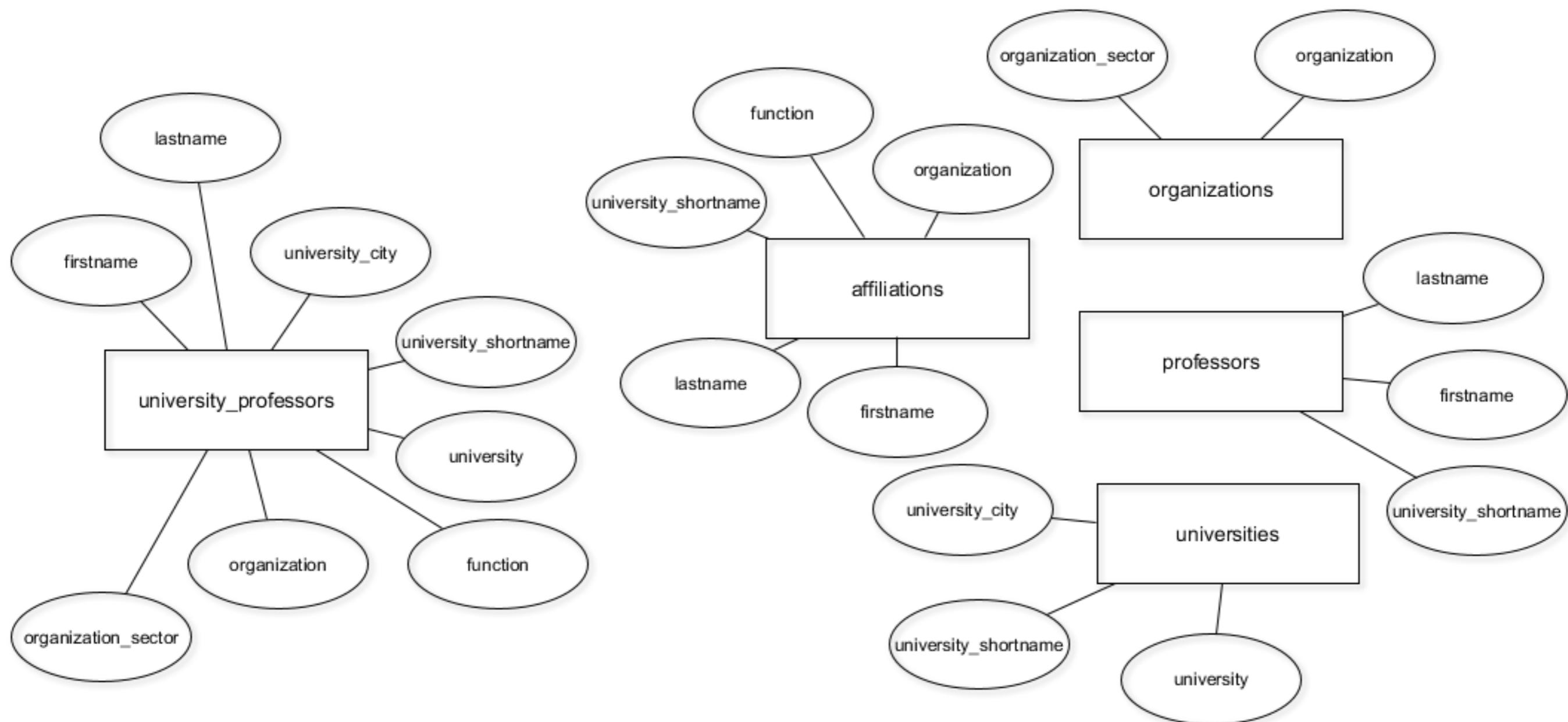
INTRODUCTION TO RELATIONAL DATABASES IN SQL



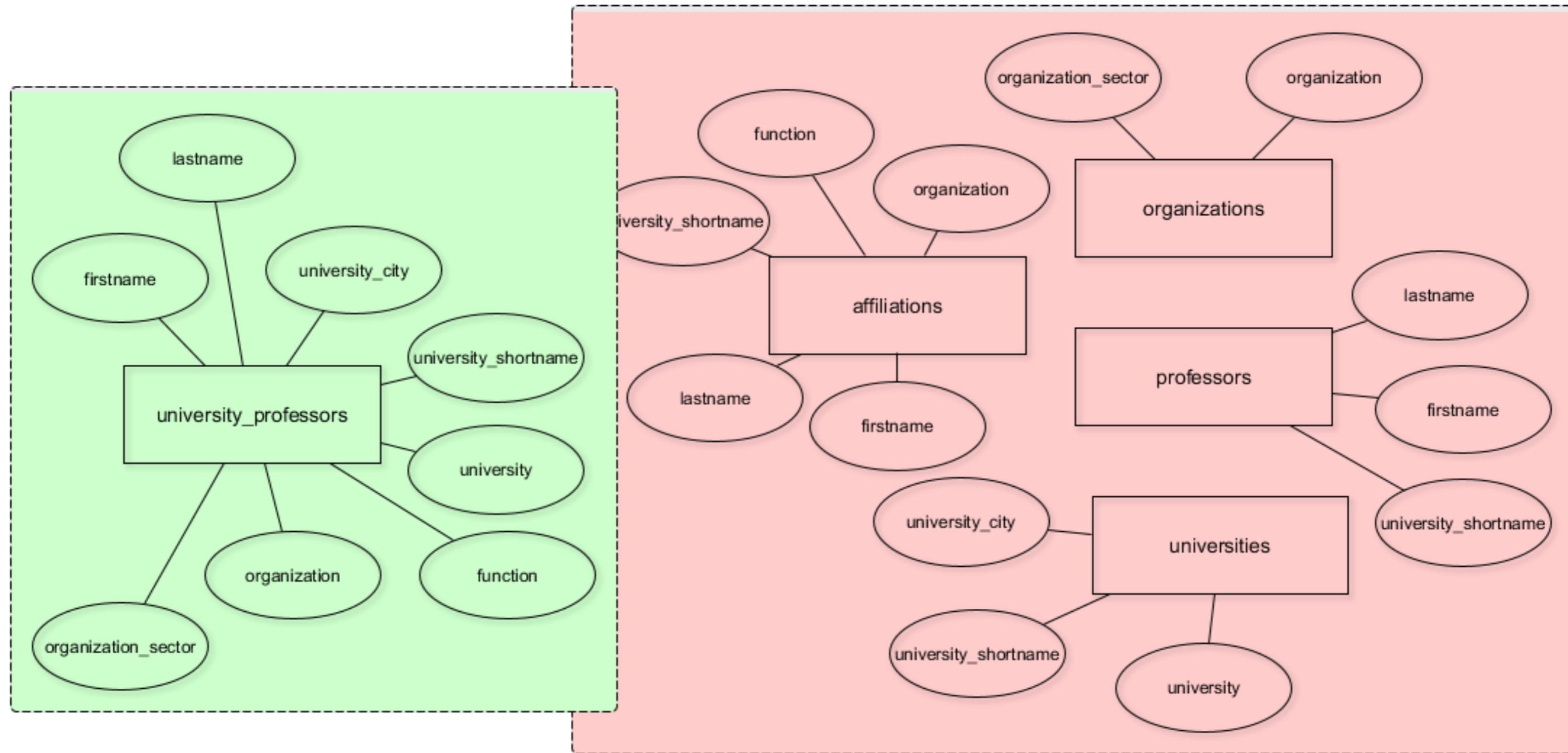
Timo Grossenbacher

Data Journalist

# The current database model



# The current database model



# Only store DISTINCT data in the new tables

```
SELECT COUNT(*)  
FROM university_professors;
```

```
count  
-----  
1377
```

```
SELECT COUNT(DISTINCT organization)  
FROM university_professors;
```

```
count  
-----  
1287
```

# INSERT DISTINCT records INTO the new tables

```
INSERT INTO organizations  
SELECT DISTINCT organization,  
    organization_sector  
FROM university_professors;
```

Output: INSERT 0 1287

```
INSERT INTO organizations  
SELECT organization,  
    organization_sector  
FROM university_professors;
```

Output: INSERT 0 1377

# The INSERT INTO statement

```
INSERT INTO table_name (column_a, column_b)  
VALUES ("value_a", "value_b");
```

# RENAME a COLUMN in affiliations

```
CREATE TABLE affiliations (
    firstname text,
    lastname text,
    university_shortname text,
    function text,
    organisation text
);
```

```
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

# DROP a COLUMN in affiliations

```
CREATE TABLE affiliations (
    firstname text,
    lastname text,
    university_shortname text,
    function text,
    organization text
);
```

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

```
SELECT DISTINCT firstname, lastname,  
    university_shortname  
FROM university_professors  
ORDER BY lastname;
```

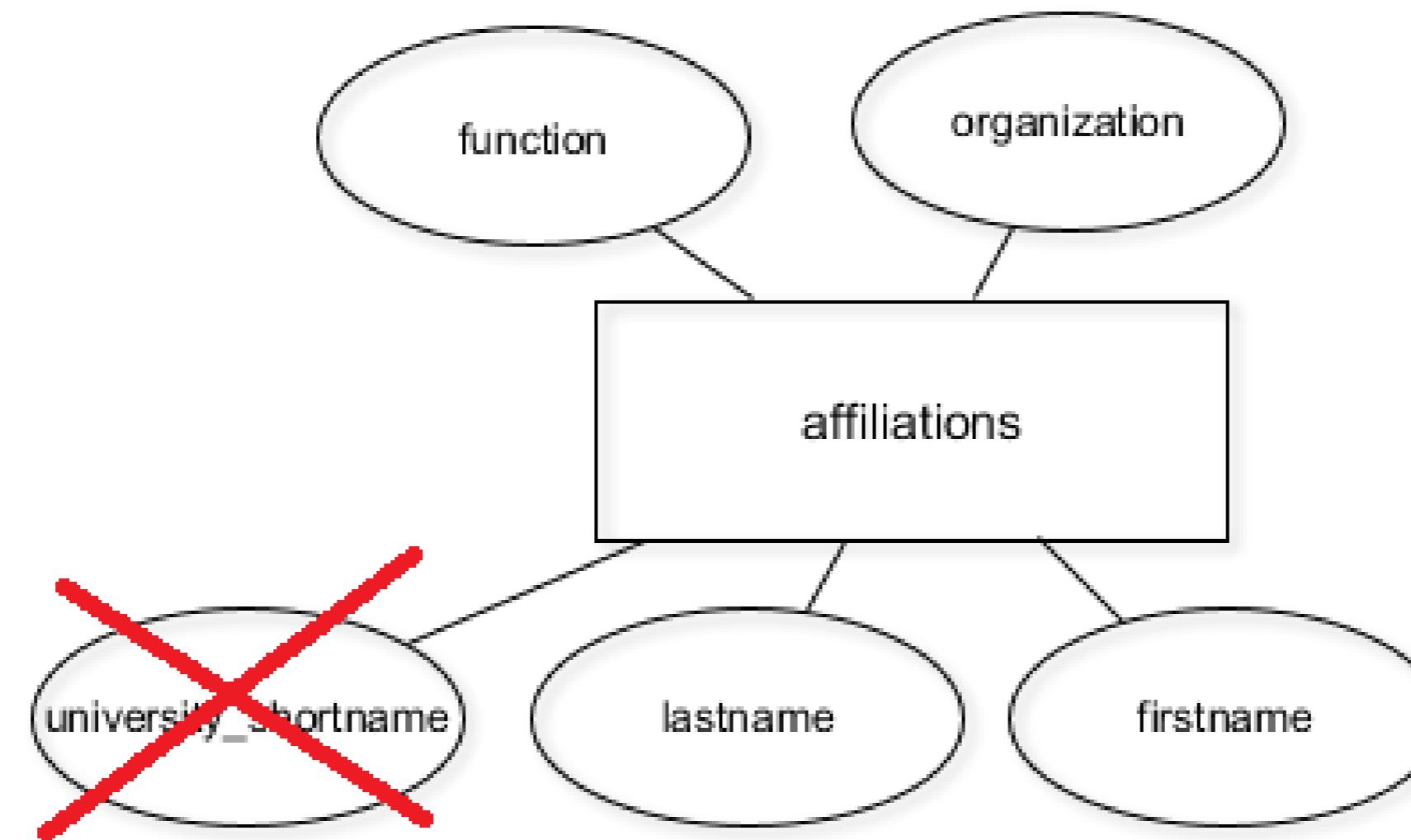
```
-[ RECORD 1 ]-----  
firstname | Karl  
lastname | Aberer  
university_shortname | EPF  
-[ RECORD 2 ]-----  
firstname | Reza Shokrollah  
lastname | Abhari  
university_shortname | ETH  
-[ RECORD 3 ]-----  
firstname | Georges  
lastname | Abou Jaoudé  
university_shortname | EPF  
(truncated)
```

(551 records)

```
SELECT DISTINCT firstname, lastname  
FROM university_professors  
ORDER BY lastname;
```

```
-[ RECORD 1 ]-----  
firstname | Karl  
lastname | Aberer  
-[ RECORD 2 ]-----  
firstname | Reza Shokrollah  
lastname | Abhari  
-[ RECORD 3 ]-----  
firstname | Georges  
lastname | Abou Jaoudé  
(truncated)  
(551 records)
```

# A professor is uniquely identified by firstname, lastname only



# **Let's get to work!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Better data quality with constraints

INTRODUCTION TO RELATIONAL DATABASES IN SQL

SQL

Timo Grossenbacher

Data Journalist

# Integrity constraints

1. **Attribute constraints**, e.g. data types on columns (Chapter 2)
2. **Key constraints**, e.g. primary keys (Chapter 3)
3. **Referential integrity constraints**, enforced through foreign keys (Chapter 4)

# Why constraints?

- Constraints give the data structure
- Constraints help with consistency, and thus data quality
- Data quality is a business advantage / data science prerequisite
- Enforcing is difficult, but PostgreSQL helps

# Data types as attribute constraints

Name	Aliases	Description
<code>bigint</code>	<code>int8</code>	signed eight-byte integer
<code>bigserial</code>	<code>serial8</code>	autoincrementing eight-byte integer
<code>bit [ (n) ]</code>		fixed-length bit string
<code>bit varying [ (n) ]</code>	<code>varbit [ (n) ]</code>	variable-length bit string
<code>boolean</code>	<code>bool</code>	logical Boolean (true/false)
<code>box</code>		rectangular box on a plane
<code>bytea</code>		binary data ("byte array")
<code>character [ (n) ]</code>	<code>char [ (n) ]</code>	fixed-length character string
<code>character varying [ (n) ]</code>	<code>varchar [ (n) ]</code>	variable-length character string
<code>cidr</code>		IPv4 or IPv6 network address

From the [PostgreSQL documentation](#).

# Dealing with data types (casting)

```
CREATE TABLE weather (
    temperature integer,
    wind_speed text);
SELECT temperature * wind_speed AS wind_chill
FROM weather;
```

operator does not exist: integer \* text  
HINT: No operator matches the given name and argument type(s).  
You might need to add explicit type casts.

```
SELECT temperature * CAST(wind_speed AS integer) AS wind_chill
FROM weather;
```

# **Let's practice!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Working with data types

INTRODUCTION TO RELATIONAL DATABASES IN SQL

SQL

Timo Grossenbacher

Data Journalist

# Working with data types

- Enforced on columns (i.e. attributes)
- Define the so-called "domain" of a column
- Define what operations are possible
- Enforce consistent storage of values

# The most common types

- `text` : character strings of any length
- `varchar [ (x) ]` : a maximum of `n` characters
- `char [ (x) ]` : a fixed-length string of `n` characters
- `boolean` : can only take three states, e.g. `TRUE` , `FALSE` and `NULL` (unknown)

From the [PostgreSQL documentation](#).

# The most common types (cont'd.)

- `date` , `time` and `timestamp` : various formats for date and time calculations
- `numeric` : arbitrary precision numbers, e.g. `3.1457`
- `integer` : whole numbers in the range of `-2147483648` and `+2147483647`

From the [PostgreSQL documentation](#).

# Specifying types upon table creation

```
CREATE TABLE students (  
    ssn integer,  
    name varchar(64),  
    dob date,  
    average_grade numeric(3, 2), -- e.g. 5.54  
    tuition_paid boolean  
)
```

# Alter types after table creation

```
ALTER TABLE students  
ALTER COLUMN name  
TYPE varchar(128);
```

```
ALTER TABLE students  
ALTER COLUMN average_grade  
TYPE integer  
-- Turns 5.54 into 6, not 5, before type conversion  
USING ROUND(average_grade);
```

For this, you can use the following syntax:

```
ALTER TABLE table_name  
ALTER COLUMN column_name  
TYPE varchar(x)  
USING SUBSTRING(column_name FROM 1 FOR x)
```

# **Let's apply this!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# The not-null and unique constraints

INTRODUCTION TO RELATIONAL DATABASES IN SQL

SQL

Timo Grossenbacher

Data Journalist

# The not-null constraint

- Disallow `NULL` values in a certain column
- Must hold true for the current state
- Must hold true for any future state

# What does NULL mean?

- unknown
- does not exist
- does not apply
- ...

# What does NULL mean? An example

```
CREATE TABLE students (
    ssn integer not null,
    lastname varchar(64) not null,
    home_phone integer,
    office_phone integer
);
```

NULL != NULL

# How to add or remove a not-null constraint

When creating a table...

