SQL MasterClass

Barcelona Data Institute

Iago Novoa
Head of BI at letgo

Write in a paper

Favourite place on earth

SQL MasterClass

Barcelona Data Institute

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

Barcelona Data Institute

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

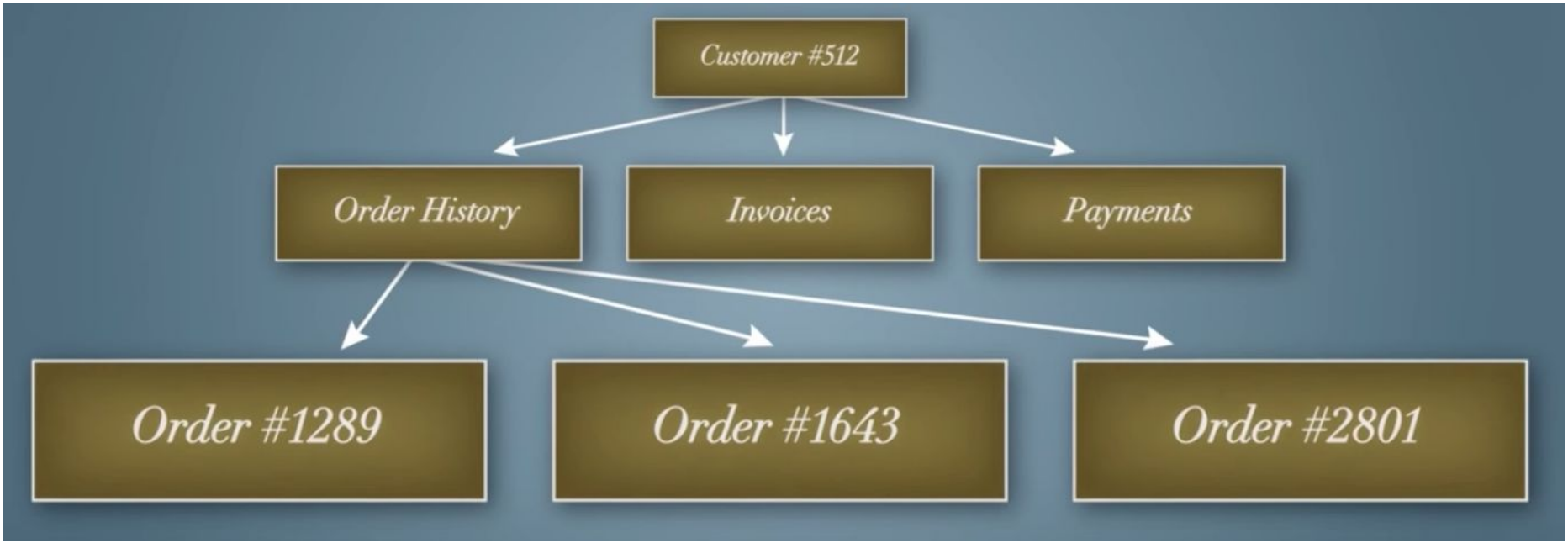
SQL MasterClass

Barcelona Data Institute

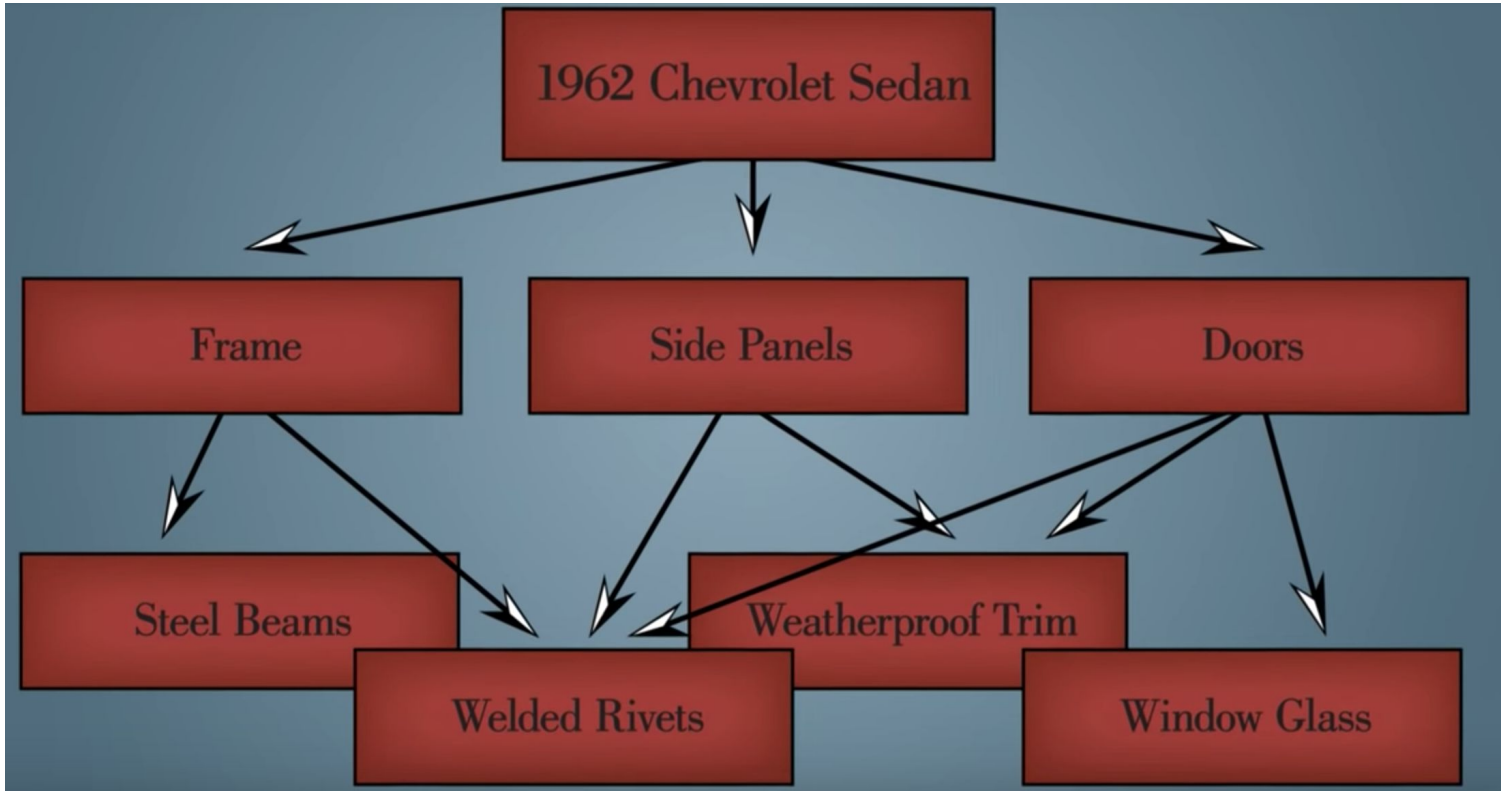
Those of us currently alive represent ~7% of total number of humans who have ever lived

Human beings began to store information very long ago

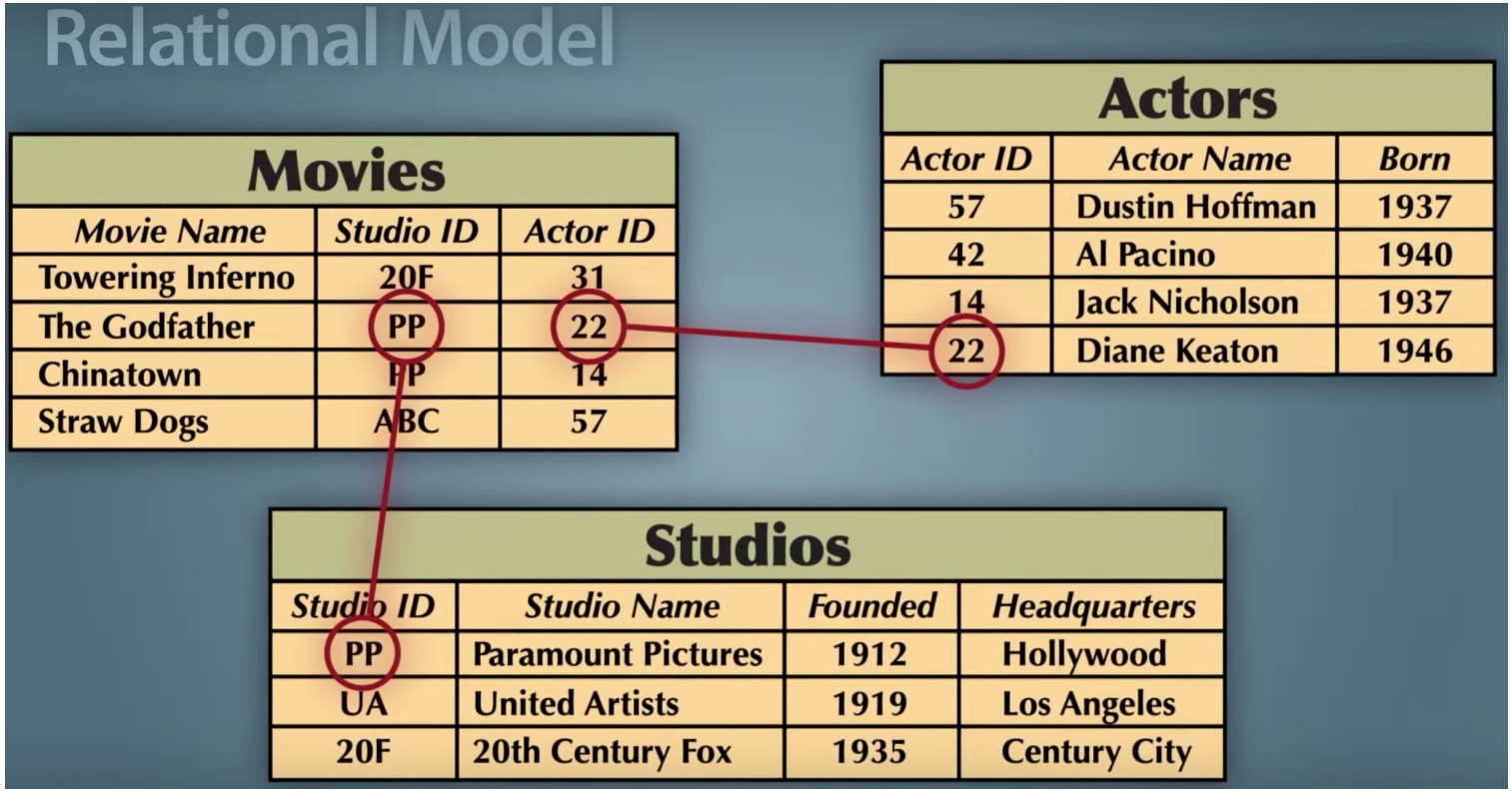
Computerized DBs started in 1960s with Hierarchical Model …