```
CREATE TABLE students (
    ssn integer not null,
    lastname varchar(64) not null,
    home_phone integer,
    office_phone integer
);
```

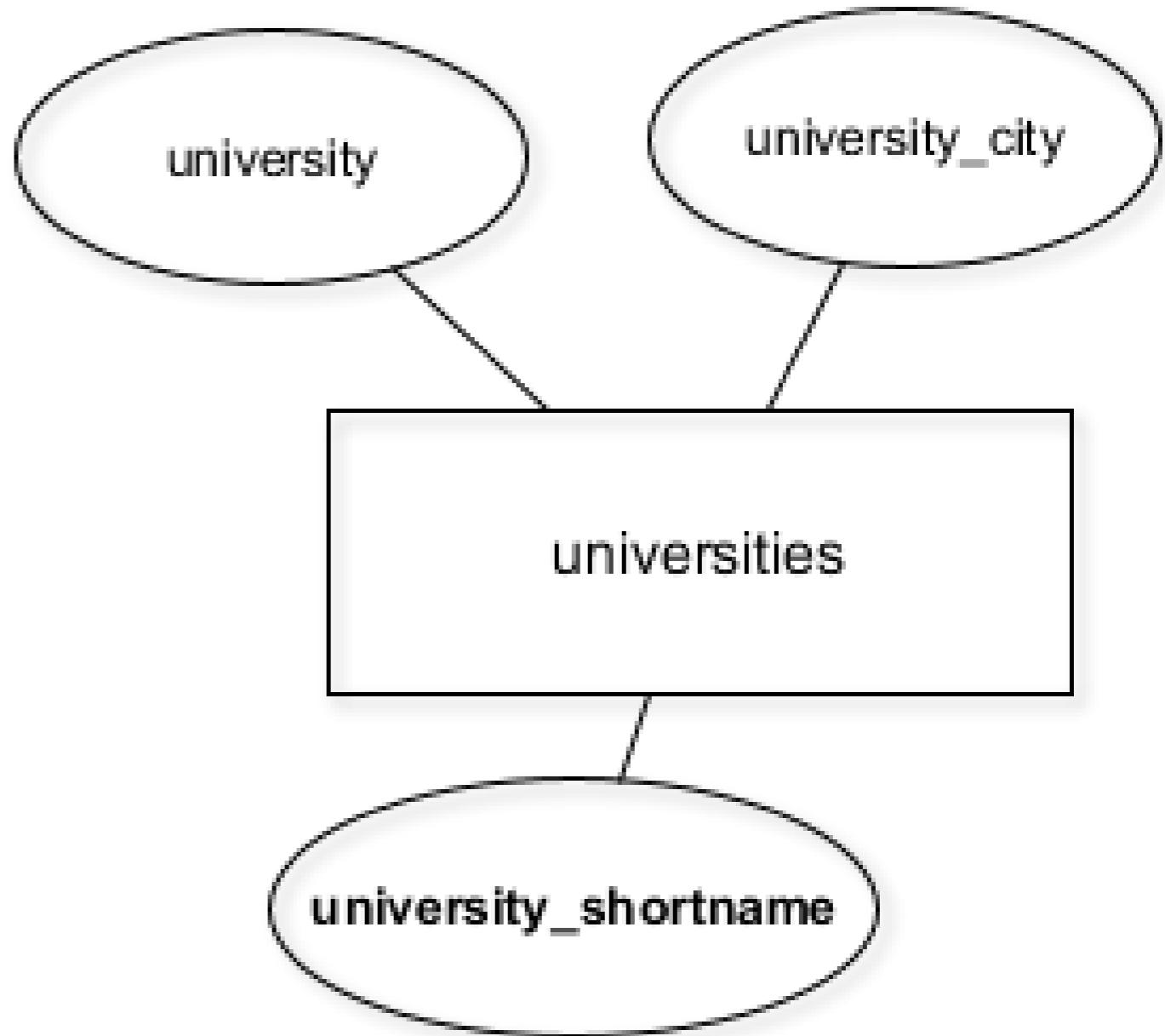
After the table has been created...

```
ALTER TABLE students
ALTER COLUMN home_phone
SET NOT NULL;
```

```
ALTER TABLE students
ALTER COLUMN ssn
DROP NOT NULL;
```

# The unique constraint

- Disallow duplicate values in a column
- Must hold true for the current state
- Must hold true for any future state



# Adding unique constraints

```
CREATE TABLE table_name (  
    column_name UNIQUE  
);
```

```
ALTER TABLE table_name  
ADD CONSTRAINT some_name UNIQUE(column_name);
```

**Let's apply this to  
the database!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Keys and superkeys

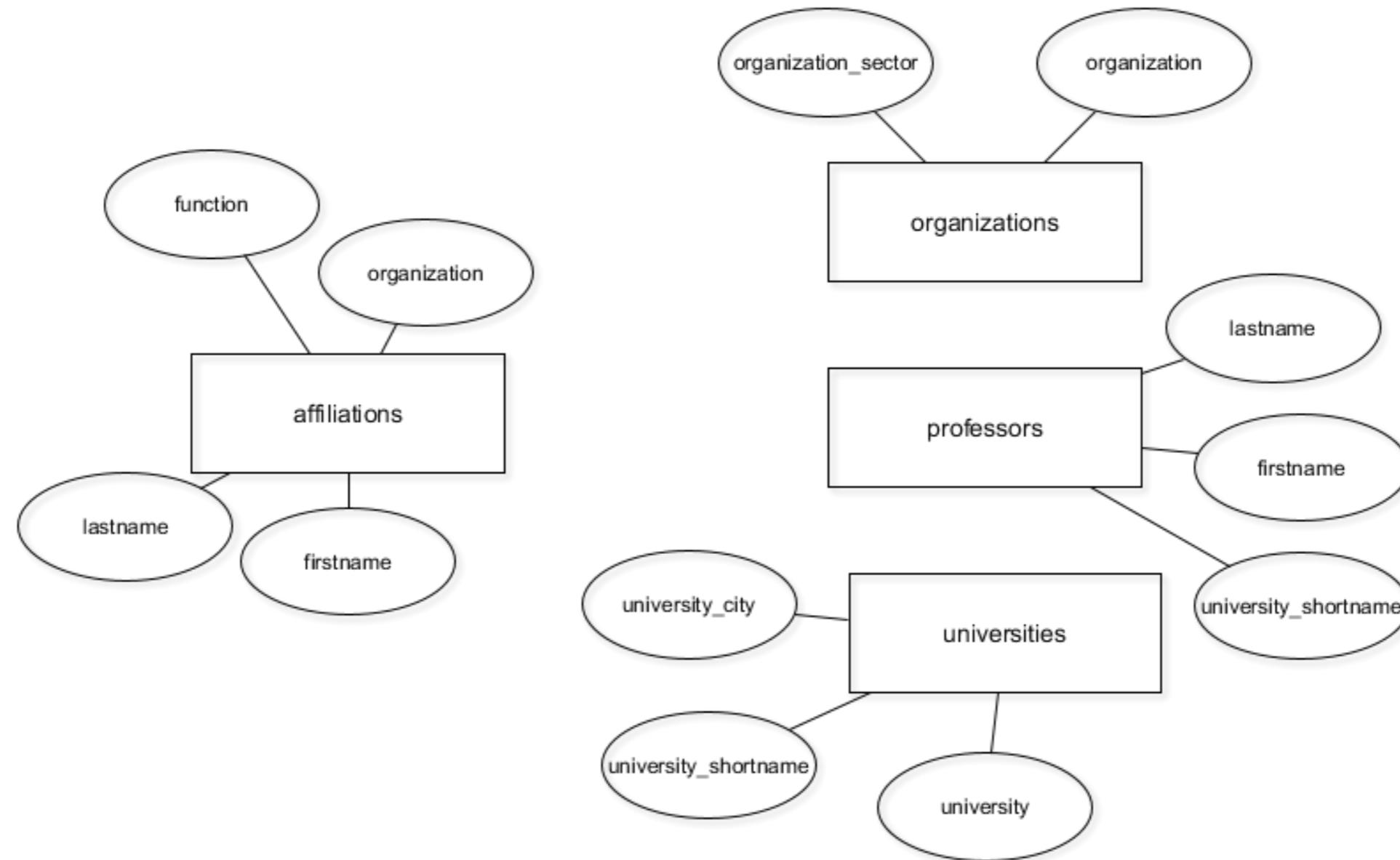
INTRODUCTION TO RELATIONAL DATABASES IN SQL



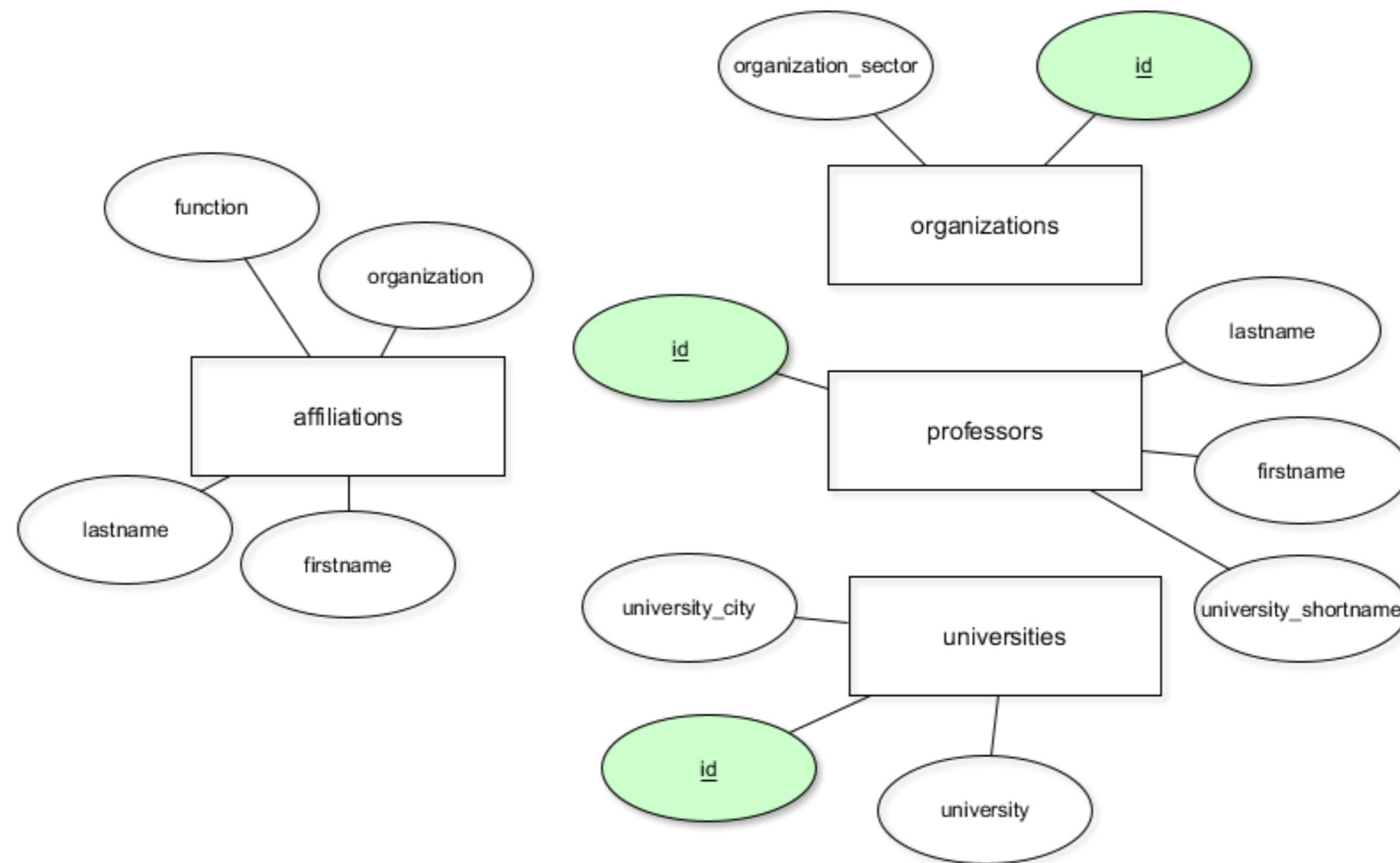
Timo Grossenbacher

Data Journalist

# The current database model



# The database model with primary keys



# What is a key?

- Attribute(s) that identify a record uniquely
- As long as attributes can be removed: **superkey**
- If no more attributes can be removed: minimal superkey or **key**

license_no	serial_no	make	model	year
Texas ABC-739	A69352	Ford	Mustang	2
Florida TVP-347	B43696	Oldsmobile	Cutlass	5
New York MPO-22	X83554	Oldsmobile	Delta	1
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	4
Texas RSK-629	U028365	Jaguar	XJS	4

SK1 = {license\_no, serial\_no, make, model, year}

SK2 = {license\_no, serial\_no, make, model}

SK3 = {make, model, year}, SK4 = {license\_no, serial\_no}, SKi, ..., SKn

*Adapted from Elmasri, Navathe (2011): Fundamentals of Database Systems, 6th Ed., Pearson*

license_no	serial_no	make	model	year
Texas ABC-739	A69352	Ford	Mustang	2
Florida TVP-347	B43696	Oldsmobile	Cutlass	5
New York MPO-22	X83554	Oldsmobile	Delta	1
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	4
Texas RSK-629	U028365	Jaguar	XJS	4

K1 = {license\_no}; K2 = {serial\_no}; K3 = {model}; K4 = {make, year}

- K1 to 3 only consist of one attribute
- Removing either "make" or "year" from K4 would result in duplicates
- Only one candidate key can be the *chosen* key

# **Let's discover some keys!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Primary keys

INTRODUCTION TO RELATIONAL DATABASES IN SQL



Timo Grossenbacher

Data Journalist

# Primary keys

- One primary key per database table, chosen from candidate keys
- Uniquely identifies records, e.g. for referencing in other tables
- Unique and not-null constraints both apply
- Primary keys are time-invariant: choose columns wisely!

# Specifying primary keys

```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name text,
    price numeric
);
```

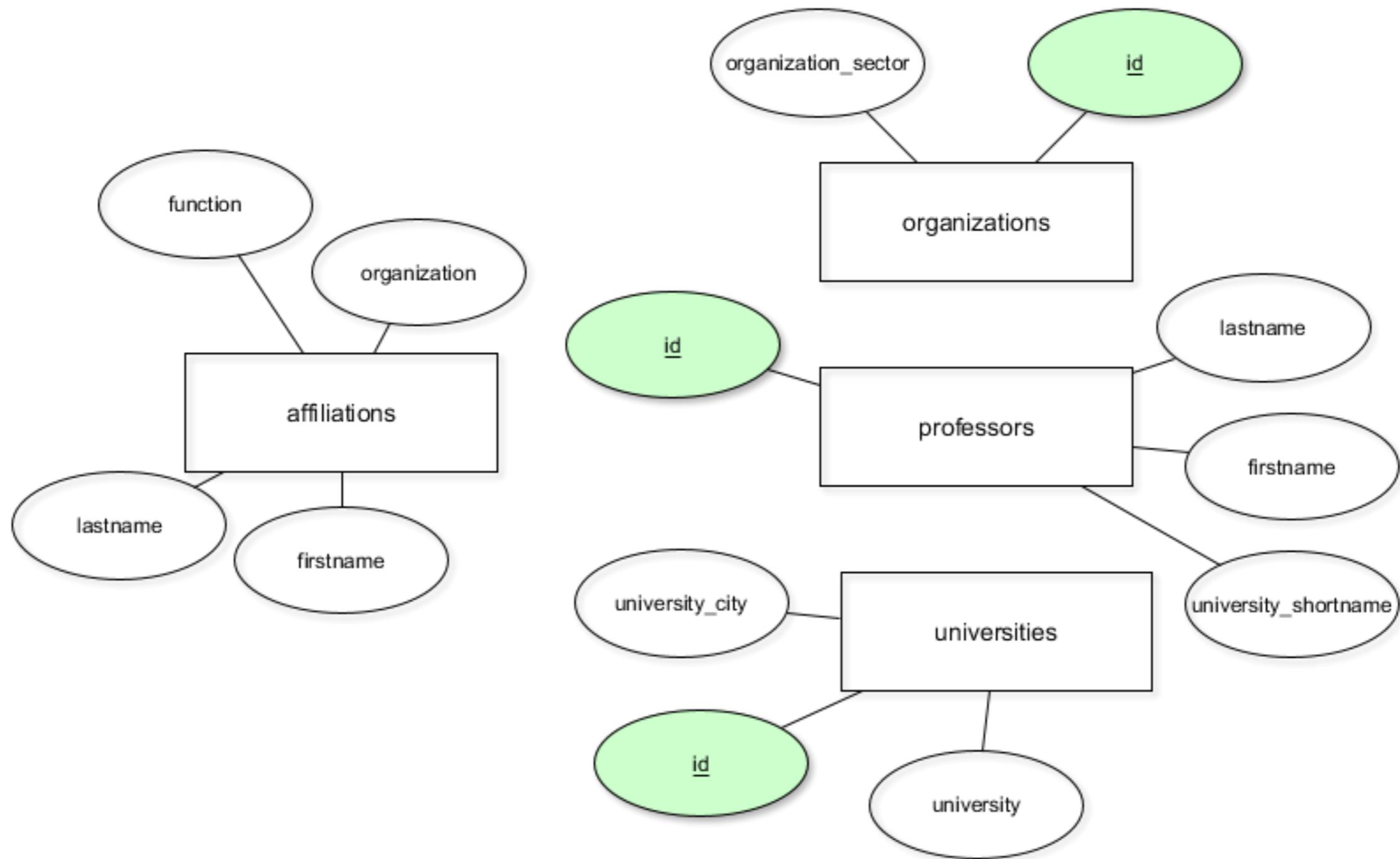
```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

Taken from the [PostgreSQL documentation](#).

# Specifying primary keys (contd.)

```
ALTER TABLE table_name  
ADD CONSTRAINT some_name PRIMARY KEY (column_name)
```



# **Let's practice!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Surrogate keys

INTRODUCTION TO RELATIONAL DATABASES IN SQL

A dark blue circular icon containing the letters "SQL" in white.