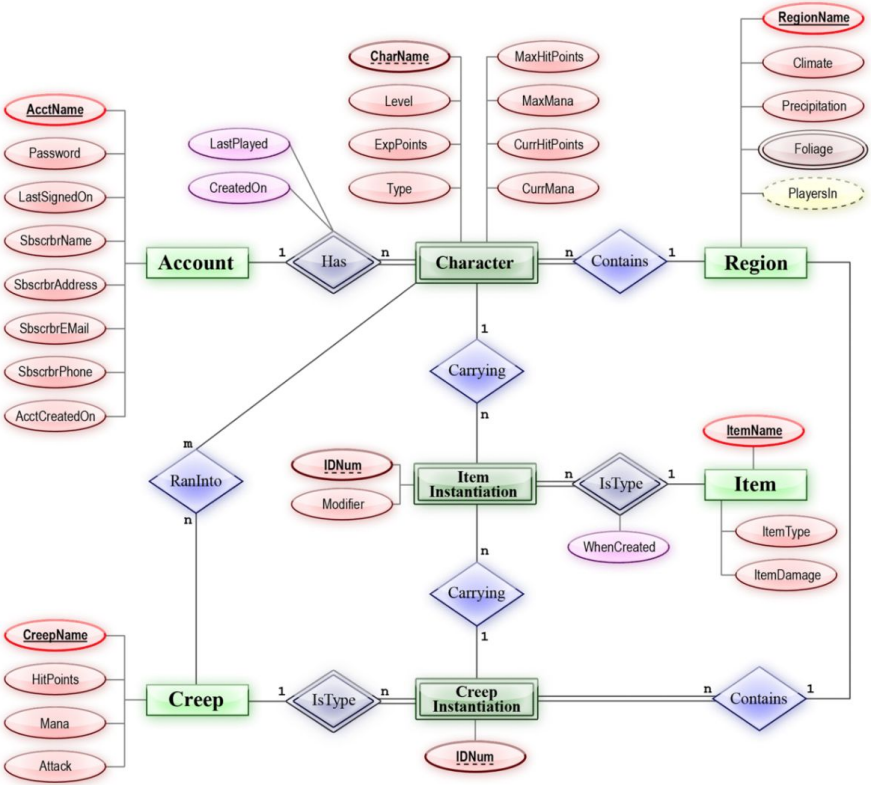
... and Network Models



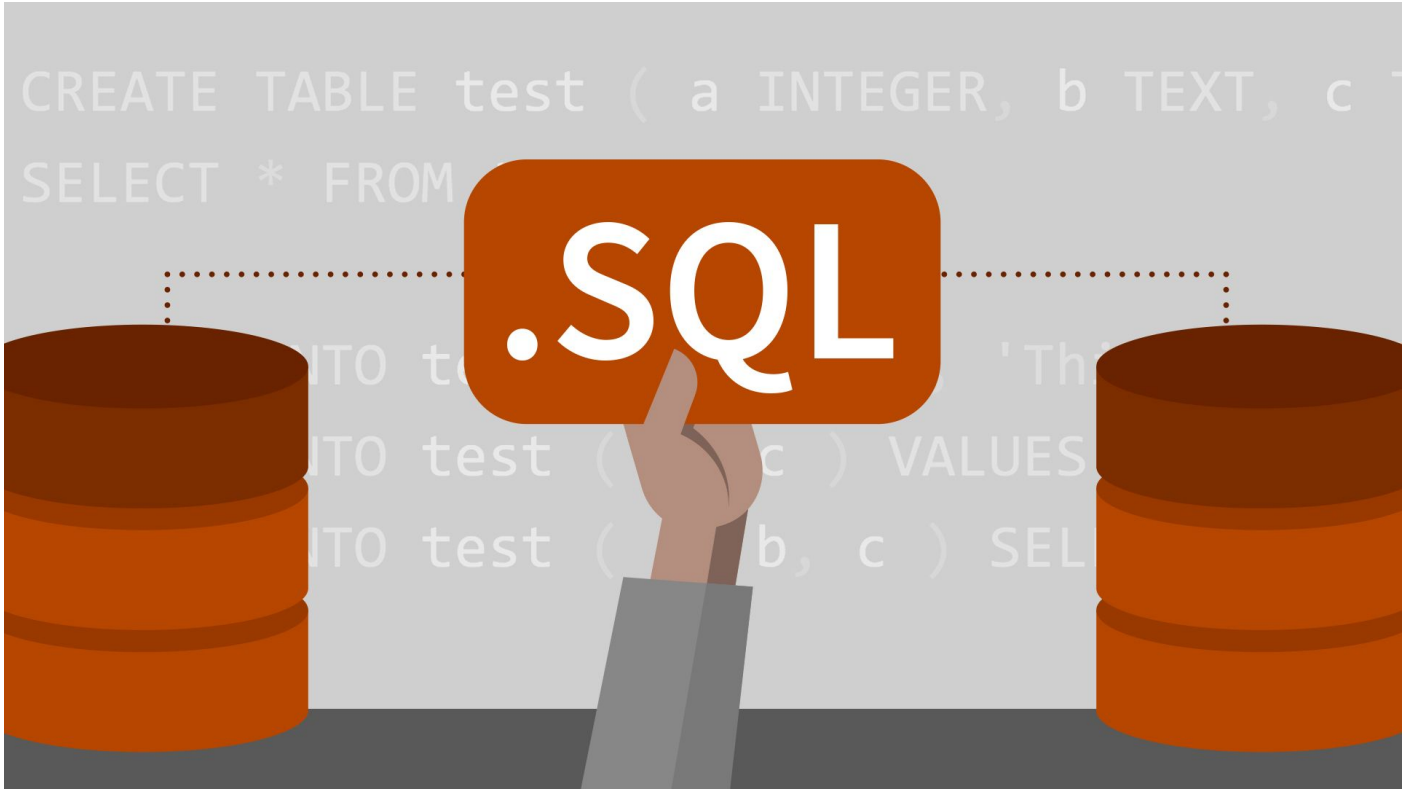
In the 1970s Codd proposed the use of Relational Model



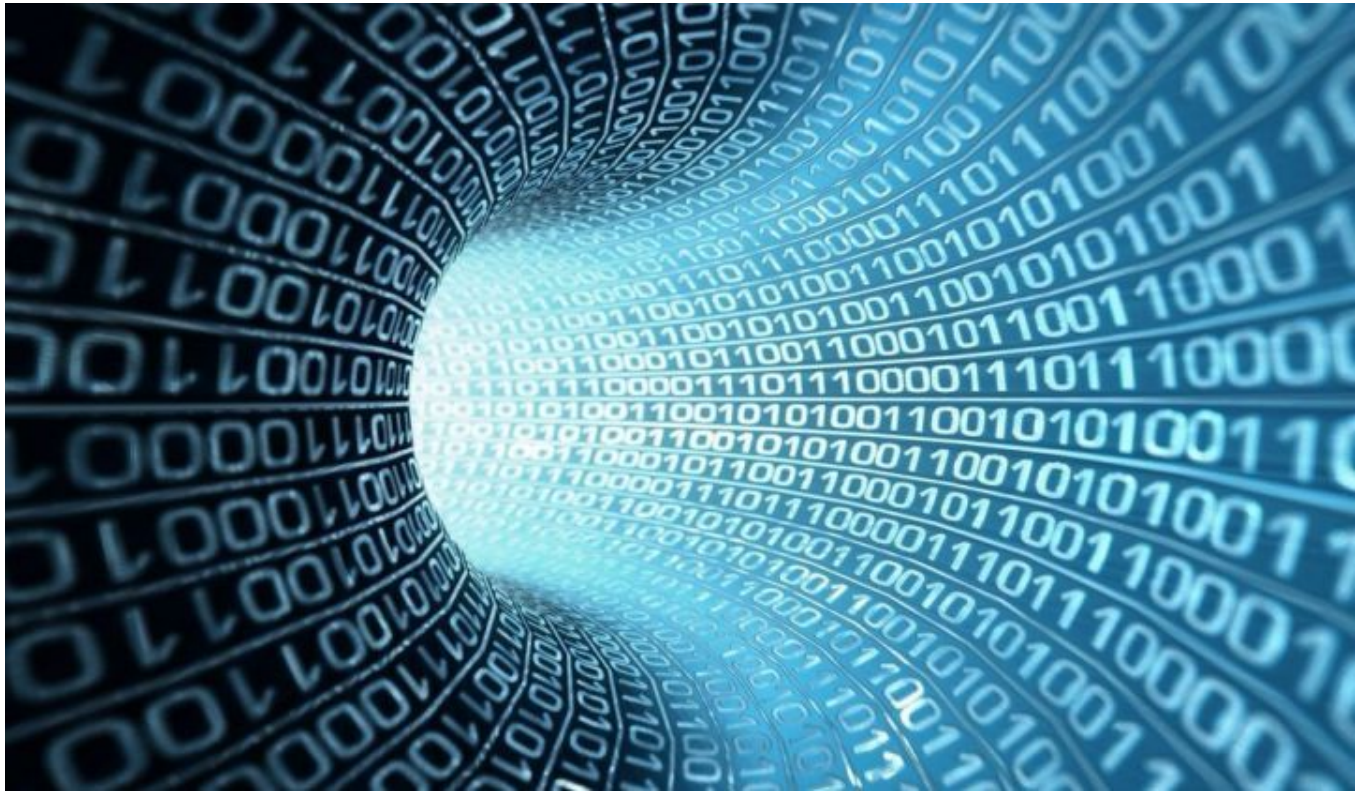
And Chen proposed Entity-Relationship (er) model



In 1980s SQL became the standard query language



Internet led to exponential growth of the database industry



2.5 QUINTILLION BYTES OF DATA CREATED EACH DAY

Google

Google now processes more than
3.5 billion searches per day

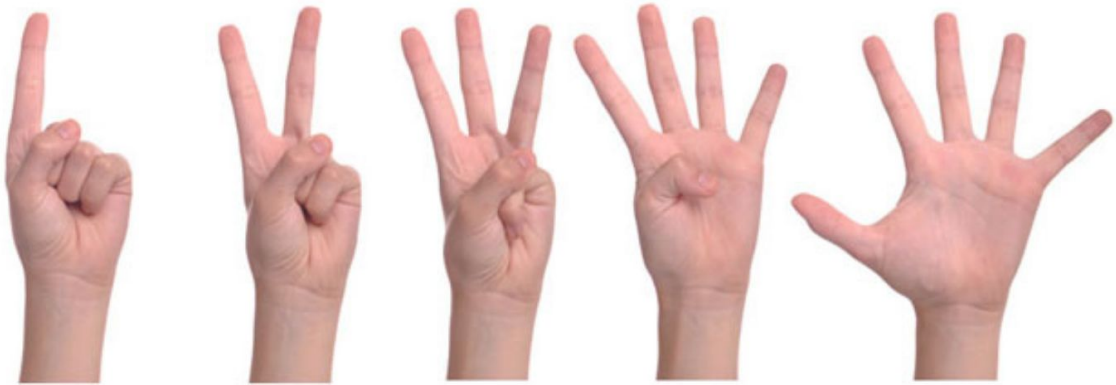
Over the last two years alone 90 percent of the data in the world was generated

Snapchat users share
527,760
photos every minute

YouTube

Users watch
248,796,600
YouTube videos per hour

DataBase Management System (DBMS) is a software to define, build, maintain, ... a database


I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. **Our Data World**
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

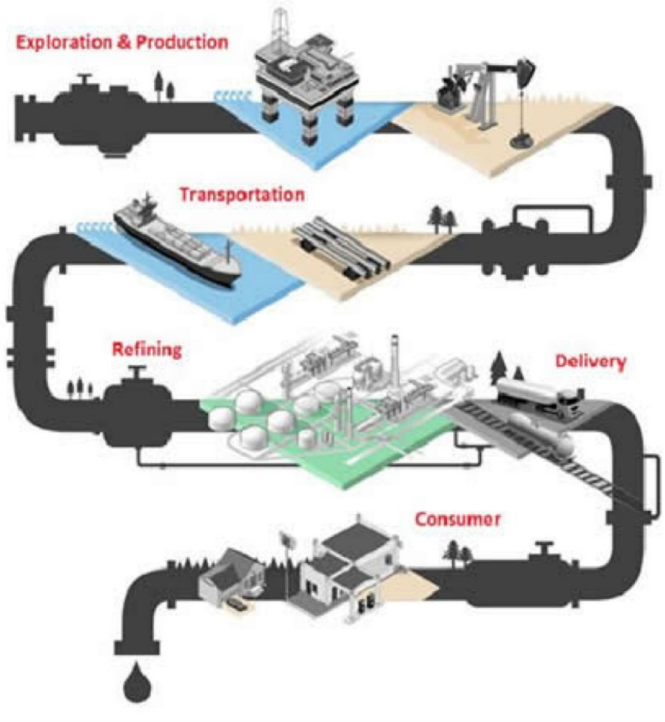
Barcelona Data Institute


amazon
UBER
Microsoft
Google
TESLA
David Parkins

Data is the main driver of many deals nowadays

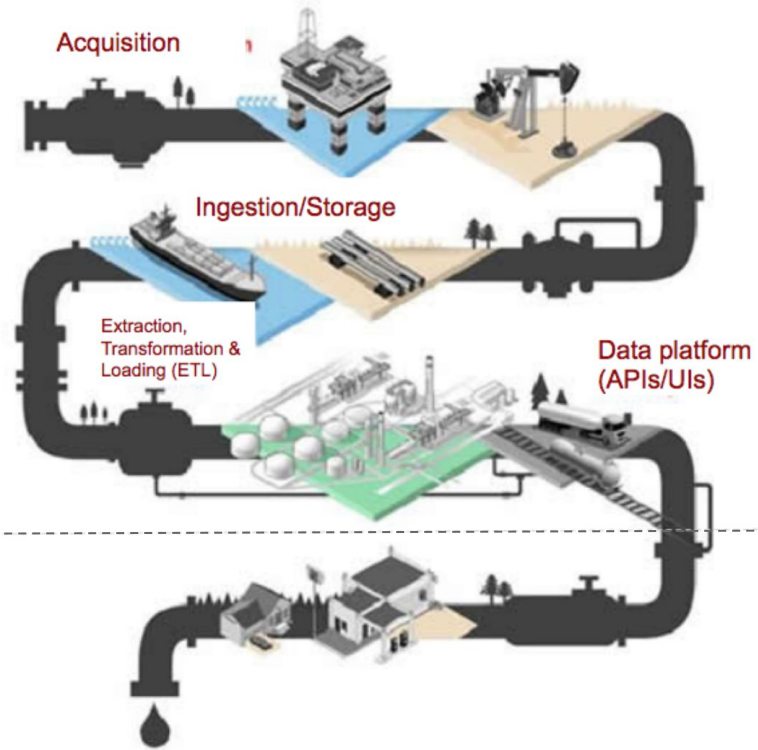
	Target company *(date)*	Value of deal *($USbn)*	Business
facebook	INSTAGRAM *(2012)*	1.0	PHOTO SHARING
	WHATSAPP *(2014)*	?	TEXT/PHOTO MESSAGING
Alphabet	WAZE *(2013)*	1.2	MAPPING AND NAVIGATION
IBM	THE WEATHER COMPANY *(2015)*	2.0	METEOROLOGY
	TRUVEN HEALTH ANALYTICS *(2016)*	2.6	HEALTH CARE
intel	MOBILEYE *(2017)*	15.3	SELF-DRIVING CARS
Microsoft	SWIFTKEY *(2016)*	0.25	KEYBOARD/AI
	LINKEDIN *(2016)*	?	BUSINESS NETWORKING
ORACLE	BLUEKAI *(2014)*	0.4	CLOUD DATA PLATFORM
	DATALOGIX *(2014)*	1.0	MARKETING

The oil generation



The data generation

Data Engineer



Data Analyst / Data Scientist

Building well-balanced data product teams

"VW Beetle Jet" Team

- 3 Data Analysts / Data Scientists
- 2 Data Engineers
- 1 Product Manager

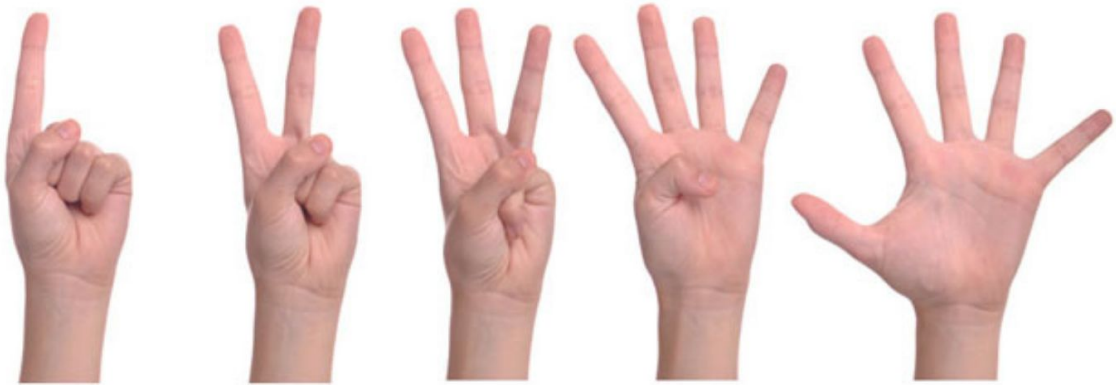
- Noisy
- Not fuel efficient
- Can explode

Building well-balanced data product teams

"Prius" Team

- 1 Data Analyst / Data Scientist
- 1 Data Engineer
- 1 Backend Engineer
- 1 Product Manager

- Nothing fancy
- Very efficient
- Will definitely not xplode


I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. **Amazon Redshift vs Google BigQuery**
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

Barcelona Data Institute

McDonald's or Burger King? Nike or Adidas?