Timo Grossenbacher

Data Journalist

# Surrogate keys

- Primary keys should be built from as few columns as possible
- Primary keys should never change over time

license_no	serial_no	make	model	color
Texas ABC-739	A69352	Ford	Mustang	blue
Florida TVP-347	B43696	Oldsmobile	Cutlass	black
New York MP0-22	X83554	Oldsmobile	Delta	silver
California 432-TFY	C43742	Mercedes	190-D	champagne
California RSK-629	Y82935	Toyota	Camry	red
Texas RSK-629	U028365	Jaguar	XJS	blue

make	model	color
Ford	Mustang	blue
Oldsmobile	Cutlass	black
Oldsmobile	Delta	silver
Mercedes	190-D	champagne
Toyota	Camry	red
Jaguar	XJS	blue

# Adding a surrogate key with serial data type

```
ALTER TABLE cars  
ADD COLUMN id serial PRIMARY KEY;  
INSERT INTO cars  
VALUES ('Volkswagen', 'Blitz', 'black');
```

make	model	color	id
Ford	Mustang	blue	1
Oldsmobile	Cutlass	black	2
Oldsmobile	Delta	silver	3
Mercedes	190-D	champagne	4
Toyota	Camry	red	5
Jaguar	XJS	blue	6
Volkswagen	Blitz	black	7

# Adding a surrogate key with serial data type (contd.)

```
INSERT INTO cars  
VALUES ('Opel', 'Astra', 'green', 1);
```

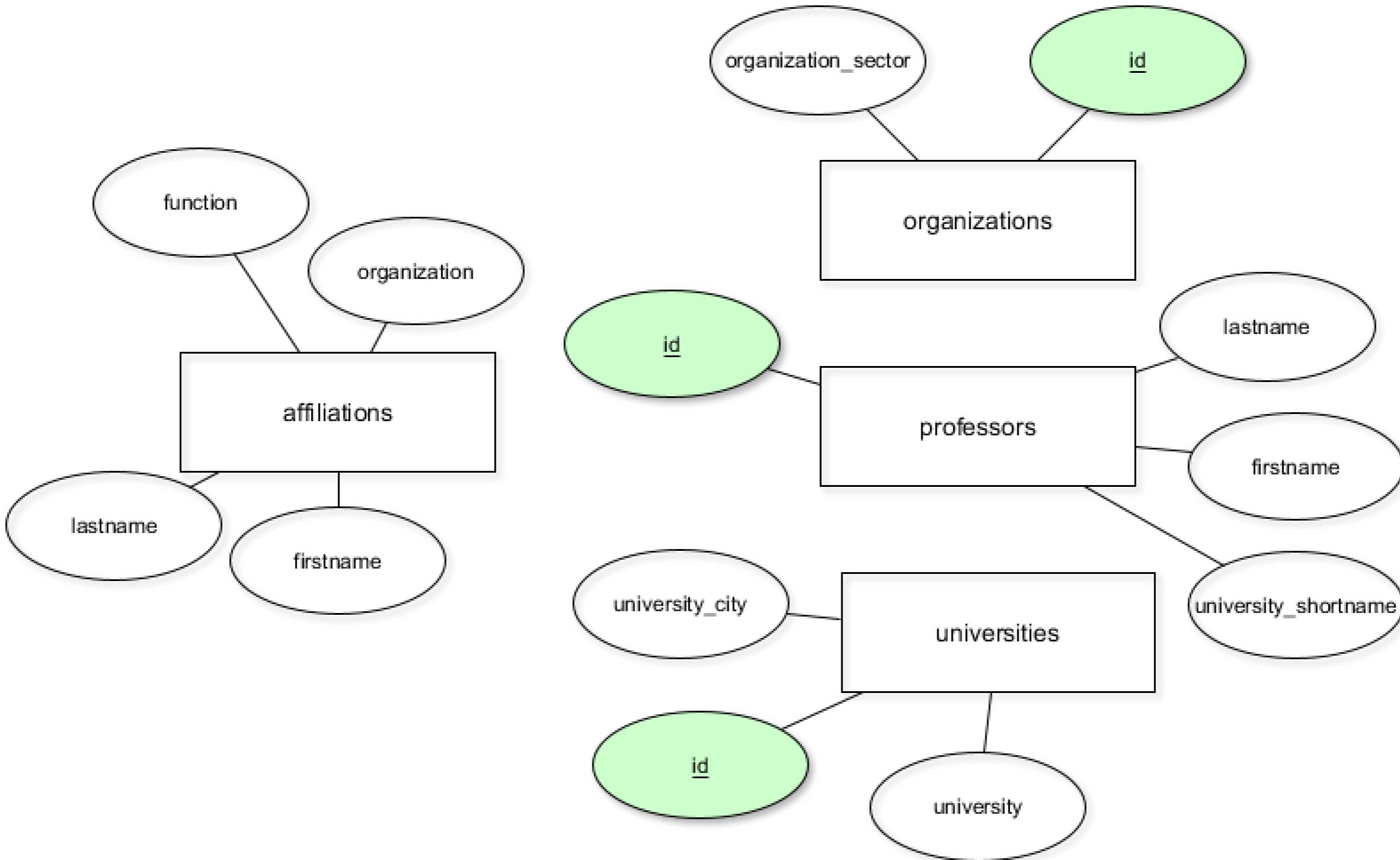
```
duplicate key value violates unique constraint "id_pkey"  
DETAIL: Key (id)=(1) already exists.
```

- "id" uniquely identifies records in the table – useful for referencing!

# Another type of surrogate key

```
ALTER TABLE table_name  
ADD COLUMN column_c varchar(256);
```

```
UPDATE table_name  
SET column_c = CONCAT(column_a, column_b);  
ALTER TABLE table_name  
ADD CONSTRAINT pk PRIMARY KEY (column_c);
```



# **Let's try this!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Model 1:N relationships with foreign keys

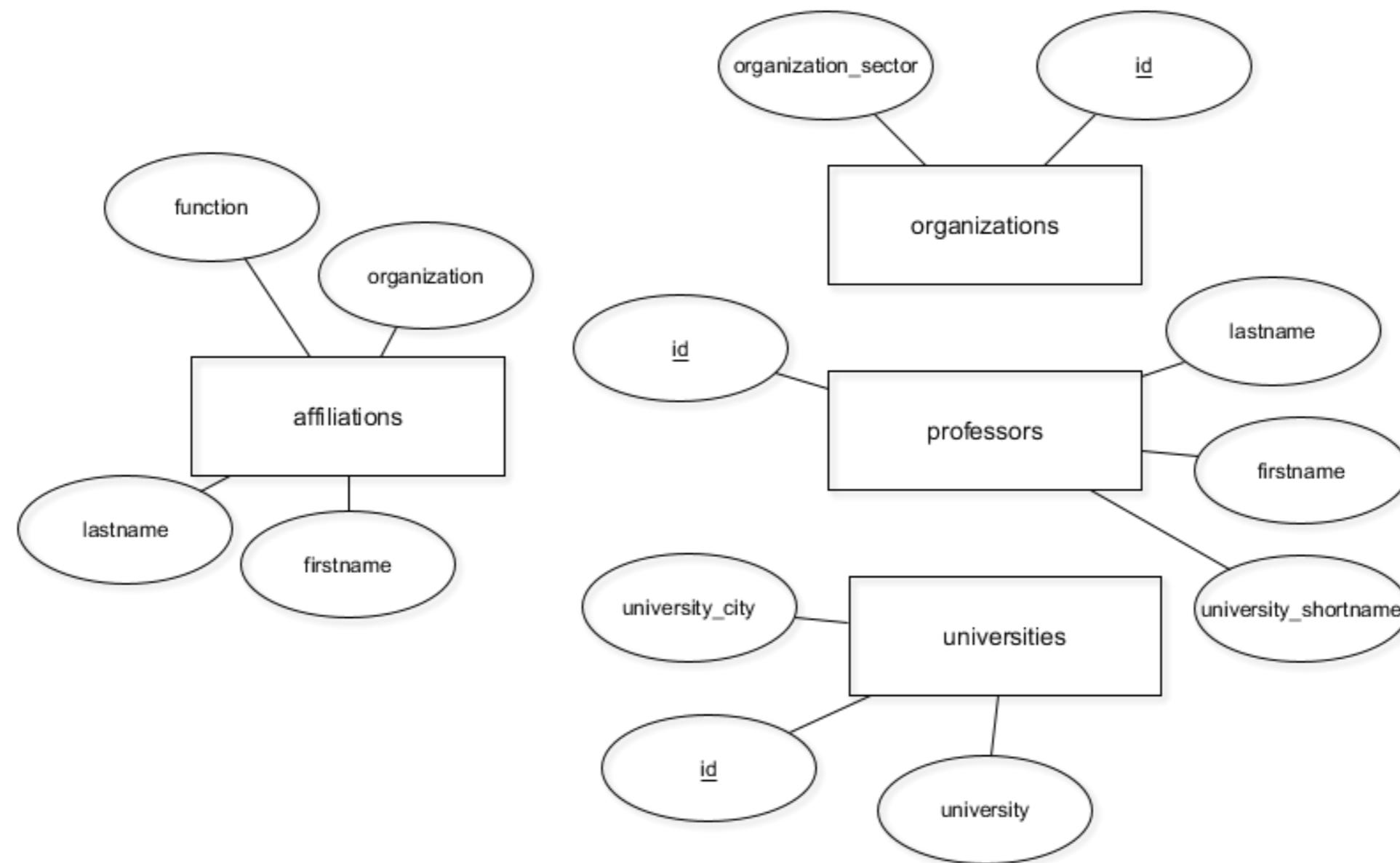
INTRODUCTION TO RELATIONAL DATABASES IN SQL



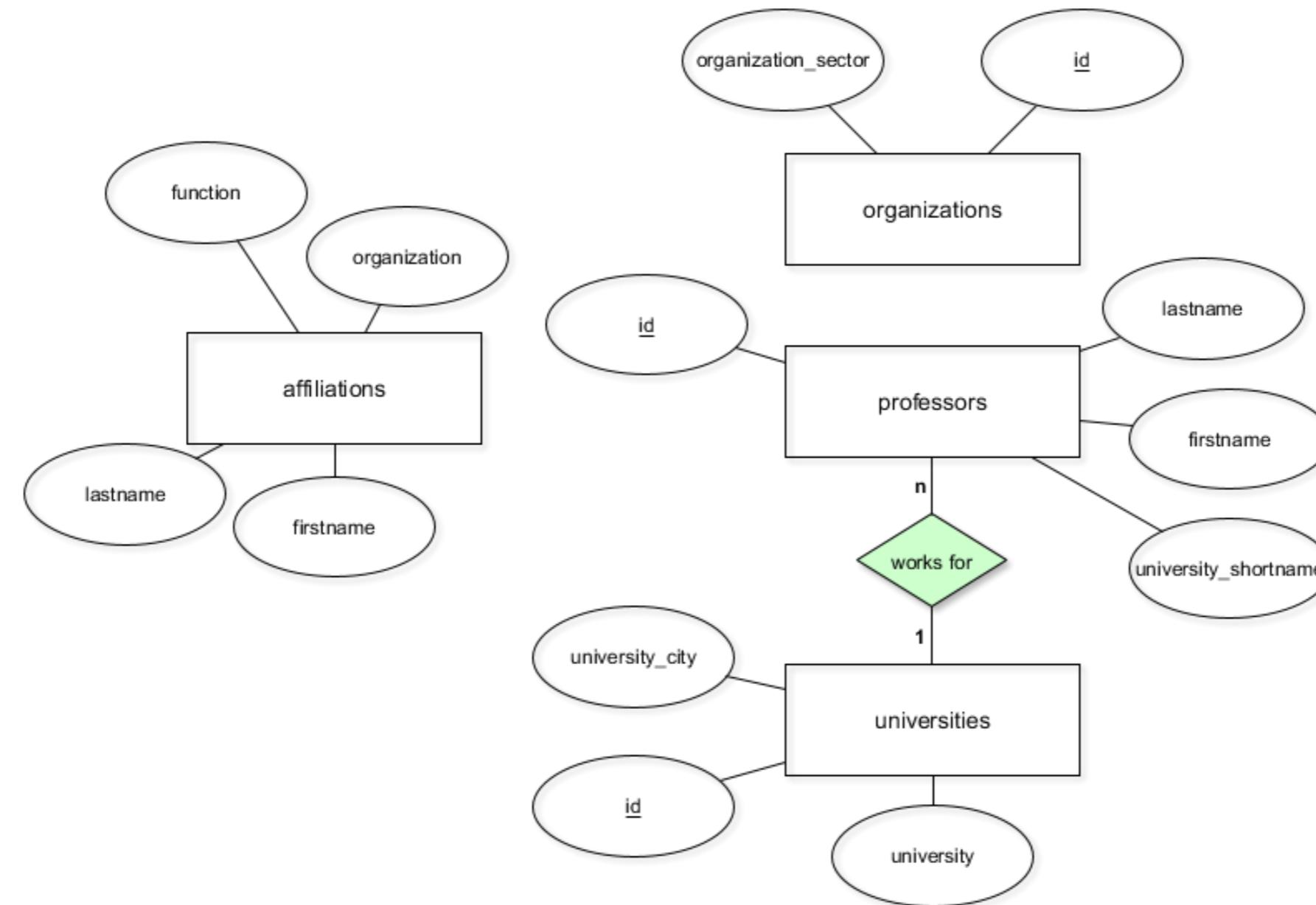
Timo Grossenbacher

Data Journalist

# The current database model



# The next database model



# Implementing relationships with foreign keys

- A foreign key (FK) points to the primary key (PK) of another table
- Domain of FK must be equal to domain of PK
- Each value of FK must exist in PK of the other table (FK constraint or "referential integrity")
- FKS are not actual *keys*

# A query

```
SELECT * FROM professors LIMIT 8;
```

<code>id</code>	<code>firstname</code>	<code>lastname</code>	<code>university_s..</code>
1	Karl	Aberer	EPF
2	Reza Shokrollah	Abhari	ETH
3	Georges	Abou Jaoudé	EPF
4	Hugues	Abriel	UBE
5	Daniel	Aebersold	UBE
6	Marcelo	Aebi	ULA
7	Christoph	Aebi	UBE
8	Patrick	Aebischer	EPF

```
SELECT * FROM universities;
```

<code>id</code>	<code>university</code>	<code>university_city</code>
EPF	ETH Lausanne	Lausanne
ETH	ETH Zürich	Zurich
UBA	Uni Basel	Basel
UBE	Uni Bern	Bern
UFR	Uni Freiburg	Fribourg
UGE	Uni Genf	Geneva
ULA	Uni Lausanne	Lausanne
UNE	Uni Neuenburg	Neuchâtel
USG	Uni St. Gallen	Saint Gallen
USI	USI Lugano	Lugano
UZH	Uni Zürich	Zurich

# Specifying foreign keys

```
CREATE TABLE manufacturers (  
    name varchar(255) PRIMARY KEY);
```

```
INSERT INTO manufacturers  
VALUES ('Ford'), ('VW'), ('GM');
```

```
CREATE TABLE cars (  
    model varchar(255) PRIMARY KEY,  
    manufacturer_name varchar(255) REFERENCES manufacturers (name));
```

```
INSERT INTO cars  
VALUES ('Ranger', 'Ford'), ('Beetle', 'VW');
```

```
-- Throws an error!  
INSERT INTO cars  
VALUES ('Tundra', 'Toyota');
```

# Specifying foreign keys to existing tables

```
ALTER TABLE a  
ADD CONSTRAINT a_fkey FOREIGN KEY (b_id) REFERENCES b (id);
```

# **Let's implement this!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Model more complex relationships

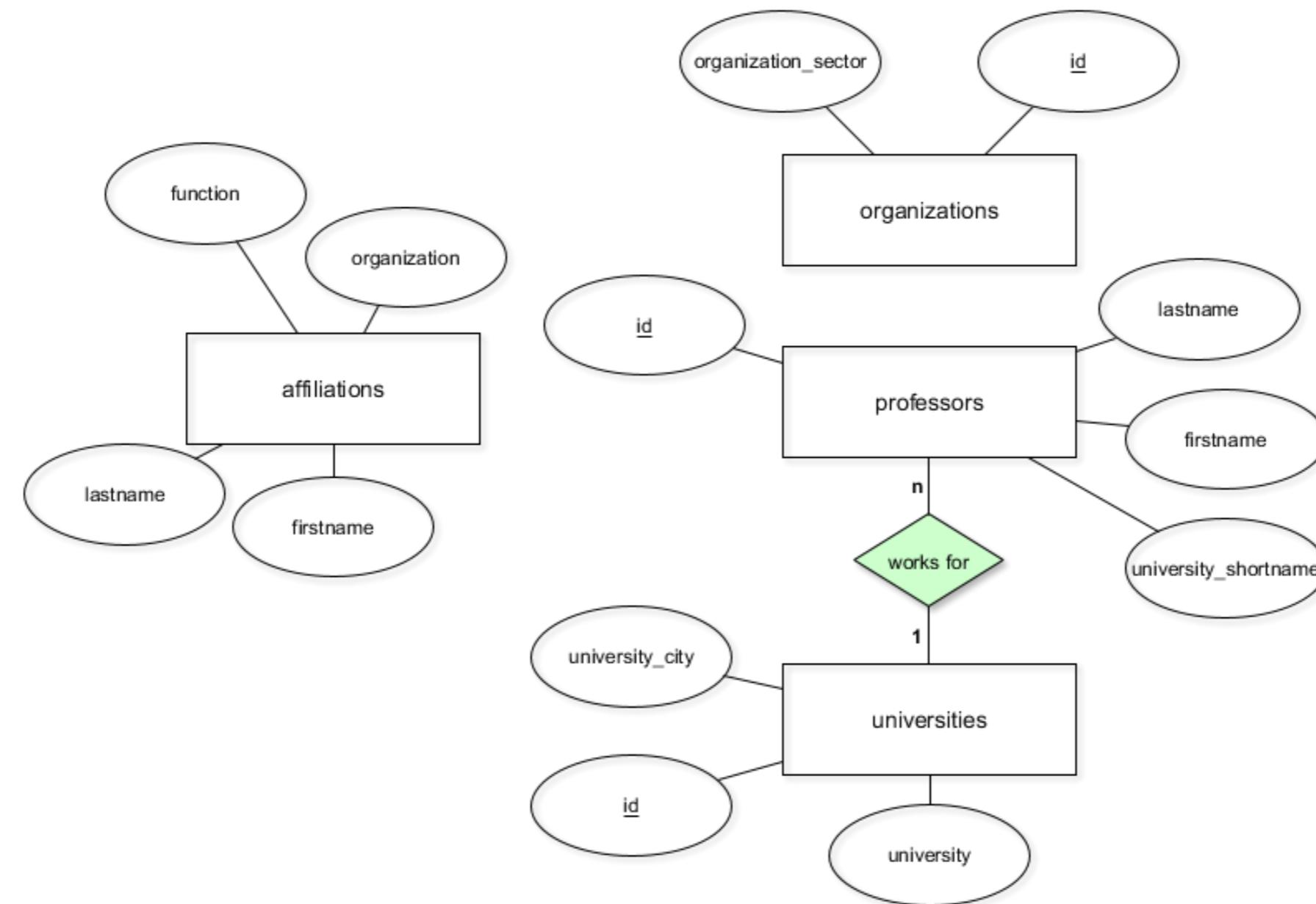
INTRODUCTION TO RELATIONAL DATABASES IN SQL