VS

Amazon Redshift vs Google BigQuery

Amazon Redshift	Google BigQuery
Fork of PostgreSQL	Was built from scratch
Relational	Nested data structures are first class citizen
SQL	NoSQL (but now it also speaks SQL)
Analysts are more familiar with it	Developers love it

Loading Data ...

	Amazon Redshift	Google BigQuery
Bulk data:	?	Google Cloud Storage
Streaming data:	Kinesis	Streaming Inserts
Other sources:		Google Analytics Premium
Data Serializations:	CSV, Avro, JSON	CSV, Avro, JSON

Data Modeling

	Amazon Redshift	Google BigQuery
Data organization:	Tables, Schemas	Tables, Datasets
Datatypes:	Subset of SQL datatypes	Custom Datatypes with mappings
Nested datatypes:	Not supported	Natively supported
Table manipulation:	Supported with SQL.	?
Support for updates and deletes:	Natively through SQL with no additional costs.	Limited and expensive.

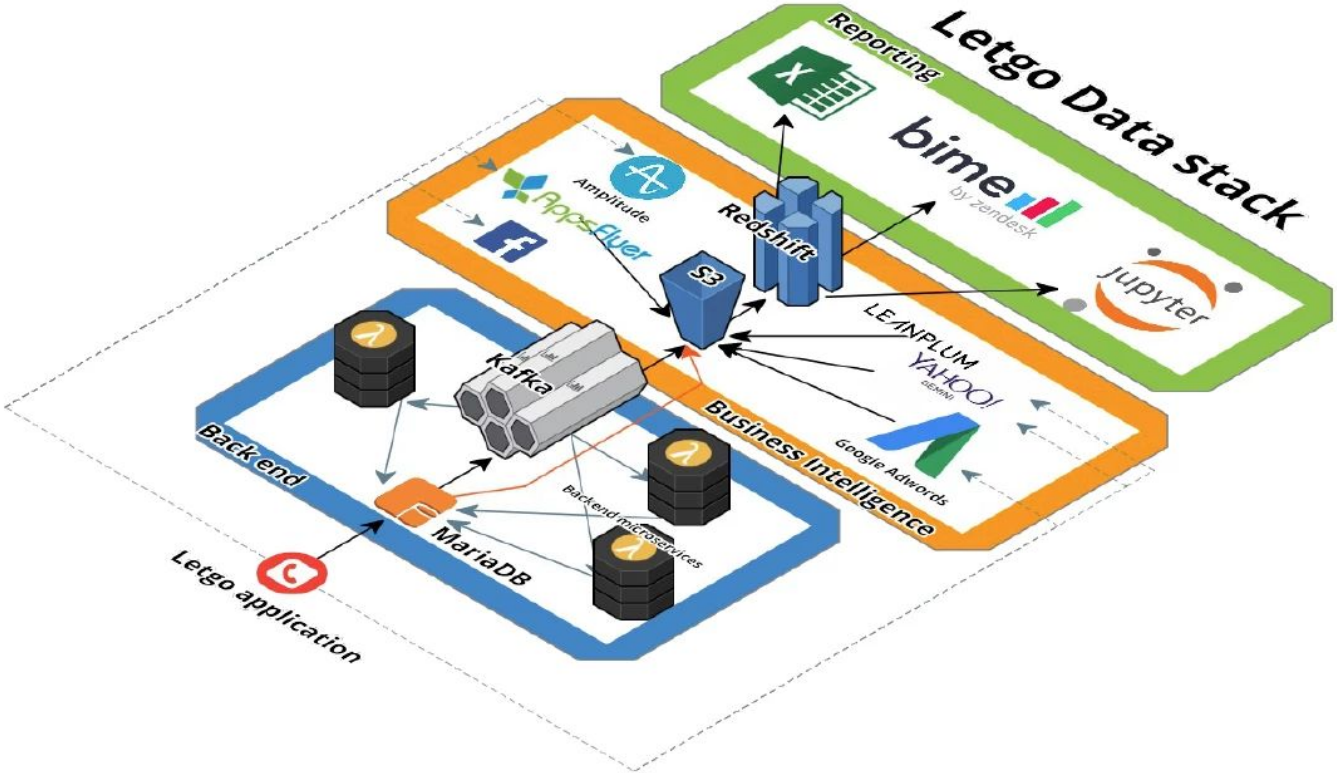
Data Consistency

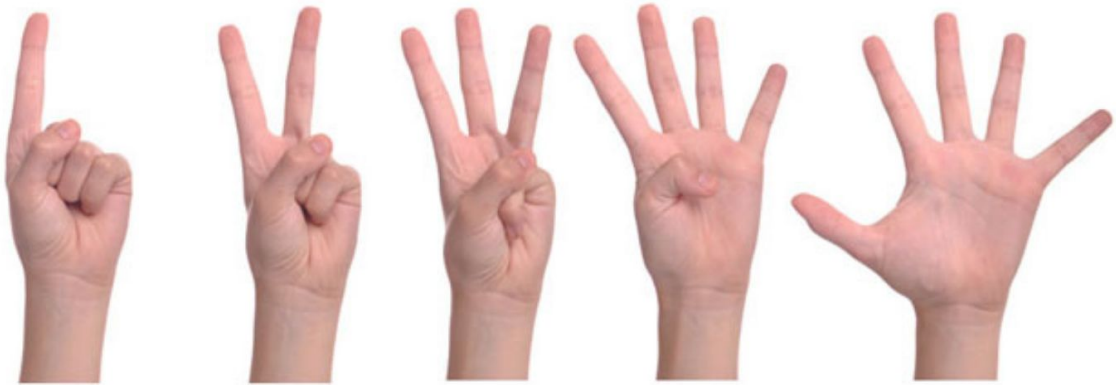
	Amazon Redshift	Google BigQuery
Transactions:	Yes	No
Deduplication (batch)	Easy to do	?
Deduplication (Streaming)	No	Yes with time window + ID
Delivery Semantics (Streaming)	At least once (Kinesis)	At least once with best effort deduplication.

Other things to take into account

- Cluster Management
- Connectivity
- Authentication
- Resource Allocation
- Database Optimization
- Cost
- Ecosystem
- ...

Letgo Data Stack

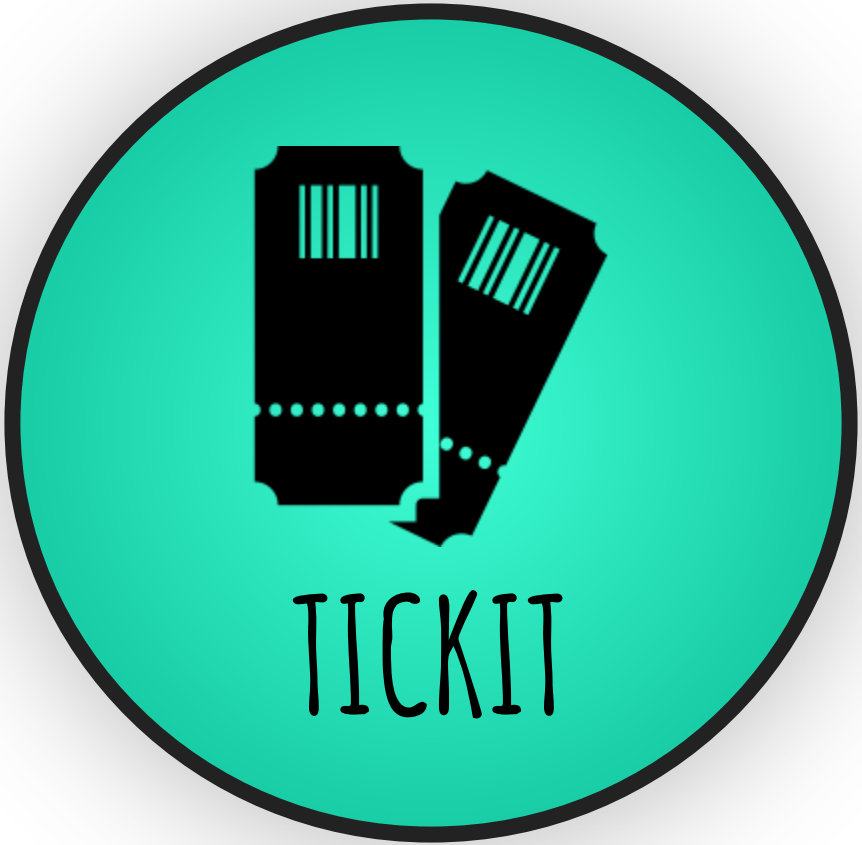



I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. **How to create a Database**
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

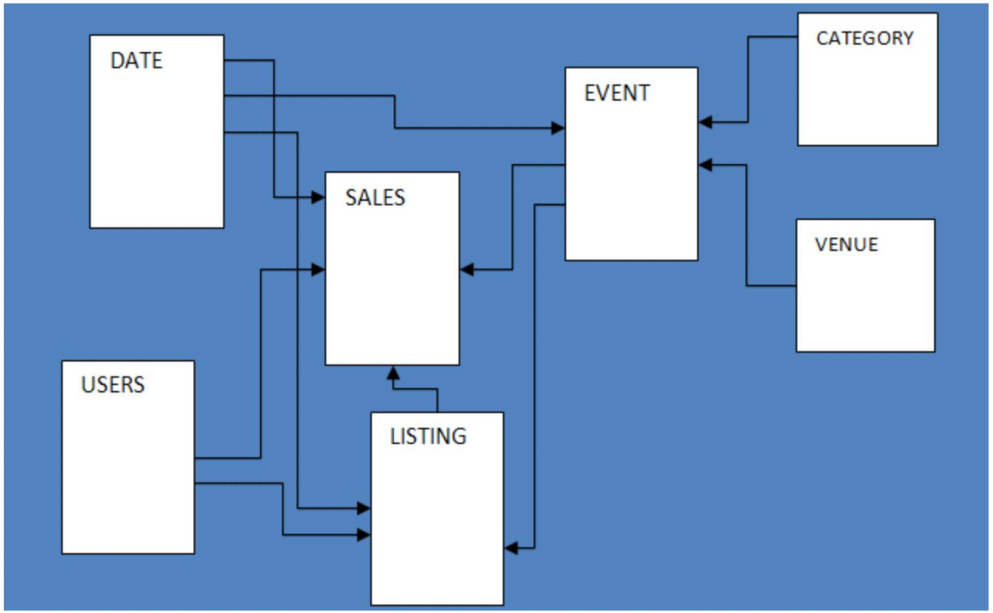
SQL MasterClass

Barcelona Data Institute


TICKIT

1. Design our DataBase

This sample database application helps analysts track sales activity for the fictional TICKIT web site, where users buy and sell tickets online for sporting events, shows, and concerts. In particular, analysts can identify ticket movement over time, success rates for sellers, and the best-selling events, venues, and seasons. Analysts can use this information to provide incentives to buyers and sellers who frequent the site, to attract new users, and to drive advertising and promotions.



2. Define Tables

CATEGORY Table

Column Name	Data Type	Description
CATID	SMALLINT	Primary key, a unique ID value for each row. Each row represents a specific type of event for which tickets are bought and sold.
CATGROUP	VARCHAR(10)	Descriptive name for a group of events, such as **Shows** and **Sports**.
CATNAME	VARCHAR(10)	Short descriptive name for a type of event within a group, such as **Opera** and **Musicals**.
CATDESC	VARCHAR(30)	Longer descriptive name for the type of event, such as **Musical theatre**.

EVENT Table

Column Name	Data Type	Description
EVENTID	INTEGER	Primary key, a unique ID value for each row. Each row represents a separate event that takes place at a specific venue at a specific time.
VENUEID	SMALLINT	Foreign-key reference to the VENUE table.
CATID	SMALLINT	Foreign-key reference to the CATEGORY table.
DATEID	SMALLINT	Foreign-key reference to the DATE table.
EVENTNAME	VARCHAR(200)	Name of the event, such as **Hamlet** or **La Traviata**.
STARTTIME	TIMESTAMP	Full date and start time for the event, such as **2008-10-10 19:30:00**.

SALES Table

Column Name	Data Type	Description
SALESID	INTEGER	Primary key, a unique ID value for each row. Each row represents a sale of one or more tickets for a specific event, as offered in a specific listing.
LISTID	INTEGER	Foreign-key reference to the LISTING table.
SELLERID	INTEGER	Foreign-key reference to the USERS table (the user who sold the tickets).
BUYERID	INTEGER	Foreign-key reference to the USERS table (the user who bought the tickets).
EVENTID	INTEGER	Foreign-key reference to the EVENT table.
DATEID	SMALLINT	Foreign-key reference to the DATE table.
QTYSOLD	SMALLINT	The number of tickets that were sold, from **1** to **8**. (A maximum of 8 tickets can be sold in a single transaction.)
PRICEPAID	DECIMAL(8,2)	The total price paid for the tickets, such as **75.00** or **488.00**. The individual price of a ticket is PRICEPAID/QTYSOLD.
COMMISSION	DECIMAL(8,2)	The 15% commission that the business collects from the sale, such as **11.25** or **73.20**. The seller receives 85% of the PRICEPAID value.
SALETIME	TIMESTAMP	The full date and time when the sale was completed, such as **2008-05-24 06:21:47**.

VENUE Table

Column Name	Data Type	Description
VENUEID	SMALLINT	Primary key, a unique ID value for each row. Each row represents a specific venue where events take place.
VENUENAME	VARCHAR(100)	Exact name of the venue, such as **Cleveland Browns Stadium**.
VENUECITY	VARCHAR(30)	City name, such as **Cleveland**.
VENUESTATE	CHAR(2)	Two-letter state or province abbreviation (United States and Canada), such as **OH**.
VENUESEATS	INTEGER	Maximum number of seats available at the venue, if known, such as **73200**. For demonstration purposes, this column contains some null values and zeroes.

DATE Table

Column Name	Data Type	Description
DATEID	SMALLINT	Primary key, a unique ID value for each row. Each row represents a day in the calendar year.
CALDATE	DATE	Calendar date, such as **2008-06-24**.
DAY	CHAR(3)	Day of week (short form), such as **SA**.
WEEK	SMALLINT	Week number, such as **26**.
MONTH	CHAR(5)	Month name (short form), such as **JUN**.
QTR	CHAR(5)	Quarter number (**1** through **4**).
YEAR	SMALLINT	Four-digit year (**2008**).
HOLIDAY	BOOLEAN	Flag that denotes whether the day is a public holiday (U.S.).