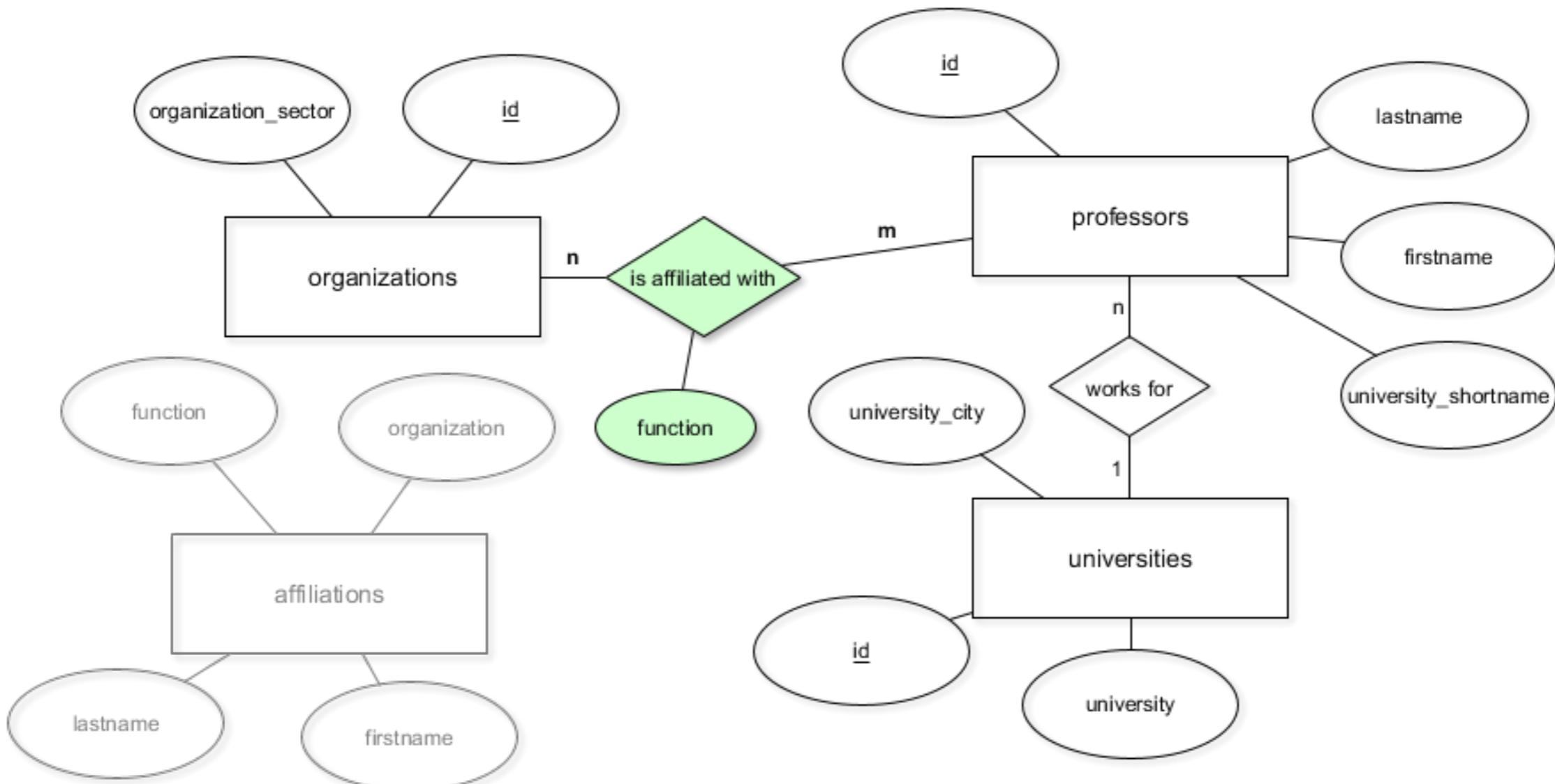
Timo Grossenbacher

Data Journalist

# The current database model



# The final database model



# How to implement N:M-relationships

- Create a table
- Add foreign keys for every connected table
- Add additional attributes

```
CREATE TABLE affiliations (
    professor_id integer REFERENCES professors (id),
    organization_id varchar(256) REFERENCES organizations (id),
    function varchar(256)
);
```

- No primary key!
- Possible PK = {professor\_id, organization\_id, function}

```
-- Set professor_id to professors.id where firstname, lastname correspond to rows in professors
UPDATE affiliations
SET professor_id = professors.id
FROM professors
WHERE affiliations.firstname = professors.firstname AND
      affiliations.lastname = professors.lastname;
```

Wow, that was a thing! As you can see, the correct professors.id has been inserted into professor\_id for each record, thanks to the matching firstname and lastname in both tables.

# **Time to implement this!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Referential integrity

INTRODUCTION TO RELATIONAL DATABASES IN SQL

A dark blue circular icon containing the white text "SQL".

Timo Grossenbacher

Data Journalist

# Referential integrity

- *A record referencing another table must refer to an existing record in that table*
- Specified between two tables
- Enforced through foreign keys

# Referential integrity violations

Referential integrity from table A to table B is violated...

- ...if a record in table B that is referenced from a record in table A is deleted.
- ...if a record in table A referencing a non-existing record from table B is inserted.
- Foreign keys prevent violations!

# Dealing with violations

```
CREATE TABLE a (
    id integer PRIMARY KEY,
    column_a varchar(64),
    ...,
    b_id integer REFERENCES b (id) ON DELETE NO ACTION
);
```

```
CREATE TABLE a (
    id integer PRIMARY KEY,
    column_a varchar(64),
    ...,
    b_id integer REFERENCES b (id) ON DELETE CASCADE
);
```

# Dealing with violations, contd.

ON DELETE...

- ...NO ACTION: Throw an error
- ...CASCADE: Delete all referencing records
- ...RESTRICT: Throw an error
- ...SET NULL: Set the referencing column to NULL
- ...SET DEFAULT: Set the referencing column to its default value

# Changing Constraints

query.sql

```
1 -- Identify the correct constraint name
2 SELECT constraint_name, table_name, constraint_type
3 FROM information_schema.table_constraints
4 WHERE constraint_type = 'FOREIGN KEY';
```



Run Code

Stop

query result

professors

affiliations

universities

organizations

constraint_name	table_name	constraint_type
affiliations_organization_id_fkey	affiliations	FOREIGN KEY
affiliations_professor_id_fkey	affiliations	FOREIGN KEY
professors_fkey	professors	FOREIGN KEY

-- Drop the right foreign key constraint

```
ALTER TABLE affiliations
DROP CONSTRAINT affiliations_organization_id_fkey;
```

-- Add a new foreign key constraint from affiliations to organizations which cascades deletion

```
ALTER TABLE affiliations
ADD CONSTRAINT affiliations_organization_id_fkey FOREIGN KEY (organization_id)
| | | | REFERENCES organizations (id) ON DELETE CASCADE;
```

# **Let's look at some examples!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

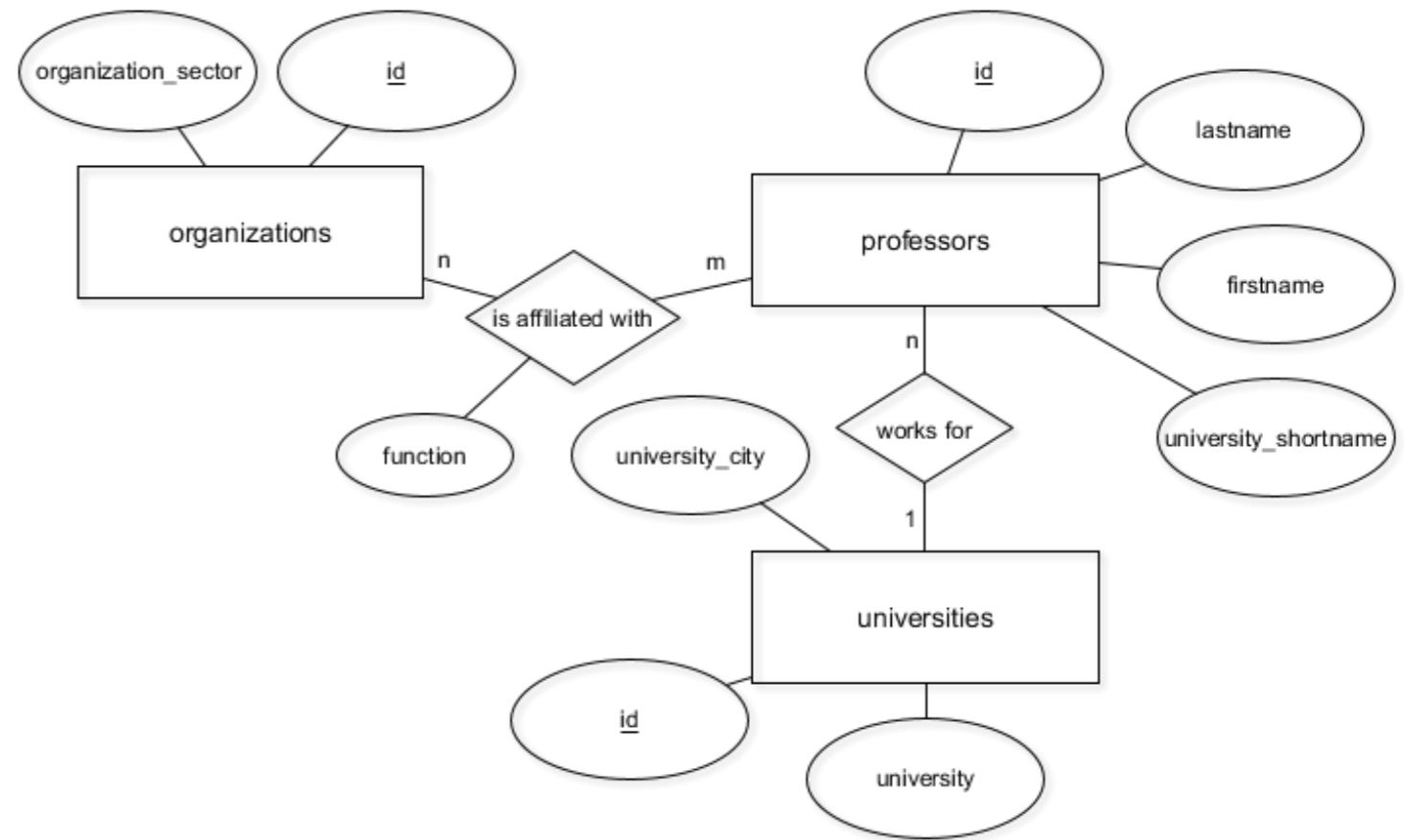
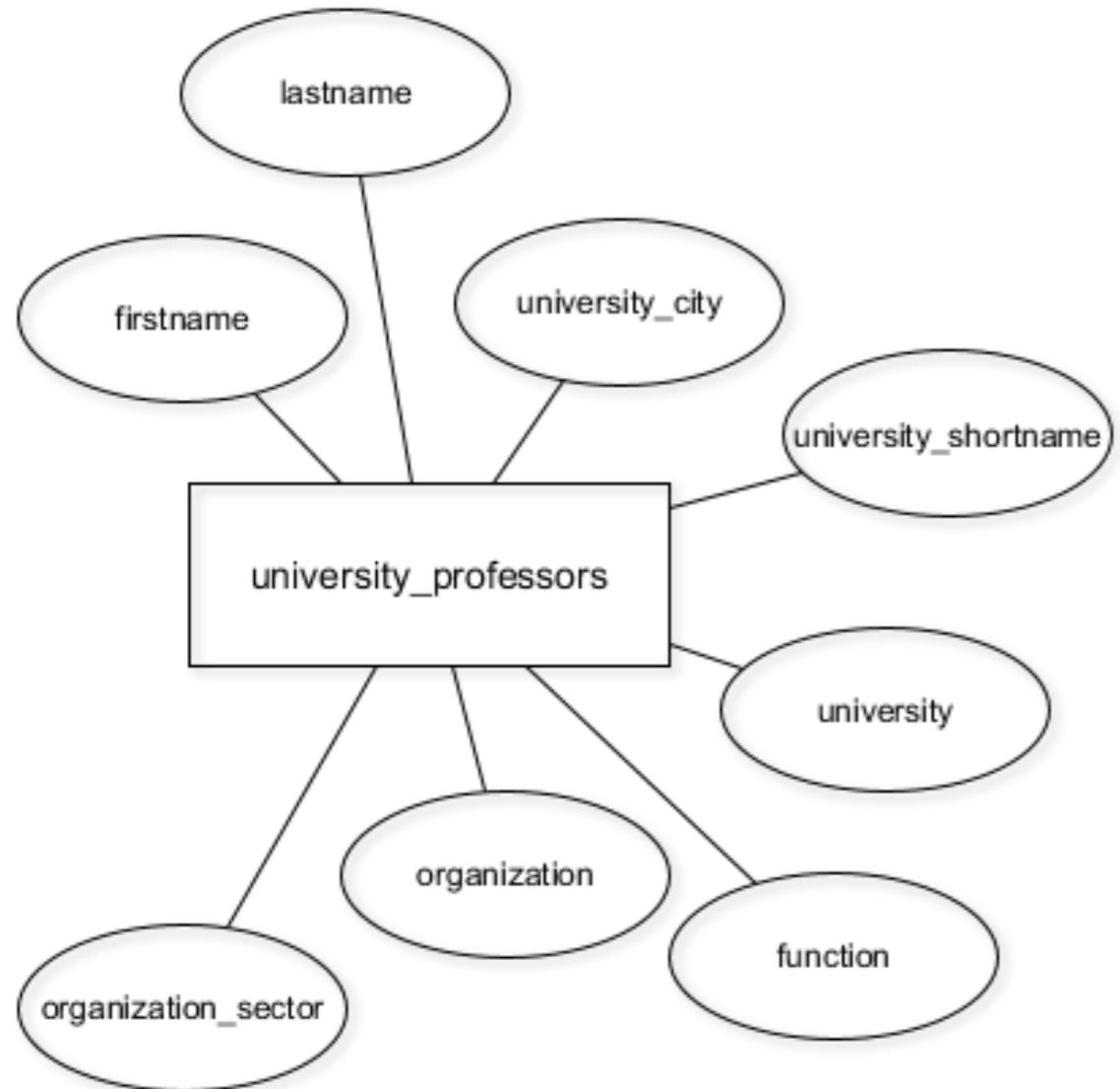
# Roundup

## INTRODUCTION TO RELATIONAL DATABASES IN SQL

SQL

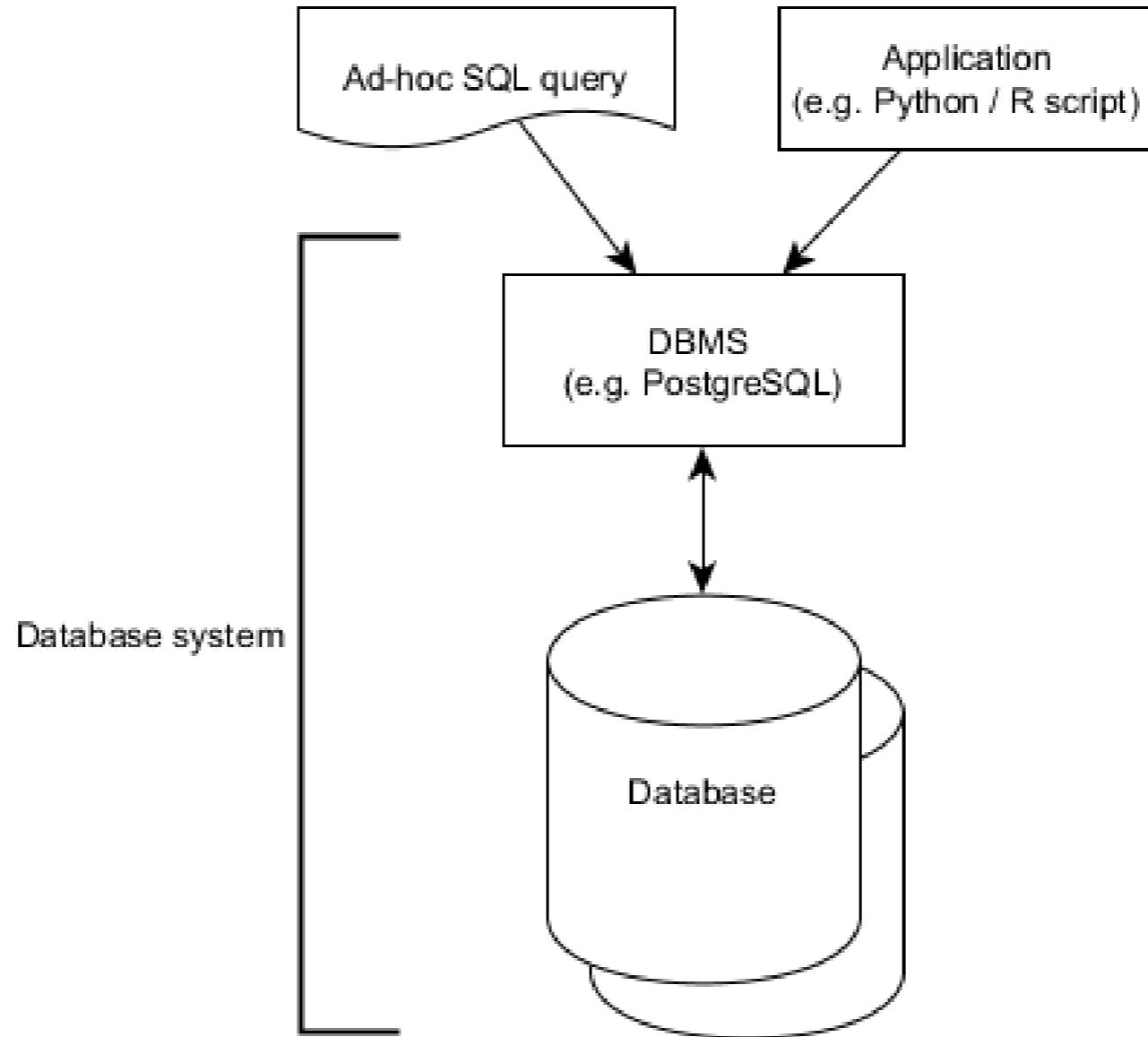
**Timo Grossenbacher**  
Data Journalist