LISTING Table

Column Name	Data Type	Description
LISTID	INTEGER	Primary key, a unique ID value for each row. Each row represents a listing of a batch of tickets for a specific event.
SELLERID	INTEGER	Foreign-key reference to the USERS table, identifying the user who is selling the tickets.
EVENTID	INTEGER	Foreign-key reference to the EVENT table.
DATEID	SMALLINT	Foreign-key reference to the DATE table.
NUMTICKETS	SMALLINT	The number of tickets available for sale, such as **2** or **20**.
PRICEPERTICKET	DECIMAL(8,2)	The fixed price of an individual ticket, such as **27.00** or **206.00**.
TOTALPRICE	DECIMAL(8,2)	The total price for this listing (NUMTICKETS*PRICEPERTICKET).
LISTTIME	TIMESTAMP	The full date and time when the listing was posted, such as **2008-03-18 07:19:35**.

USERS Table

Column Name	Data Type	Description
USERID	INTEGER	Primary key, a unique ID value for each row. Each row represents a registered user (a buyer or seller or both) who has listed or bought tickets for at least one event.
USERNAME	CHAR(8)	An 8-character alphanumeric username, such as **PGL08LJI**.
FIRSTNAME	VARCHAR(30)	The user's first name, such as **Victor**.
LASTNAME	VARCHAR(30)	The user's last name, such as **Hernandez**.
CITY	VARCHAR(30)	The user's home city, such as **Naperville**.
STATE	CHAR(2)	The user's home state, such as **GA**.
EMAIL	VARCHAR(100)	The user's email address; this column contains random Latin values, such as **turpis@accumsanlaoreet.org**.
PHONE	CHAR(14)	The user's 14-character phone number, such as **(818) 765-4255**.
LIKESPORTS, ...	BOOLEAN	A series of 10 different columns that identify the user's likes and dislikes with **true** and **false** values.

3. Create Tables

```
create table users(
        userid integer not null distkey sortkey,
        username char(8),
        firstname varchar(30),
        lastname varchar(30),
        city varchar(30),
        state char(2),
        email varchar(100),
        phone char(14),
        likesports boolean,
        liketheatre boolean,
        likeconcerts boolean,
        likejazz boolean,
        likeclassical boolean,
        likeopera boolean,
        likerock boolean,
        likevegas boolean,
        likebroadway boolean,
        likemusicals boolean);

create table venue(
        venueid smallint not null distkey sortkey,
        venuename varchar(100),
        venuecity varchar(30),
        venuestate char(2),
```

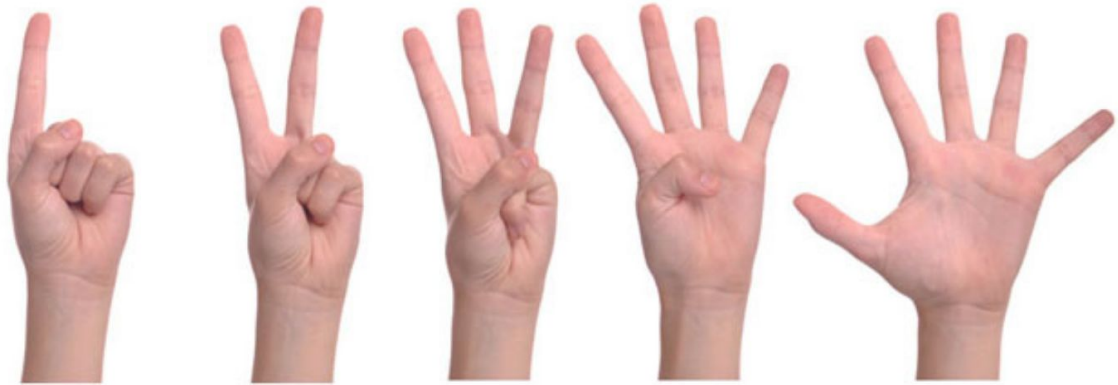
4. Load data

```
copy users from 's3://awssampledbuswest2/tickit/allusers_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' region 'us-west-2';

copy venue from 's3://awssampledbuswest2/tickit/venue_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' region 'us-west-2';

copy category from 's3://awssampledbuswest2/tickit/category_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' region 'us-west-2';

copy date from 's3://awssampledbuswest2/tickit/date2008_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' region 'us-west-2';

copy event from 's3://awssampledbuswest2/tickit/allevents_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-west-2';

copy listing from 's3://awssampledbuswest2/tickit/listings_pipe.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '|' region 'us-west-2';

copy sales from 's3://awssampledbuswest2/tickit/sales_tab.txt'
credentials 'aws_iam_role=<iam-role-arn>'
delimiter '\t' timeformat 'MM/DD/YYYY HH:MI:SS' region 'us-west-2';
```

Now we are ready to start querying...

select * from sql_masterclass.sales;

select * from sql_masterclass.sales limit 10;


I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. **Basic SQL**
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

Barcelona Data Institute

SQL is a key skill for any Data Analyst

SELECT ...

Get the name of 5 events

```
select
  eventname
from
  sql_masterclass.event
limit 5;
```

Result

select eventname from sql_m

	eventname
1	Mamma Mia!
2	Return To Forever
3	Hannah Montana
4	Smashing Pumpkins
5	K.D. Lang

Basic Elements

Get all information from categories within Sports group

```
select * from sql_masterclass.category where catgroup='Sports';
```

Result

select * from sql_masterclass.categ | Enter a SQL expression to filter results (use Ct

	catid	catgroup	catname	catdesc
1	1	Sports	MLB	Major League Baseball
2	2	Sports	NHL	National Hockey League
3	3	Sports	NFL	National Football League
4	4	Sports	NBA	National Basketball Association
5	5	Sports	MLS	Major League Soccer

Expression Lists

Get name of all the venues in Florida and Ohio

```
select
  venuename
from
  sql_masterclass.venue
where
  venuestate IN ('FL', 'OH');
```

Result

select venuename from sql_masterc | En

	venuename
1	Quicken Loans Arena
2	American Airlines Arer
3	Progressive Field
4	BankAtlantic Center
5	Nationwide Arena
6	Dolphin Stadium
7	Paul Brown Stadium
8	Raymond James Stadi
9	Tropicana Field
10	E.J. Nutter Center
11	Great American Ball Pa
12	Columbus Crew Stadiu
13	Amway Arena
14	St. Pete Times Forum
15	Cleveland Browns Stac
16	Jacksonville Municipal

Matching Conditions

Get Full Name (first and last name) of 10 users with emails that end by 'nunc.edu'

```
select
    firstname || ' ' || lastname as full_name,
    email
from
    sql_masterclass.users
where
    email like '%nunc.edu'
limit 10;
```

Result

select firstname || ' ' || lastname as | Enter a SQL expressi

	full_name	email
1	Lana Ross	mi.lorem@nunc.edu
2	Anika Camacho	nibh.Aliquam@Fusce
3	Forrest Dunn	augue.scelerisque.m
4	Wesley Bowen	aliquam.adipiscing@i
5	Alma Bean	ln@consectetueripsu
6	Jescie Pickett	lobortis@Sednunc.ed
7	Yoko Burgess	vitae.velit@maurisanu
8	Xavier Mccray	velit.Pellentesque@nu
9	Medge Witt	Sed.nunc.est@eratSe
10	Rylee Cummings	rutrum.magna.Cras@