# How you've transformed the database

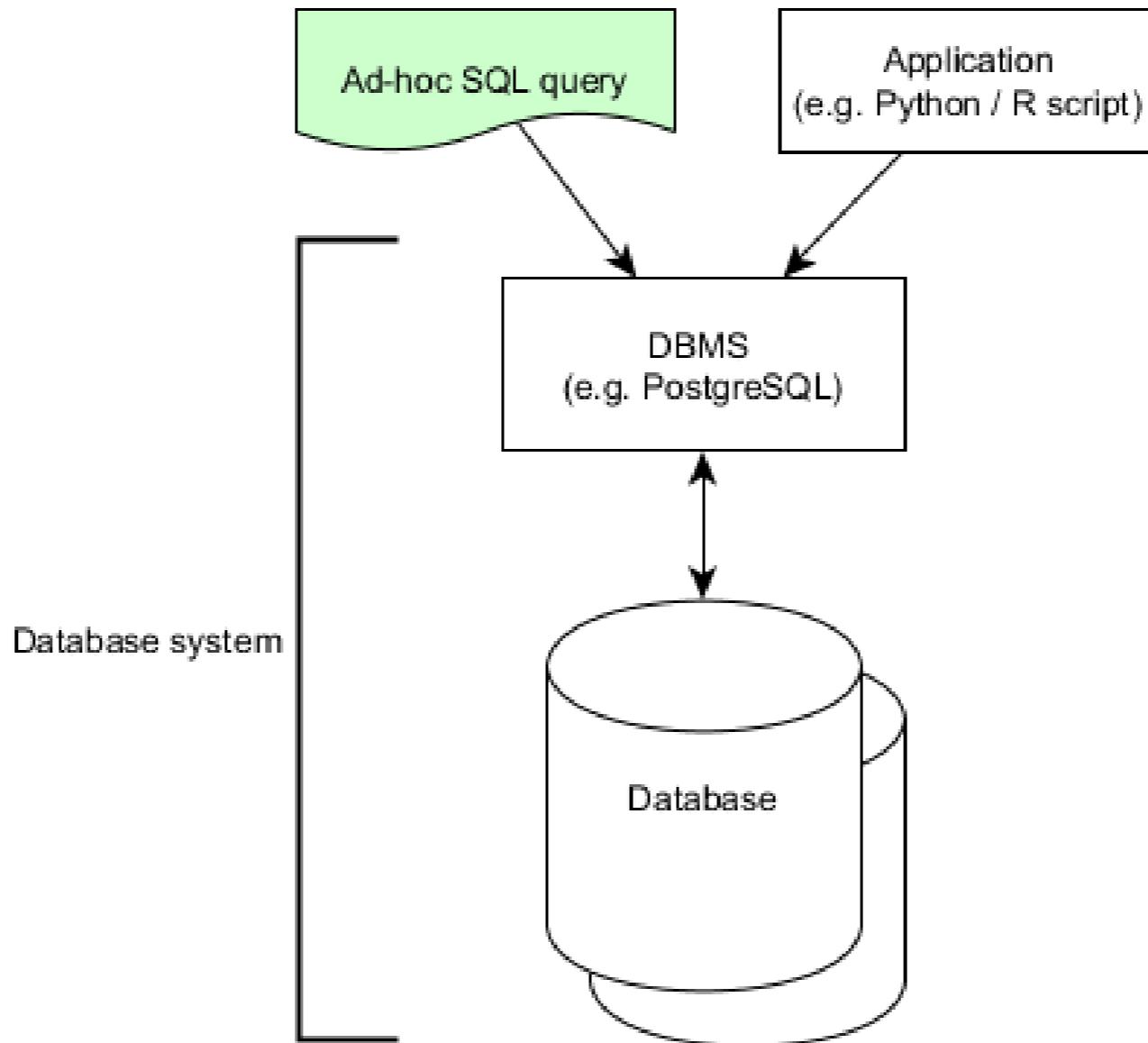


- Column data types
- Key constraints
- Relationships between tables

# The database ecosystem



# The database ecosystem



# **Thank you!**

**INTRODUCTION TO RELATIONAL DATABASES IN SQL**

# Welcome

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Course overview

- Chapter 1: Summarizing data
- Chapter 2: Date and math functions
- Chapter 3: Processing data with T-SQL
- Chapter 4: Window functions

# Exploring Data with Aggregation

- Reviewing summarized values for each column is a common first step in analyzing data
- If the data exists in a database, fastest way to aggregate is to use SQL

# Data Exploration with EconomicIndicators

```
SELECT Country, Year, InternetUse, GDP,  
       ExportGoodsPercent, CellPhonesper100  
FROM EconomicIndicators
```

Country	Year	InternetUse	GDP	ExportGoodsPercent	CellPhonesper100
Swaziland	2011	20.43165813	7335004354	56.30476059	63.7015615
Sweden	2011	90.88204559	394271163688	49.93022195	118.5711258
Switzerland	2011	82.98773087	395111518596	51.20242546	130.0623629
...					

# Common summary statistics

- `MIN()` for the minimum value of a column
- `MAX()` for the maximum value of a column
- `AVG()` for the mean or average value of a column

# Common summary statistics in T-SQL

This T-SQL query returns the aggregated values of column InternetUse

```
SELECT AVG(InternetUse) AS MeanInternetUse,  
MIN(InternetUse) AS MINInternet,  
MAX(InternetUse) AS MAXInternet  
FROM EconomicIndicators
```

MeanInternetUse	MINInternet	MAXInternet
18.9854496196171	0	375.5970064

# Filtering Summary Data with WHERE

This T-SQL query filters the aggregated values using a WHERE clause

Notice the text value is in

```
SELECT AVG(InternetUse) AS MeanInternetUse,  
MIN(InternetUse) AS MINInternet,  
MAX(InternetUse) AS MAXInternet  
FROM EconomicIndicators  
WHERE Country = 'Solomon Islands'
```

MeanInternetUse	MINInternet	MAXInternet
1.79621	0	6.00

# Subtotaling Aggregations into Groups with GROUP BY

```
SELECT Country, AVG(InternetUse) AS MeanInternetUse,  
MIN(InternetUse) AS MINInternet,  
MAX(InternetUse) AS MAXInternet  
FROM EconomicIndicators  
GROUP BY Country
```

Country	MeanInternetUse	MINInternet	MAXInternet
Solomon Islands	1.79621	0	6.00
Hong Kong	245.1067	0	375.00
Liechtenstein	63.8821	36.5152	85.00
...			

# HAVING is the WHERE for Aggregations

Cannot use WHERE with GROUP BY as it will give you an error

```
-- This throws an error  
...  
GROUP BY  
WHERE Max(InternetUse) > 100
```

Instead, use HAVING

```
-- This is how you filter with a GROUP BY  
...  
GROUP BY  
HAVING Max(InternetUse) > 100
```

# HAVING is the WHERE for Aggregations

```
SELECT Country, AVG(InternetUse) AS MeanInternetUse,  
MIN(GDP) AS SmallestGDP,  
MAX(InternetUse) AS MAXInternetUse  
FROM EconomicIndicators  
GROUP BY Country  
HAVING MAX(InternetUse) > 100
```

Country	MeanInternetUse	SmallestGDP	MAXInternetUse
Macedonia	71.3060150792857	-0.465059948	110.5679538
Hong Kong	245.106718614286	0	375.5970064
Congo	60.8972476010714	-9.492757847	104.6455529
...			

# Examining UFO Data in the Incidents Table

- The exercise will explore data gathered from Mutual UFO Network
- UFO spotted all over the world are contained in the Incidents Table

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Finding and Resolving Missing Data

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Detecting missing values

- When you have no data, the empty database field contains the word `NULL`
- Because `NULL` is not a number, it is not possible to use `=`, `<`, or `>` to find or compare missing values
- To determine if a column contains a `NULL` value, use `IS NULL` and `IS NOT NULL`

# Returning No NULL Values in T-SQL

```
SELECT Country, InternetUse, Year  
FROM EconomicIndicators  
WHERE InternetUse IS NOT NULL
```

Country	InternetUse	Year
Afghanistan	4.58066992	2011
Albania	49	2011
Algeria	14	2011
...		

# Detecting NULLs in T-SQL

```
SELECT Country, InternetUse, Year  
FROM EconomicIndicators  
WHERE InternetUse IS NULL
```

Country	InternetUse	Year
Angola	NULL	2013
Argentina	NULL	2013
Armenia	NULL	2013
...		

# Blank is not NULL

- A blank is not the same as a NULL value
- May show up in columns containing text
- An empty string '' can be used to find blank values
- The best way is to look for a column where the Length or LEN > 0

# Blank is not NULL

```
SELECT Country, GDP, Year  
FROM EconomicIndicators  
WHERE LEN(GDP) > 0
```

Country	GDP	Year
Afghanistan	54852215624	2011
Albania	29334492905	2011
Algeria	453558093404	2011
...		

# Substituting missing data with a specific value using ISNULL

```
SELECT GDP, Country,  
ISNULL(Country, 'Unknown') AS NewCountry  
FROM EconomicIndicators
```

GDP	Country	NewCountry
5867920022	NULL	Unknown
597873038497	South Africa	South Africa
1474091271101	NULL	Unknown
...		

# Substituting missing data with a column using ISNULL

```
/*Substituting values from one column for another with ISNULL*/
SELECT TradeGDPPercent, ImportGoodPercent,
ISNULL(TradeGDPPercent, ImportGoodPercent) AS NewPercent
FROM EconomicIndicators
```

TradeGDPPercent	ImportGoodPercent	NewPercent
NULL	56.7	56.7
52.18720739	51.75273421	52.18720739
NULL	NULL	NULL
...		

# Substituting NULL values using COALESCE

COALESCE returns the first non-missing value

```
COALESCE( value_1, value_2, value_3, ... value_n )
```

- If value\_1 is NULL and value\_2 is not NULL , return value\_2
- If value\_1 and value\_2 are NULL and value\_3 is not NULL , return value\_3
- ...

# SQL Statement using COALESCE

```
SELECT TradeGDPPercent, ImportGoodPercent,  
COALESCE(TradeGDPPercent, ImportGoodPercent, 'N/A') AS NewPercent  
FROM EconomicIndicators
```

TradeGDPPercent	ImportGoodPercent	NewPercent
NULL	56.7	56.7
NULL	NULL	N/A
52.18720739	51.75273421	52.18720739

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Binning Data with Case

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Changing column values with CASE

CASE

```
    WHEN Boolean_expression THEN result_expression [ ...n ]  
    [ ELSE else_result_expression ]
```

END

# Changing column values with CASE in T-SQL

```
SELECT Continent,  
CASE WHEN Continent = 'Europe' or Continent = 'Asia' THEN 'Eurasia'  
ELSE 'Other'  
END AS NewContinent  
FROM EconomicIndicators
```

Continent	NewContinent
Europe	Eurasia
Asia	Eurasia
Americas	Other
...	

# Changing column values with CASE in T-SQL

```
SELECT Continent,  
CASE WHEN Continent = 'Europe' or Continent = 'Asia' THEN 'Eurasia'  
ELSE Continent  
END AS NewContinent  
FROM EconomicIndicators
```

Continent	NewContinent
Europe	Eurasia
Asia	Eurasia
Americas	Americas
...	

# Using CASE statements to create value groups

```
-- We are binning the data here into discrete groups
SELECT Country, LifeExp,
CASE WHEN LifeExp < 30 THEN 1
    WHEN LifeExp > 29 AND LifeExp < 40 THEN 2
    WHEN LifeExp > 39 AND LifeExp < 50 THEN 3
    WHEN LifeExp > 49 AND LifeExp < 60 THEN 4
    ELSE 5
END AS LifeExpGroup
FROM EconomicIndicators
WHERE Year = 2007
```

LifeExp	LifeExpGroup
25	1
30	2
65	5
...	

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Counts and Totals

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Examining Totals with Counts

```
SELECT COUNT(*) FROM Incidents
```

```
+-----+  
| (No column name) |  
+-----+  
| 6452 |  
+-----+
```

# COUNT with DISTINCT

```
COUNT(DISTINCT COLUMN_NAME)
```

# COUNT with DISTINCT in T-SQL (I)

```
SELECT COUNT(DISTINCT Country) AS Countries  
FROM Incidents
```

Countries
3

# COUNT with DISTINCT in T-SQL (II)

```
SELECT COUNT(DISTINCT Country) AS Countries,  
COUNT(DISTINCT City) AS Cities  
FROM Incidents
```

Countries	Cities
3	3566

# COUNT AGGREGATION

- GROUP BY can be used with COUNT() in the same way as the other aggregation functions such as AVG(), MIN(), MAX()
- Use the ORDER BY command to sort the values
  - ASC will return the smallest values first (default)
  - DESC will return the largest values first

# COUNT with GROUP BY in T-SQL

```
-- Count the rows, subtotalized by Country  
SELECT COUNT(*) AS TotalRowsbyCountry, Country  
FROM Incidents  
GROUP BY Country
```

TotalRowsbyCountry	Country
5452	us
750	NULL
249	ca
1	gb

# COUNT with GROUP BY and ORDER BY in T-SQL (I)

```
-- Count the rows, subtotalized by Country  
SELECT COUNT(*) AS TotalRowsbyCountry, Country  
FROM Incidents  
GROUP BY Country  
ORDER BY Country ASC
```

TotalRowsbyCountry	Country
750	NULL
249	ca
1	gb
5452	us

# COUNT with GROUP BY and ORDER BY in T-SQL (II)

```
-- Count the rows, subtotalized by Country  
SELECT COUNT(*) AS TotalRowsbyCountry, Country  
FROM Incidents  
GROUP BY Country  
ORDER BY Country DESC
```

TotalRowsbyCountry	Country
5452	us
1	gb
249	ca
750	NULL

# Column totals with SUM

- `SUM()` provides a numeric total of the values in a column
- It follows the same pattern as other aggregations
- Combine it with GROUP BY to get subtotals based on columns specified

# Adding column values in T-SQL

```
-- Calculate the values subtotalized by Country  
SELECT SUM(DurationSeconds) AS TotalDuration, Country  
FROM Incidents  
GROUP BY Country
```

Country	TotalDuration
us	17024946.750001565
null	18859192.800000012
ca	200975
gb	120

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Math with Dates

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# DATEPART

`DATEPART` is used to determine what part of the date you want to calculate. Some of the common abbreviations are:

- `DD` for Day
- `MM` for Month
- `YY` for Year
- `HH` for Hour

# Common date functions in T-SQL

- `DATEADD()` : Add or subtract datetime values
  - Always returns a date
- `DATEDIFF()` : Obtain the difference between two datetime values
  - Always returns a number

# DATEADD

To Add or subtract a value to get a new date use `DATEADD()`

`DATEADD (DATEPART, number, date)`

- `DATEPART` : Unit of measurement (DD, MM etc.)
- `number` : An integer value to add
- `date` : A datetime value

# Date math with DATEADD (I)

*What date is 30 days from June 21, 2020?*

```
SELECT DATEADD(DD, 30, '2020-06-21')
```

(No Column Name)
2020-07-21 00:00

# Date math with DATEADD (II)

*What date is 30 days before June 21, 2020?*

```
SELECT DATEADD(DD, -30, '2020-06-21')
```

(No Column Name)
2020-05-22 00:00

# DATEDIFF

Returns a date after a number has been added or subtracted to a date

```
DATEDIFF (datepart, startdate, enddate)
```

- **datepart** : Unit of measurement (DD, MM etc.)
- **startdate** : The starting date value
- **enddate** : An ending datetime value

# Date math with DATEDIFF

```
SELECT DATEDIFF(DD, '2020-05-22', '2020-06-21') AS Difference1,  
       DATEDIFF(DD, '2020-07-21', '2020-06-21') AS Difference2
```

Difference1	Difference2
30	-30

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Rounding and Truncating numbers

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Rounding numbers in T-SQL

```
ROUND(number, length [,function])
```

# Rounding numbers in T-SQL

```
SELECT DurationSeconds,  
ROUND(DurationSeconds, 0) AS RoundToZero,  
ROUND(DurationSeconds, 1) AS RoundToOne  
FROM Incidents
```

DurationSeconds	RoundToZero	RoundToOne
121.6480	122.0000	121.6000
170.3976	170.0000	170.4000
336.0652	336.0000	336.1000
...		

# Rounding on the left side of the decimal

```
SELECT DurationSeconds,  
ROUND(DurationSeconds, -1) AS RoundToTen,  
ROUND(DurationSeconds, -2) AS RoundToHundred  
FROM Incidents
```

DurationSeconds	RoundToTen	RoundToHundred
121.6480	120.0000	100.0000
170.3976	170.0000	200.0000
336.0652	340.0000	300.0000
...		

# Truncating numbers

**TRUNCATE**

17.85 → 17

**ROUND**

17.85 → 18

# Truncating with ROUND()

The `ROUND()` function can be used to truncate values when you specify the third argument

```
ROUND(number, length [,function])
```

- Set the third value to a non-zero number

# Truncating in T-SQL

```
SELECT Profit,  
ROUND(DurationSeconds, 0) AS RoundingtoWhole,  
ROUND(DurationSeconds, 0, 1) AS Truncating  
FROM Incidents
```

Profit	RoundingtoWhole	Truncating
15.6100	16.0000	15.0000
13.2444	13.0000	13.0000
17.9260	18.0000	17.0000
...		

Truncating just cuts all numbers off after the specified digit

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# More math functions

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Absolute value

Use ABS() to return non-negative values

ABS(number)

# Using ABS in T-SQL (I)

```
SELECT ABS(-2.77), ABS(3), ABS(-2)
```

(No column name)	(No column name)	(No column name)
2.77	3	2

# Using ABS in T-SQL (II)

```
SELECT DurationSeconds, ABS(DurationSeconds) AS AbsSeconds  
FROM Incidents
```

DurationSeconds	AbsSeconds
-25.36	25.36
-258482.44	258482.44
45.66	45.66

# Squares and square roots in T-SQL

```
SELECT SQRT(9) AS Sqrt,  
       SQUARE(9) AS Square
```

Sqrt	Square
3	81

# Logs

- `LOG()` returns the natural logarithm
- Optionally, you can set the base, which if not set is 2.718281828

```
LOG(number [,Base])
```

# Calculating logs in T-SQL

```
SELECT DurationSeconds, LOG(DurationSeconds, 10) AS LogSeconds  
FROM Incidents
```

DurationSeconds	LogSeconds
37800	4.577491799837225
5	0.6989700043360187
20	1.301029995663981
...	

# Log of 0

You cannot take the log of 0 as it will give you an error

```
SELECT LOG(0, 10)
```

An invalid floating point operation occurred.

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# **WHILE loops**

## INTERMEDIATE SQL SERVER



**Ginger Grant**  
Instructor

# Using variables in T-SQL

- Variables are needed to set values `DECLARE @variablename data_type`
  - Must start with the character @

# Variable data types in T-SQL

- `VARCHAR(n)` : variable length text field
- `INT` : integer values from -2,147,483,647 to +2,147,483,647
- `DECIMAL(p ,s)` or `NUMERIC(p ,s)` :
  - `p` : total number of decimal digits that will be stored, both to the left and to the right of the decimal point
  - `s` : number of decimal digits that will be stored to the right of the decimal point

# Declaring variables in T-SQL

```
-- Declare Snack as a VARCHAR with length 10  
DECLARE @Snack VARCHAR(10)
```

# Assigning values to variables

```
-- Declare the variable  
DECLARE @Snack VARCHAR(10)  
-- Use SET a value to the variable  
SET @Snack = 'Cookies'  
-- Show the value  
SELECT @Snack
```

```
+-----+  
|(No column name)|  
+-----+  
|Cookies|  
+-----+
```

```
-- Declare the variable  
DECLARE @Snack VARCHAR(10)  
-- Use SELECT assign a value  
SELECT @Snack = 'Candy'  
-- Show the value  
SELECT @Snack
```

```
+-----+  
|(No column name)|  
+-----+  
|Candy|  
+-----+
```

# WHILE loops

- WHILE evaluates a true or false condition
- After the WHILE, there should be a line with the keyword BEGIN
- Next include code to run until the condition in the WHILE loop is true
- After the code add the keyword END
- BREAK will cause an exit out of the loop
- CONTINUE will cause the loop to continue

# WHILE loop in T-SQL (I)

```
-- Declare ctr as an integer
DECLARE @ctr INT
-- Assign 1 to ctr
SET @ctr = 1
-- Specify the condition of the WHILE loop
WHILE @ctr < 10
    -- Begin the code to execute inside WHILE loop
    BEGIN
        -- Keep incrementing the value of @ctr
        SET @ctr = @ctr + 1
        -- End WHILE loop
    END
-- View the value after the loop
SELECT @ctr
```

(No column name)
10

# WHILE loop in T-SQL (II)

```
-- Declare ctr as an integer
DECLARE @ctr INT
-- Assign 1 to ctr
SET @ctr = 1
-- Specify the condition of the WHILE loop
WHILE @ctr < 10
    -- Begin the code to execute inside WHILE loop
    BEGIN
        -- Keep incrementing the value of @ctr
        SET @ctr = @ctr + 1
```

```
-- Check if ctr is equal to 4
IF @ctr = 4
    -- When ctr is equal to 4, the loop will break
    BREAK
-- End WHILE loop
END
```

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Derived tables

INTERMEDIATE SQL SERVER

A dark blue circular icon containing the white text "SQL".

Ginger Grant  
Instructor

# What are Derived tables?

- Query which is treated like a temporary table
- Always contained within the main query
- They are specified in the `FROM` clause
- Can contain intermediate calculations to be used in the main query or different joins than in the main query

# Derived tables in T-SQL

```
SELECT a.* FROM Kidney a  
-- This derived table computes the Average age joined to the actual table  
JOIN (SELECT AVG(Age) AS AverageAge  
      FROM Kidney) b  
ON a.Age = b.AverageAge
```

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Common Table Expressions

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# CTE syntax

```
-- CTE definitions start with the keyword WITH
-- Followed by the CTE names and the columns it contains
WITH CTENName (Col1, Col2)
AS
-- Define the CTE query
(
-- The two columns from the definition above
    SELECT Col1, Col2
    FROM TableName
)
```

# CTEs in T-SQL

```
-- Create a CTE to get the Maximum BloodPressure by Age
```

```
WITH BloodPressureAge(Age, MaxBloodPressure)
```

```
AS
```

```
(SELECT Age, MAX(BloodPressure) AS MaxBloodPressure
```

```
FROM Kidney
```

```
GROUP BY Age)
```

```
-- Create a query to use the CTE as a table
```

```
SELECT a.Age, MIN(a.BloodPressure), b.MaxBloodPressure
```

```
FROM Kidney a
```

```
-- Join the CTE with the table
```

```
JOIN BloodpressureAge b
```

```
ON a.Age = b.Age
```

```
GROUP BY a.Age, b.MaxBloodPressure
```

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Window functions

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

	SalesPerson	SalesYear	CurrentQuota	ModifiedDate
1	Bob	2011	28000.00	2011-04-16
2	Bob	2011	7000.00	2011-07-17
3	Bob	2011	91000.00	2011-10-17
4	Mary	2011	367000.00	2011-04-16
5	Mary	2011	556000.00	2011-07-17
6	Mary	2011	502000.00	2011-10-17
7	Bob	2012	140000.00	2012-01-15
8	Bob	2012	70000.00	2012-04-15

# Grouping data in T-SQL

```
SELECT SalesPerson, SalesYear,  
       CurrentQuota, ModifiedDate  
FROM SaleGoal  
WHERE SalesYear = 2011
```

SalesPerson	SalesYear	CurrentQuota	ModifiedDate
Bob	2011	28000.00	2011-04-16
Bob	2011	7000.00	2011-07-16
Bob	2011	91000.00	2011-10-16
Mary	2011	367000.00	2011-04-16
Mary	2011	556000.00	2011-07-16
Mary	2011	502000.00	2011-10-16

# Window syntax in T-SQL

- Create the window with `OVER` clause
- `PARTITION BY` creates the frame
- If you do not include `PARTITION BY` the frame is the entire table
- To arrange the results, use `ORDER BY`
- Allows aggregations to be created at the same time as the window

• • •

```
-- Create a Window data grouping  
OVER (PARTITION BY SalesYear ORDER BY SalesYear)
```

# Window functions (SUM)

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       SUM(CurrentQuota)  
   OVER (PARTITION BY SalesYear) AS YearlyTotal,  
     ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	YearlyTotal	ModDate
Bob	2011	28000.00	1551000.00	2011-04-16
Bob	2011	7000.00	1551000.00	2011-07-17
Mary	2011	367000.00	1551000.00	2011-04-16
Mary	2011	556000.00	1551000.00	2011-07-15
Bob	2012	70000.00	1859000.00	2012-01-15
Bob	2012	154000.00	1859000.00	2012-04-16
Bob	2012	107000.00	1859000.00	2012-07-16
...				

# Window functions (COUNT)

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       COUNT(CurrentQuota)  
OVER (PARTITION BY SalesYear) AS QuotaPerYear,  
     ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	QuotaPerYear	ModDate
Bob	2011	28000.00	4	2011-04-16
Bob	2011	7000.00	4	2011-07-17
Mary	2011	367000.00	4	2011-04-16
Mary	2011	556000.00	4	2011-07-15
Bob	2012	70000.00	8	2012-01-15
Bob	2012	154000.00	8	2012-04-15
Bob	2012	107000.00	8	2012-10-16
...				

- Notice the count starts over for each window in column **QuotaPerYear**

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Common window functions

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# FIRST\_VALUE() and LAST\_VALUE()

- FIRST\_VALUE() returns the first value in the window
- LAST\_VALUE() returns the last value in the window

	SalesPerson	SalesYear	CurrentQuota	ModifiedDate
1	Bob	2011	28000.00	2011-04-16 00:00:00.000
2	Bob	2011	7000.00	2011-07-17 00:00:00.000
3	Bob	2011	91000.00	2011-10-17 00:00:00.000
4	Bob	2012	140000.00	2012-01-15 00:00:00.000
5	Bob	2012	70000.00	2012-04-15 00:00:00.000
6	Bob	2012	154000.00	2012-07-16 00:00:00.000
7	Bob	2012	107000.00	2012-10-16 00:00:00.000
8	Mary	2011	367000.00	2011-04-16 00:00:00.000
9	Mary	2011	556000.00	2011-07-17 00:00:00.000
10	Mary	2011	502000.00	2011-10-17 00:00:00.000

# FIRST\_VALUE() and LAST\_VALUE() in T-SQL

- Note that for FIRST\_VALUE and LAST\_VALUE the ORDER BY command is required

```
-- Select the columns
SELECT SalesPerson, SalesYear, CurrentQuota,
    -- First value from every window
    FIRST_VALUE(CurrentQuota)
        OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS StartQuota,
    -- Last value from every window
    LAST_VALUE(CurrentQuota)
        OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS EndQuota,
    ModifiedDate as ModDate
FROM SaleGoal
```

# Results

SalesPerson	SalesYear	CurrentQuota	StartQuota	EndQuota	ModDate
Bob	2011	28000.00	28000.00	91000.00	2011-04-16
Bob	2011	7000.00	28000.00	91000.00	2011-07-17
Bob	2011	91000.00	28000.00	91000.00	2011-10-17
Bob	2012	140000.00	140000.00	107000.00	2012-01-15
Bob	2012	70000.00	140000.00	107000.00	2012-04-15
Bob	2012	154000.00	140000.00	107000.00	2012-07-16
Bob	2012	107000.00	140000.00	107000.00	2012-10-16
...					

# Getting the next value with LEAD()

- Provides the ability to query the value from the next row
- NextQuota column is created by using `LEAD()`
- Requires the use of `ORDER BY` to order the rows

	SalesPerson	SalesYear	CurrentQuota	NextQuota	ModDate
1	Bob	2011	28000.00	367000.00	2011-04-15
2	Mary	2011	367000.00	556000.00	2011-04-16
3	Mary	2011	556000.00	7000.00	2011-07-15
4	Bob	2011	7000.00	NULL	2011-07-17
5	Bob	2012	70000.00	502000.00	2012-01-15

# LEAD() in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
-- Create a window function to get the values from the next row  
    LEAD(CurrentQuota)  
    OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS NextQuota,  
    ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	NextQuota	ModDate
Bob	2011	28000.00	367000.00	2011-04-15
Mary	2011	367000.00	556000.00	2011-04-16
Mary	2011	556000.00	7000.00	2011-07-15
Bob	2011	7000.00	NULL	2011-07-17
Bob	2012	70000.00	502000.00	2012-01-15
Mary	2012	502000.00	154000.00	2012-01-16
...				

# Getting the previous value with LAG()

- Provides the ability to query the value from the previous row
- PreviousQuota column is created by using LAG()
- Requires the use of ORDER BY to order the rows

	SalesPerson	SalesYear	CurrentQuota	PreviousQuota	ModDate
1	Bob	2011	28000.00	NULL	2011-04-15
2	Mary	2011	367000.00	28000.00	2011-04-16
3	Mary	2011	556000.00	367000.00	2011-07-15
4	Bob	2011	7000.00	556000.00	2011-07-17
5	Bob	2012	70000.00	NULL	2012-01-15
6	Mary	2012	502000.00	70000.00	2012-01-15

# LAG() in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
-- Create a window function to get the values from the previous row  
    LAG(CurrentQuota)  
        OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS PreviousQuota,  
    ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	PreviousQuota	ModDate
Bob	2011	28000.00	NULL	2011-04-15
Mary	2011	367000.00	28000.00	2011-04-16
Mary	2011	556000.00	367000.00	2011-07-15
Bob	2011	7000.00.00	556000.00	2011-07-17
Bob	2012	7000.00	NULL	2012-01-15
Mary	2012	502000.00	7000.00	2012-01-16
...				

# **Let's practice !**

## **INTERMEDIATE SQL SERVER**

## First value in a window

Suppose you want to figure out the first `OrderDate` in each territory or the last one.

How would you do that? You can use the window functions `FIRST_VALUE()` and `LAST_VALUE()`, respectively! Here are the steps:

- First, create partitions for each territory
- Then, order by `OrderDate`
- Finally, use the `FIRST_VALUE()` and/or `LAST_VALUE()` functions as per your requirement

```
SELECT TerritoryName, OrderDate,
    -- Select the first value in each partition
    FIRST_VALUE(OrderDate)
    -- Create the partitions and arrange the rows
    OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS FirstOrder
FROM Orders
```

# Increasing window complexity

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Reviewing aggregations

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       SUM(CurrentQuota)  
  OVER (PARTITION BY SalesYear) AS YearlyTotal,  
        ModifiedDate as ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	YearlyTotal	ModDate
Bob	2011	28000.00	1551000.00	2011-04-16
Bob	2011	7000.00	1551000.00	2011-07-17
Bob	2011	91000.00	1551000.00	2011-10-17
Mary	2011	140000.00	1551000.00	2012-04-15
Mary	2011	70000.00	1551000.00	2012-07-15
Mary	2011	154000.00	1551000.00	2012-01-15
Mary	2012	107000.00	1859000.00	2012-01-16
...				

# Adding ORDER BY to an aggregation

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       SUM(CurrentQuota)  
  OVER (PARTITION BY SalesYear ORDER BY SalesPerson) AS YearlyTotal,  
     ModifiedDate as ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	YearTotal	ModDate
Bob	2011	28000.00	35000.00	2011-04-16
Bob	2011	7000.00	35000.00	2011-07-17
Mary	2011	367000.00	958000.00	2011-10-17
Mary	2011	556000.00	958000.00	2012-04-15
Bob	2012	70000.00	401000.00	2012-07-15
Bob	2012	154000.00	401000.00	2012-10-16
...				

# Creating a running total with ORDER BY

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       SUM(CurrentQuota)  
   OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) as RunningTotal,  
         ModifiedDate as ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	RunningTotal	ModDate
Bob	2011	28000.00	28000.00	2011-04-16
Mary	2011	367000.00	395000.00	2011-07-17
Mary	2011	556000.00	951000.00	2011-10-17
Bob	2011	7000.00	958000.00	2012-04-15
Bob	2012	70000.00	70000.00	2012-01-15
Mary	2012	502000.00	572000.00	2012-01-16
...				

# Adding row numbers

- `ROW_NUMBER()` sequentially numbers the rows in the window
- `ORDER BY` is required when using `ROW_NUMBER()`

	SalesPerson	SalesYear	CurrentQuota	QuotaBySalesPerson
1	Bob	2011	28000.00	1
2	Bob	2011	7000.00	2
3	Bob	2012	70000.00	3
4	Bob	2012	154000.00	4
5	Bob	2012	70000.00	5
6	Bob	2012	107000.00	6
7	Bob	2013	91000.00	7
8	Mary	2011	367000.00	1
9	Mary	2011	556000.00	2

# Adding row numbers in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       ROW_NUMBER()  
             OVER (PARTITION BY SalesPerson ORDER BY SalesYear) AS QuotabySalesPerson  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	QuotabySalesPerson
Bob	2011	28000.00	1
Bob	2011	7000.00	2
Bob	2011	70000.00	3
Bob	2011	154000.00	4
Bob	2012	70000.00	5
Bob	2012	107000.00	6
Bob	2012	91000.00	7
Mary	2011	367000.00	1
...			

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Using windows for calculating statistics

INTERMEDIATE SQL SERVER



Ginger Grant  
Instructor

# Calculating the standard deviation

- Calculate standard deviation either for the entire table or for each window
- `STDEV()` calculates the standard deviation

# Calculating the standard deviation for the entire table

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       STDEV(CurrentQuota)  
     OVER () AS StandardDev,  
      ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	StandardDev	ModDate
Bob	2011	28000.00	267841.370964233	2011-04-16
Bob	2011	7000.00	267841.370964233	2011-07-17
Bob	2011	91000.00	267841.370964233	2011-10-17
Bob	2012	140000.00	267841.370964233	2012-01-15
Bob	2012	70000.00	267841.370964233	2012-04-15
...				

# Calculating the standard deviation for each partition

```
SELECT SalesPerson, SalesYear, CurrentQuota,  
       STDEV(CurrentQuota)  
  OVER (PARTITION BY SalesYear ORDER BY SalesYear) AS StDev,  
        ModifiedDate AS ModDate  
FROM SaleGoal
```

SalesPerson	SalesYear	CurrentQuota	StDev	ModDate
Bob	2011	28000.00	267841.54080	2011-04-16
Bob	2011	7000.00	267841.54080	2011-07-17
Mary	2011	91000.00	267841.54080	2011-04-16
Mary	2011	140000.00	267841.54080	2011-07-15
Bob	2012	70000.00	246538.86248	2012-01-15
Bob	2012	154000.00	246538.86248	2012-04-15
Bob	2012	107000.00	246538.86248	2012-07-16
...				

# Calculating the mode

- Mode is the value which appears the most often in your data
- To calculate mode:
  - Create a CTE containing an ordered count of values using ROW\_NUMBER
  - Write a query using the CTE to pick the value with the highest row number

# Calculating the mode in T-SQL (I)

```
WITH QuotaCount AS (
    SELECT SalesPerson, SalesYear, CurrentQuota,
           ROW_NUMBER()
      OVER (PARTITION BY CurrentQuota ORDER BY CurrentQuota) AS QuotaList
   FROM SaleGoal
)
SELECT * FROM QuotaCount
```

SalesPerson	SalesYear	CurrentQuota	QuotaList	
Bob	2011	7000.00	1	
Bob	2011	28000.00	1	
Bob	2011	70000.00	1	
Bob	2012	70000.00	2	
Mary	2012	73000.00	1	
...				

- Notice there are two values for 70,000.00

# Calculating the mode in T-SQL (II)

```
WITH QuotaCount AS (
    SELECT SalesPerson, SalesYear, CurrentQuota,
        ROW_NUMBER()
            OVER (PARTITION BY CurrentQuota ORDER BY CurrentQuota) AS QuotaList
    FROM SaleGoal
)
SELECT CurrentQuota, QuotaList AS Mode
FROM QuotaCount
WHERE QuotaList IN (SELECT MAX(QuotaList) FROM QuotaCount)
```

CurrentQuota	Mode
70000.00	2

# **Let's practice!**

## **INTERMEDIATE SQL SERVER**

# Building dates

TIME SERIES ANALYSIS IN SQL SERVER



**Kevin Feasel**  
CTO, Envizage



# What you will learn

- Working with component date parts
- Translating strings to dates, including date-time offsets and invalid dates
- Filtering, grouping, and aggregating data by time periods
- Upsampling and downsampling data
- Aggregations over windows
- Calculating running totals and moving averages
- Finding overlap in date ranges



(Photo by [Aron Visuals](#))

# Building a date

```
SELECT
```

```
    GETDATE() AS DateTime_LTz,  
    GETUTCDATE() AS DateTime_UTC;
```

```
SELECT
```

```
    SYSDATETIME() AS DateTime2_LTz  
    SYSUTCDATETIME() AS DateTime2_UTC;
```

Results:

DateTime_LTz	DateTime_UTC	DateTime2_LTz	DateTime2_UTC
2019-03-07 21:21:33.670	2019-03-08 02:21:33.670	2019-03-07 21:21:33.6716402	2019-03-08 02:21:33.6716402

# Breaking down a date

```
DECLARE
```

```
    @SomeDate DATETIME2(3) = '2019-03-01 08:17:19.332';
```

```
SELECT YEAR(@SomeDate);
```

```
SELECT MONTH(@SomeDate);
```

```
SELECT DAY(@SomeDate);
```

YEAR = 2019

MONTH = 3

DAY = 1

# Parsing dates with date parts

## Functions

**DATEPART()**

**SELECT**

```
DATEPART(YEAR, @dt) AS TheYear;
```

**DATENAME()**

**SELECT**

```
DATENAME(MONTH, @dt) AS TheMonth;
```

## Parts

- Year / Month / Day
- Day of year
- Day of week
- Week of year
- ISO week of year
- Minute / Second
- Millisecond / Nanosecond

# Adding and subtracting dates

DECLARE

```
@SomeTime DATETIME2(7) = '1992-07-14 14:49:36.2294852';
```

SELECT

```
DATEADD(DAY, 1, @SomeTime) AS NextDay,  
DATEADD(DAY, -1, @SomeTime) AS PriorDay;
```

SELECT

```
DATEADD(HOUR, -3, DATEADD(DAY, -4, @SomeTime)) AS Minus4Days3Hours;
```

NextDay	PriorDay
1992-07-15 14:49:36.2294852	1992-07-13 14:49:36.2294852
Minus4Days3Hours	
1992-07-10 11:49:36.2294852	

# Comparing dates

DECLARE

```
@StartTime DATETIME2(7) = '2012-03-01 14:29:36',  
@EndTime DATETIME2(7) = '2012-03-01 18:00:00';
```

SELECT

```
DATEDIFF(SECOND, @StartTime, @EndTime) AS SecondsElapsed,  
DATEDIFF(MINUTE, @StartTime, @EndTime) AS MinutesElapsed,  
DATEDIFF(HOUR, @StartTime, @EndTime) AS HoursElapsed;
```

SecondsElapsed	MinutesElapsed	HoursElapsed
12624	211	4

## Rounding dates

SQL Server does not have an intuitive way to round down to the month, hour, or minute. You can, however, combine the `DATEADD()` and `DATEDIFF()` functions to perform this rounding.

To round the date 1914-08-16 down to the year, we would call

`DATEADD(YEAR, DATEDIFF(YEAR, 0, '1914-08-16'), 0)`. To round that date down to the month, we would call

`DATEADD(MONTH, DATEDIFF(MONTH, 0, '1914-08-16'), 0)`. This works for several other date parts as well.

# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Formatting dates for reporting

TIME SERIES ANALYSIS IN SQL SERVER

SQL

**Kevin Feasel**  
CTO, Envizage

# Formatting functions

`CAST()`

`CONVERT()`

`FORMAT()`

# The CAST() function

- Supported going back at least to SQL Server 2000
- Useful for converting one data type to another data type, including date types
- No control over formatting from dates to strings
- ANSI SQL standard, meaning any relational and most non-relational databases have this function

# Using the CAST() function

## DECLARE

```
@SomeDate DATETIME2(3) = '1991-06-04 08:00:09',  
@SomeString NVARCHAR(30) = '1991-06-04 08:00:09',  
@OldDateTime DATETIME = '1991-06-04 08:00:09';
```

## SELECT

```
CAST(@SomeDate AS NVARCHAR(30)) AS DateToString,  
CAST(@SomeString AS DATETIME2(3)) AS StringToDate,  
CAST(@OldDateTime AS NVARCHAR(30)) AS OldDateToString;
```

DateToString	StringToDate	OldDateToString
1991-06-04 08:00:09.000	1991-06-04 08:00:09.000	Jun 4 1991 8:00AM

# The CONVERT() function

- Supported going back at least to SQL Server 2000
- Useful for converting one data type to another data type, including date types
- Some control over formatting from dates to strings using the style parameter
- Specific to T-SQL

# Using the CONVERT() function

DECLARE

```
@SomeDate DATETIME2(3) = '1793-02-21 11:13:19.033';
```

SELECT

```
CONVERT(NVARCHAR(30), @SomeDate, 0) AS DefaultForm,  
CONVERT(NVARCHAR(30), @SomeDate, 1) AS US_mdy,  
CONVERT(NVARCHAR(30), @SomeDate, 101) AS US_mdyyyy,  
CONVERT(NVARCHAR(30), @SomeDate, 120) AS ODBC_sec;
```

GO

DefaultForm	US_mdy	US_mdyyyy	ODBC_sec
Feb 21 1793 11:13 AM	02/21/93	02/21/1793	1793-02-21 11:13:19

<https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql?view=sql-server-ver15#date-and-time-styles>

# Sample CONVERT() styles

## Style Code

- 1 / 101
- 3 / 103
- 4 / 104
- 11 / 111
- 12 / 112
- 20 / 120
- 126
- 127

## Format

- United States m/d/y
- British/French d/m/y
- German d.m.y
- Japanese y/m/d
- ISO standard yyyyymmdd
- ODBC standard (121 for ms)
- ISO8601 yyyy-mm-dd hh:mi:ss.mmm
- yyyy-mm-ddThh:mi:ss.mmmZ

# The FORMAT() function

- Supported as of SQL Server 2012
- Useful for formatting a date or number in a particular way for reporting
- Much more flexibility over formatting from dates to strings than either `CAST()` or `CONVERT()`
- Specific to T-SQL
- Uses the .NET framework for conversion
- Can be slower as you process more rows

# Using the FORMAT() function

## DECLARE

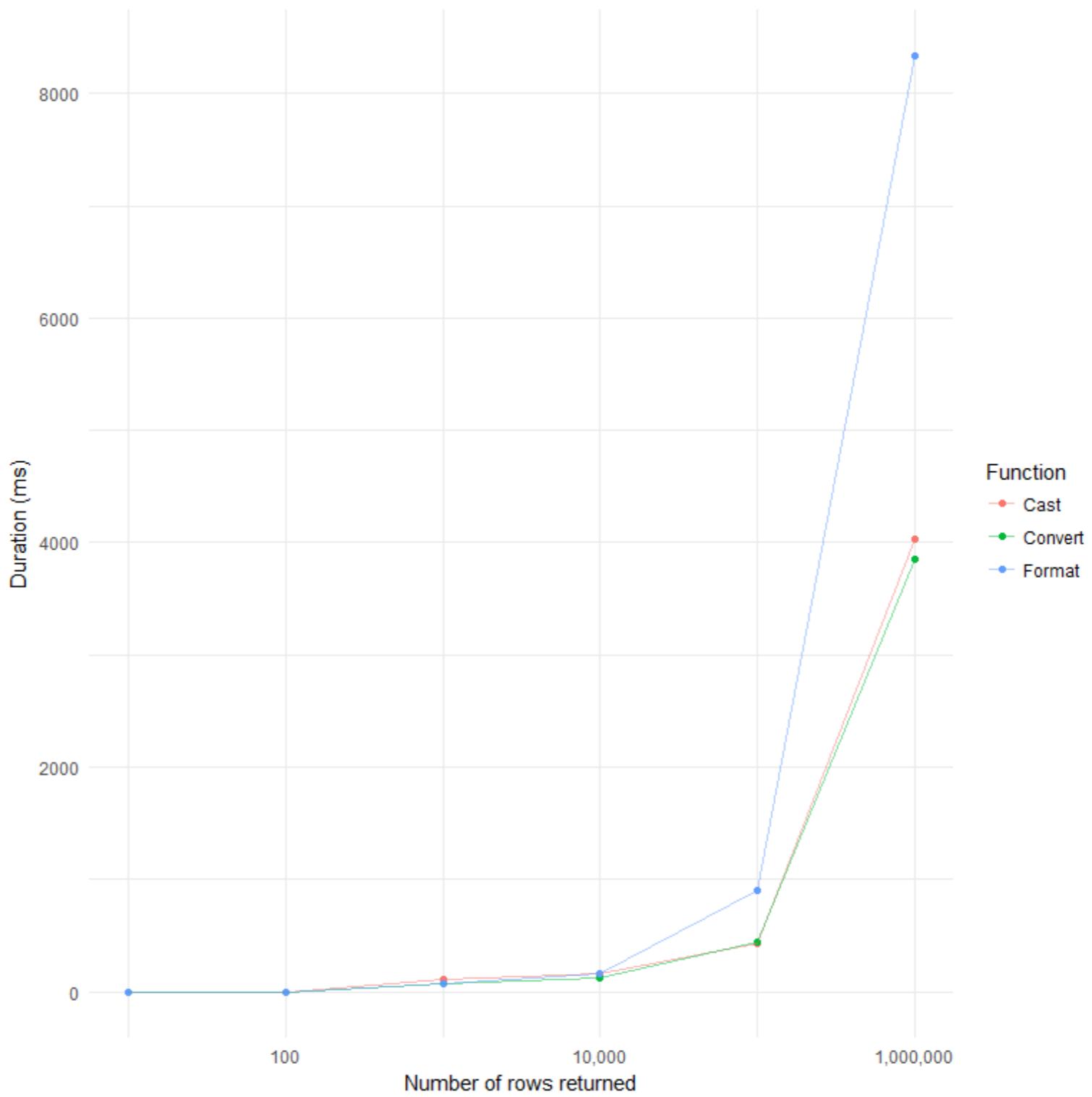
```
@SomeDate DATETIME2(3) = '1793-02-21 11:13:19.033';
```

## SELECT

```
FORMAT(@SomeDate, 'd', 'en-US') AS US_d,  
FORMAT(@SomeDate, 'd', 'de-DE') AS DE_d,  
FORMAT(@SomeDate, 'D', 'de-DE') AS DE_D,  
FORMAT(@SomeDate, 'yyyy-MM-dd') AS yMd;
```

US_d	DE_d	DE_D	yMd
2/21/1793	21.02.1793	Donnerstag, 21. February 1793	1793-02-21

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings>



# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Working with calendar tables

TIME SERIES ANALYSIS IN SQL SERVER



**Kevin Feasel**  
CTO, Envizage

# What is a calendar table?

```
SELECT *
FROM dbo.Calendar;
```

DateKey	Date	Day	DayOfWeek	DayName	...
20000101	2000-01-01	1	7	Saturday	...
20000102	2000-01-02	2	1	Sunday	...
20000103	2000-01-03	3	2	Monday	...

# Contents of a calendar table

## General Columns

- Date
- Day Name
- Is Weekend

## Fiscal Year

- Fiscal week of year
- Fiscal quarter
- Fiscal first day of year

## Calendar Year

- Calendar month
- Calendar quarter
- Calendar year

## Specialized Columns

- Holiday name
- Lunar details
- ISO week of year

# Building a calendar table

```
CREATE TABLE dbo.Calendar
```

```
(  
    DateKey INT NOT NULL,  
    [Date] DATE NOT NULL,  
    [Day] TINYINT NOT NULL,  
    DayOfWeek TINYINT NOT NULL,  
    DayName VARCHAR(10) NOT NULL,  
    ...  
)
```

```
SELECT
```

```
    CAST(D.DateKey AS INT) AS DateKey,  
    D.[DATE] AS [Date],  
    CAST(D.[day] AS TINYINT) AS [day],  
    CAST(d.[dayofweek] AS TINYINT) AS [DayOfWeek],  
    CAST(DATENAME(WEEKDAY, d.[Date]) AS VARCHAR(10)) AS [DayName],  
    ...
```

# Using a calendar table

**SELECT**

c.Date

**FROM** dbo.Calendar c

**WHERE**

c.MonthName = 'April'

**AND** c.DayName = 'Saturday'

**AND** c.CalendarYear = 2020

**ORDER BY**

c.Date;

**Date**

2020-04-04

2020-04-11

2020-04-18

2020-04-25

# Using a calendar table

**SELECT**

c.Date

**FROM** dbo.Calendar c

**WHERE**

c.MonthName = 'April'

**AND** c.DayName = 'Saturday'

**AND** c.CalendarYear = 2020

**ORDER BY**

c.Date;

**Date**

2020-04-04

2020-04-11

2020-04-18

2020-04-25

# A quick note on APPLY()

```
SELECT  
    FYStart =  
        DATEADD(MONTH, -6,  
            DATEADD(YEAR,  
                DATEDIFF(YEAR, 0,  
                    DATEADD(MONTH, 6, d.[date]))), 0)),  
    FiscalDayOfYear =  
        DATEDIFF(DAY,  
            DATEADD(MONTH, -6,  
                DATEADD(YEAR,  
                    DATEDIFF(YEAR, 0,  
                        DATEADD(MONTH, 6, d.[date]))), 0)), d.[Date]) + 1,  
    FiscalWeekOfYear =  
        DATEDIFF(WEEK,  
            DATEADD(MONTH, -6,  
                DATEADD(YEAR,  
                    DATEDIFF(YEAR, 0,  
                        DATEADD(MONTH, 6, d.[date]))), 0)), d.[Date]) + 1  
FROM dbo.Calendar d;
```

# A quick note on APPLY()

```
SELECT
```

```
    fy.FYStart,  
    FiscalDayOfYear = DATEDIFF(DAY, fy.FYStart, d.[Date]) + 1,  
    FiscalWeekOfYear = DATEDIFF(WEEK, fy.FYStart, d.[Date]) + 1
```

```
FROM dbo.Calendar d
```

```
CROSS APPLY
```

```
(
```

```
    SELECT FYStart =  
        DATEADD(MONTH, -6,  
                DATEADD(YEAR,  
                        DATEDIFF(YEAR, 0,  
                                DATEADD(MONTH, 6, d.[date]))), 0))
```

```
) fy;
```

# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Building dates from parts

TIME SERIES ANALYSIS IN SQL SERVER



SQL

**Kevin Feasel**  
CTO, Envizage

# Dates from parts

`DATEFROMPARTS(year, month, day)`

`TIMEFROMPARTS(hour, minute, second, fraction, precision)`

`DATETIMEFROMPARTS(year, month, day, hour, minute, second, ms)`

`DATETIME2FROMPARTS(year, month, day, hour, minute, second, fraction, precision)`

`SMALLDATETIMEFROMPARTS(year, month, day, hour, minute)`

`DATETIMEOFFSETFROMPARTS(year, month, day, hour, minute, second, fraction, hour_offset, minute_offset, precision)`

# Dates and times together

SELECT

```
DATETIMEFROMPARTS(1918, 11, 11, 05, 45, 17, 995) AS DT,  
DATETIME2FROMPARTS(1918, 11, 11, 05, 45, 17, 0, 0) AS DT20,  
DATETIME2FROMPARTS(1918, 11, 11, 05, 45, 17, 995, 3) AS DT23,  
DATETIME2FROMPARTS(1918, 11, 11, 05, 45, 17, 9951234, 7) AS DT27;
```

DT	DT20	DT23	DT27
1918-11-11 05:45:17.997	1918-11-11 05:45:17	1918-11-11 05:45:17.995	1918-11-11 05:45:17.9951234

# Working with offsets

**SELECT**

```
DATETIMEOFFSETFROMPARTS(2009, 08, 14, 21,  
    00, 00, 0, 5, 30, 0) AS IST,  
DATETIMEOFFSETFROMPARTS(2009, 08, 14, 21,  
    00, 00, 0, 5, 30, 0)  
    AT TIME ZONE 'UTC' AS UTC;
```

IST	UTC
2009-08-14 21:00:00 +05:30	2009-08-14 15:30:00 +00:00

# Gotchas when working with parts

```
DATEFROMPARTS(1999, 12, NULL)
```

```
DATEFROMPARTS(10000, 01, 01)
```

```
DATETIME2FROMPARTS(1918, 11, 11, 05, 45, 17, 995, 0)
```

```
NULL
```

Cannot construct data type date, some of the arguments have values which are not valid.

Cannot construct data type datetime2, some of the arguments have values which are not valid.

# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Translating date strings

TIME SERIES ANALYSIS IN SQL SERVER



SQL

**Kevin Feasel**  
CTO, Envizage

# Casting strings

```
SELECT
```

```
    CAST('09/14/99' AS DATE) AS USDate;
```

USDate
--------

1999-09-14
------------

# Converting Strings

```
SELECT
```

```
    CONVERT(DATETIME2(3),  
        'April 4, 2019 11:52:29.998 PM') AS April4
```

```
April4
```

```
2019-04-04 23:52:29.998
```

# Parsing strings

```
SELECT
```

```
PARSE('25 Dezember 2014' AS DATE  
      USING 'de-de') AS Weihnachten;
```

Weihnachten

2014-12-25

# The cost of parsing

Function	Conversions Per Second
CONVERT()	251,997
CAST()	240,347
PARSE()	12,620

# Setting languages

```
SET LANGUAGE 'FRENCH'
```

```
DECLARE
```

```
    @FrenchDate NVARCHAR(30) = N'18 avril 2019',  
    @FrenchNumberDate NVARCHAR(30) = N'18/4/2019';
```

```
SELECT
```

```
    CAST(@FrenchDate AS DATETIME),  
    CAST(@FrenchNumberDate AS DATETIME);
```

```
2019-04-18 00:00:00.000
```

# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Working with offsets

TIME SERIES ANALYSIS IN SQL SERVER



SQL

**Kevin Feasel**  
CTO, Envizage

# Anatomy of a DATETIMEOFFSET

## Components

Date Part	Example
Date	2019-04-10
Time	12:59:02.3908505
UTC Offset	-04:00

# Anatomy of a DATETIMEOFFSET

## Components

Date Part	Example
Date	2019-04-10
Time	12:59:02.3908505
UTC Offset	-04:00

## Display

2019-04-10 12:59:02.3908505 -04:00

# Changing offsets

```
DECLARE @SomeDate DATETIMEOFFSET =  
'2019-04-10 12:59:02.3908505 -04:00';  
  
SELECT  
SWITCHOFFSET(@SomeDate, '-07:00') AS LATime;
```

LATime

2019-04-10 09:59:02.3908505 -07:00

# Converting to DATETIMEOFFSET

```
DECLARE @SomeDate DATETIME2(3) =  
'2019-04-10 12:59:02.390';
```

```
SELECT  
TODATETIMEOFFSET(@SomeDate, '-04:00') AS EDT;
```

EDT

2019-04-10 12:59:02.390 -04:00

# Time zone swaps with TODATETIMEOFFSET

```
DECLARE @SomeDate DATETIME2(3) =  
'2016-09-04 02:28:29.681';
```

```
SELECT  
    TODATETIMEOFFSET(  
        DATEADD(HOUR, 7, @SomeDate),  
        '+02:00') AS BonnTime;
```

BonnTime

2016-09-04 09:28:29.681 +02:00

# Discovering time zones

**SELECT**

```
tzi.name,  
tzi.current_utc_offset,  
tzi.is_currently_dst
```

**FROM** sys.time\_zone\_info tzi

**WHERE**

```
tzi.name LIKE '%Time Zone%';
```

<b>name</b>	<b>current_utc_offset</b>	<b>is_currently_dst</b>
Russia Time Zone 3	+04:00	0
Russia Time Zone 10	+11:00	0
Russia Time Zone 11	+12:00	0

# **Let's practice!**

**TIME SERIES ANALYSIS IN SQL SERVER**

# Handling invalid dates

TIME SERIES ANALYSIS IN SQL SERVER



Kevin Feasel  
CTO

# Error-safe date conversion functions

## "Unsafe" Functions

`CAST()`

`CONVERT()`

`PARSE()`

## Safe Functions

`TRY_CAST()`

`TRY_CONVERT()`

`TRY_PARSE()`

# When everything goes right

```
SELECT
```

```
    PARSE('01/08/2019' AS DATE USING 'en-us') AS January8US,  
    PARSE('01/08/2019' AS DATE USING 'fr-fr') AS August1FR;
```

```
GO
```

Results:

January8US	August1FR
2019-01-08	2019-08-01

# When everything goes wrong

```
SELECT
```

```
    PARSE('01/13/2019' AS DATE USING 'en-us') AS January13US,  
    PARSE('01/13/2019' AS DATE USING 'fr-fr') AS Smarch1FR;
```

```
GO
```

**Msg 9819, Level 16, State 1, Line 1**

Error converting string value '01/13/2019' into data type date using culture 'fr-fr'.

# Doing right when everything goes wrong

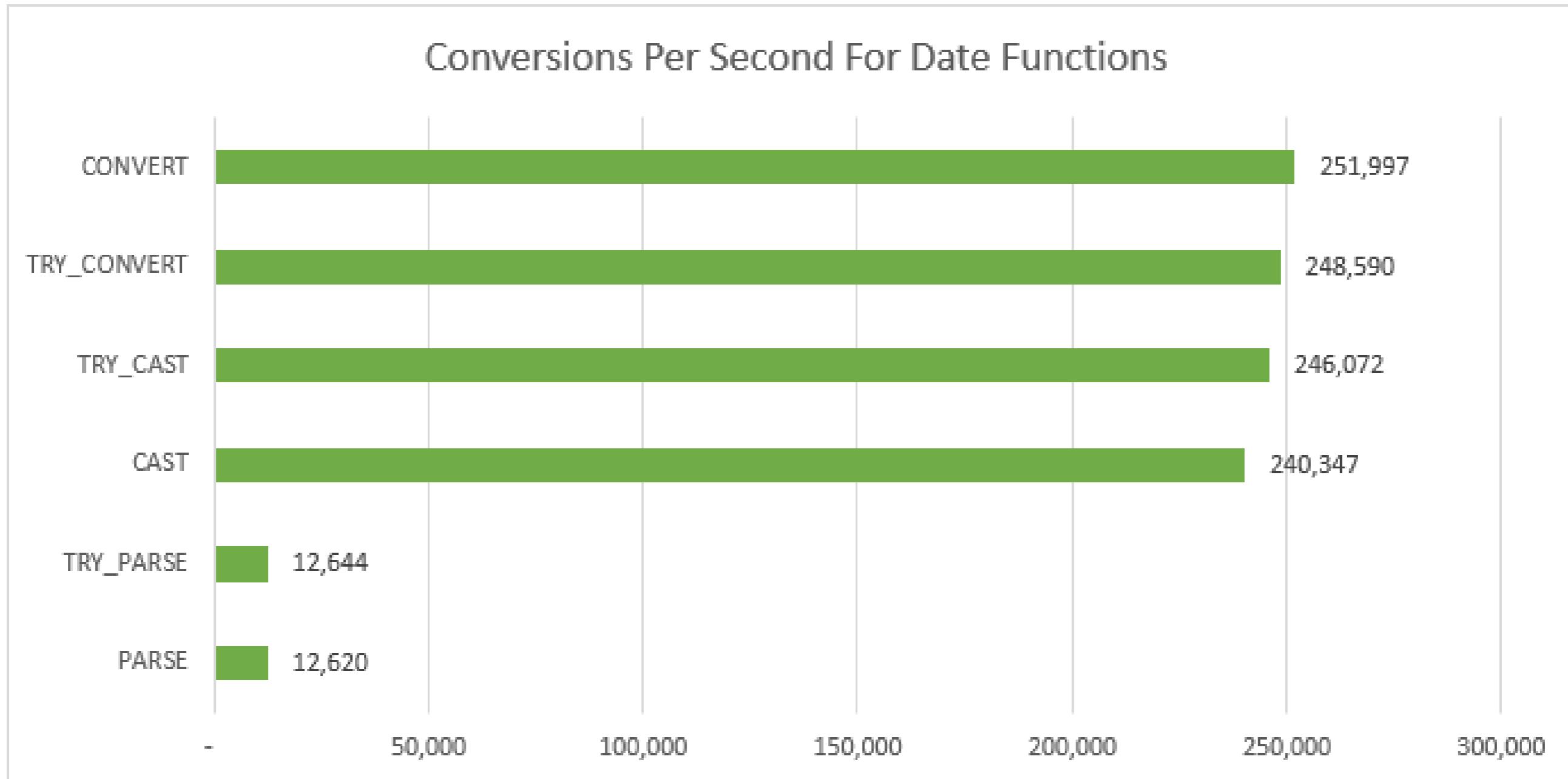
```
SELECT
```

```
    TRY_PARSE('01/13/2019' AS DATE USING 'en-us') AS January13US,  
    TRY_PARSE('01/13/2019' AS DATE USING 'fr-fr') AS Smarch1FR;
```

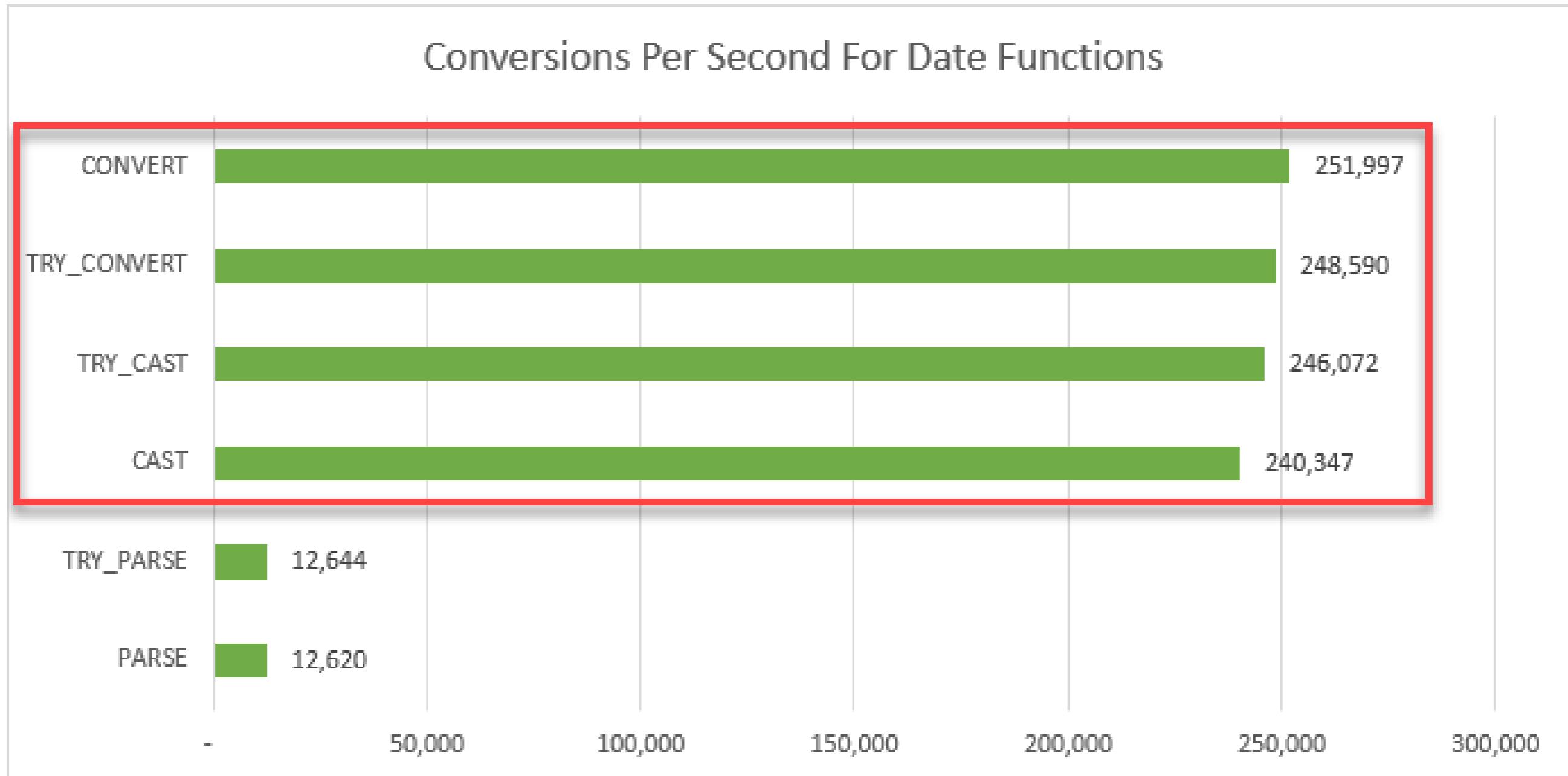
```
GO
```

January13US	Smarch1FR
2019-01-13	NULL

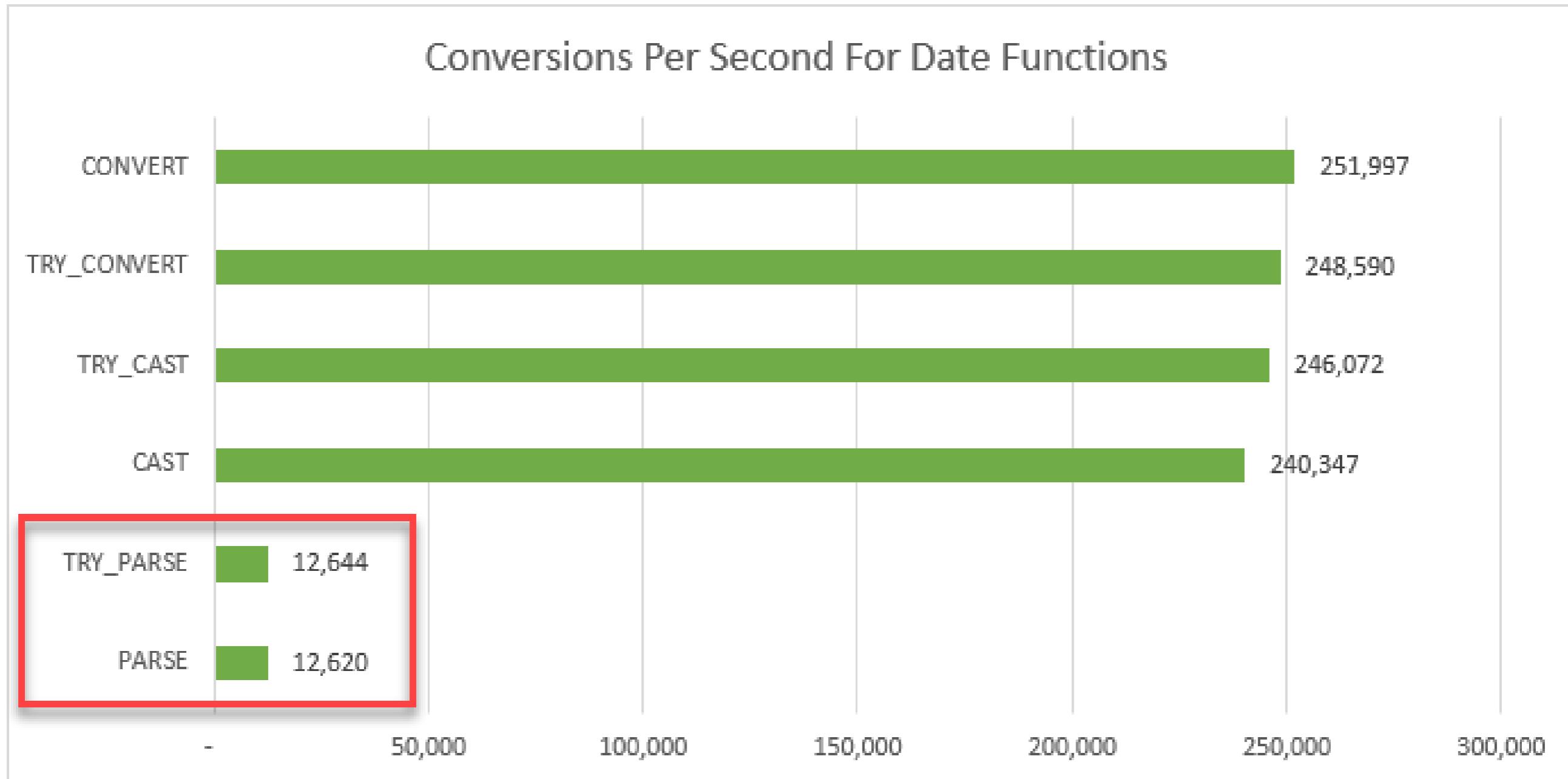
# The cost of safety



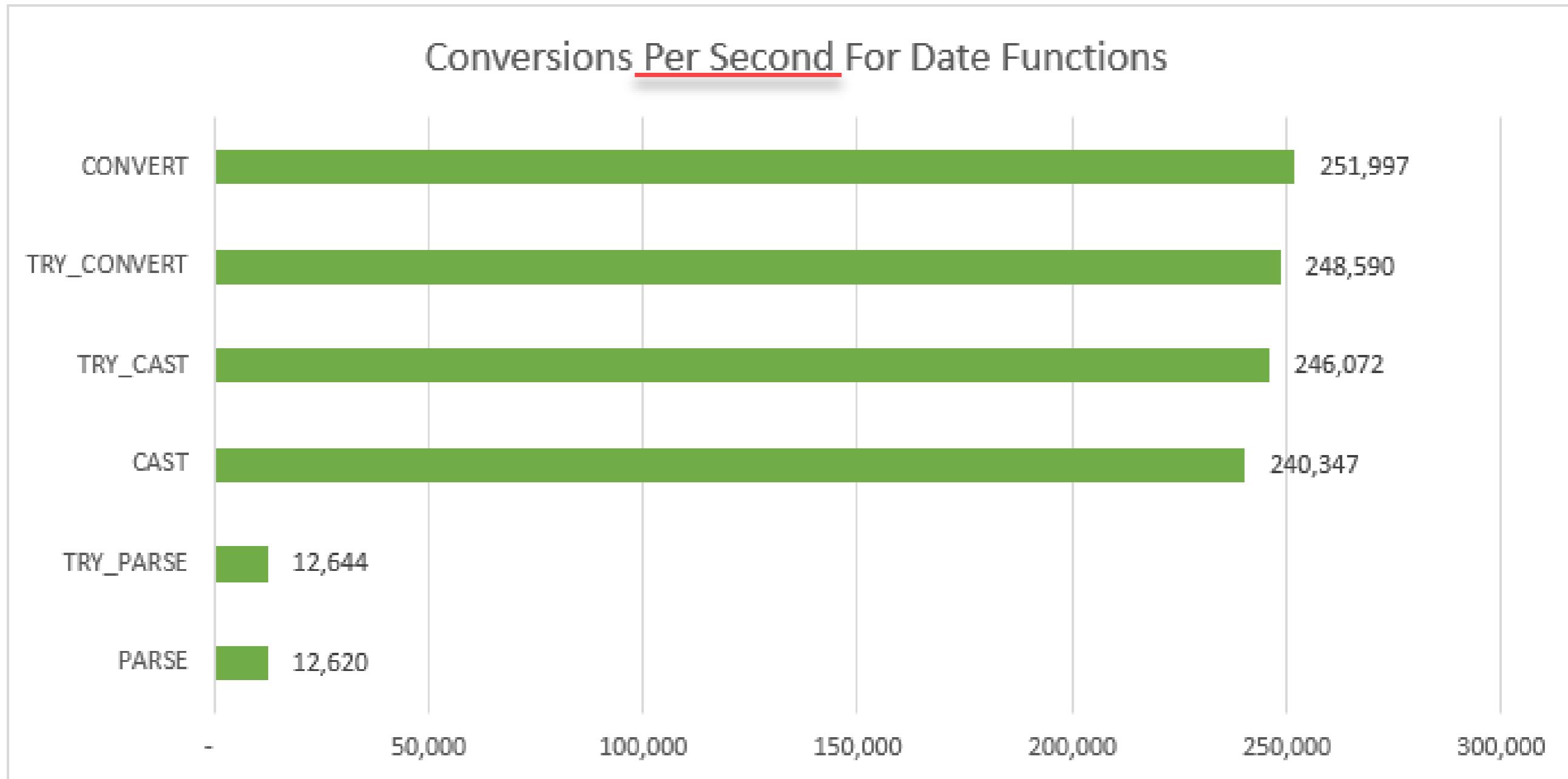
# The cost of safety



# The cost of safety



# The cost of safety



# **Let's practice**

## **TIME SERIES ANALYSIS IN SQL SERVER**