Comparison Conditions

Get Full Name of the venue and number of seats for venues with more than 1.000 seats ordering by number of seats (highest first)

```
select
    venuename,
    venueseats
from
    sql_masterclass.venue
where
    venueseats > 10000
order by venueseats desc;
```

Result

select venuename, venueseats from | Enter a SQL expr

	venuename	venueseats
1	FedExField	91,704
2	New York Giants Stadium	80,242
3	Arrowhead Stadium	79,451
4	INVESCO Field	76,125
5	Dolphin Stadium	74,916
6	Ralph Wilson Stadium	73,967
7	Jacksonville Municipal Stadium	73,800
8	Bank of America Stadium	73,298
9	Cleveland Browns Stadium	73,200
10	Lambeau Field	72,922
11	Reliant Stadium	72,000
12	Louisiana Superdome	72,000

Logical Conditions

Get username of 10 users that like musical and they also like jazz or opera

```
select
    username,
    likemusicals,
    likejazz,
    likeopera
from
    sql_masterclass.users
where
    likemusicals=true and (likejazz=true or likeopera=true)
limit 10;
```

Result

select username, likemusicals, likeja | Enter a SQL expression to filter result

	username	likemusicals	likejazz	likeopera
1	NDQ15VBM	true	true	true
2	TNP95CNK	true	[NULL]	true
3	UKK70QBF	true	[NULL]	true
4	MWK08RDN	true	[NULL]	true
5	UJE76BTX	true	[NULL]	true
6	MVZ73SEM	true	true	false
7	BFL14DIV	true	false	true
8	NBV80XDJ	true	[NULL]	true
9	RJK72ITL	true	true	[NULL]
10	QSF62BKU	true	true	true

Aggregate Functions

Get average and median of prices in sales

```
select
    avg(pricepaid),
    median(pricepaid)
from
    sql_masterclass.sales;
```

Result

select avg(pricepaid), median(price

	avg	median
1	642.28	386

Group By and Order By

Get number of users per state

```
select
    state,
    count(1)
from
    sql_masterclass.users
group by 1;
```

Result

select state, count(1) from sql_mas

	state	count
1	SK	1,835
2	WV	481
3	MB	1,916
4	BC	1,958
5	NU	1,883
6	MO	525
7	NL	1,919
8	CO	493

Get number of users per state ordering by biggest first

```
select
    state,
    count(1)
from
    sql_masterclass.users
group by 1
order by 2 desc;
```

Result

select state, count(1) from sql_mas

	state	count
1	NT	1,998
2	NB	1,960
3	BC	1,958
4	QC	1,929
5	NL	1,919
6	YT	1,919
7	MB	1,916
8	PE	1,906

Having

Get number of users per state ordering by biggest first only for those states with more than 1.950 users

```
select
    state,
    count
from
    (select
        state,
        count(1)
    from
        sql_masterclass.users
    group by 1)
where count>1950
order by 2 desc;
```

Result

select state, count from (select stat

	state	count
1	NT	1,998
2	NB	1,960
3	BC	1,958

```
select
    state,
    count(1)
from
    sql_masterclass.users
group by 1
having count(1)>1950
order by 2 desc;
```

Result

select state, count(1) from sql_mas

	state	count
1	NT	1,998
2	NB	1,960
3	BC	1,958

Conditional Functions

Get number of users in Pennsylvania, Washington and the rest of the country

```
select
    case
        when state='PA' then 'Pennsylvania'
        when state='WA' then 'Washington'
        else 'Rest of the Country'
    end as location,
    count(distinct userid) as users
from
    sql_masterclass.users
group by 1;
```

Result

select case when state='PA' then 'P | Enter a SQL expre

	location	users
1	Pennsylvania	497
2	Rest of the Country	49,004
3	Washington	489

Homework 1

Task:

- Get first name, last name and initials (first letter of the first name and first letter of last name) from 10 users
- Save the query as homework1.sql in GitHub in sql_masterclass folder

Tips:

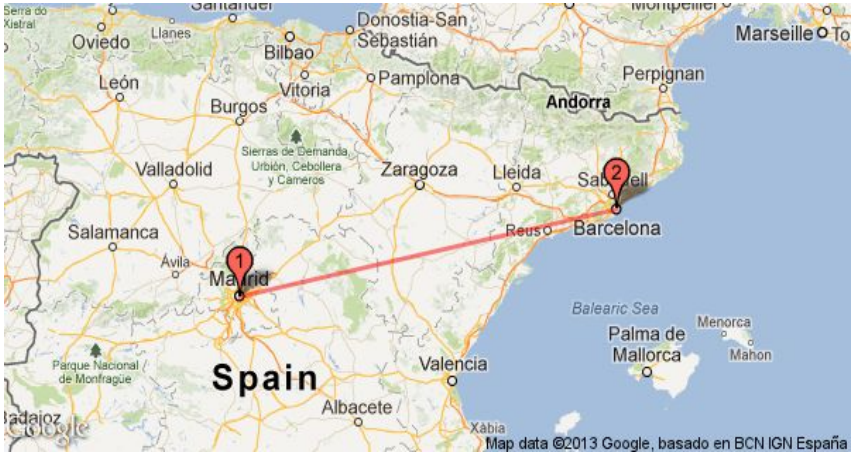
- Use String functions

	firstname	lastname	initials
1	Lars	Ratliff	LR
2	Colton	Roy	CR
3	Bruce	Beck	BB
4	Henry	Cochran	HC
5	Cody	Moss	CM
6	Ralph	Bird	RB
7	Wing	Jennings	WJ
8	Guy	Cochran	GC
9	Emerald	Chan	EC
10	Gloria	Rodriguez	GR

Homework 2

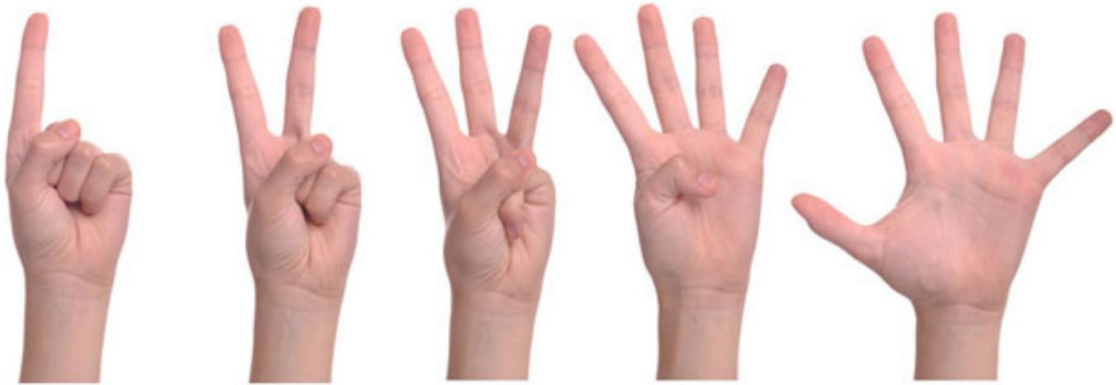
Task:

- Calculate distance in km between Madrid (41.3851 N, 2.1734 E) and Barcelona (40.4168 N, 3.7038 W)
- Save the query as homework2.sql in GitHub in sql_masterclass folder



Tips:

- Look for a math formula to calculate distance between 2 points

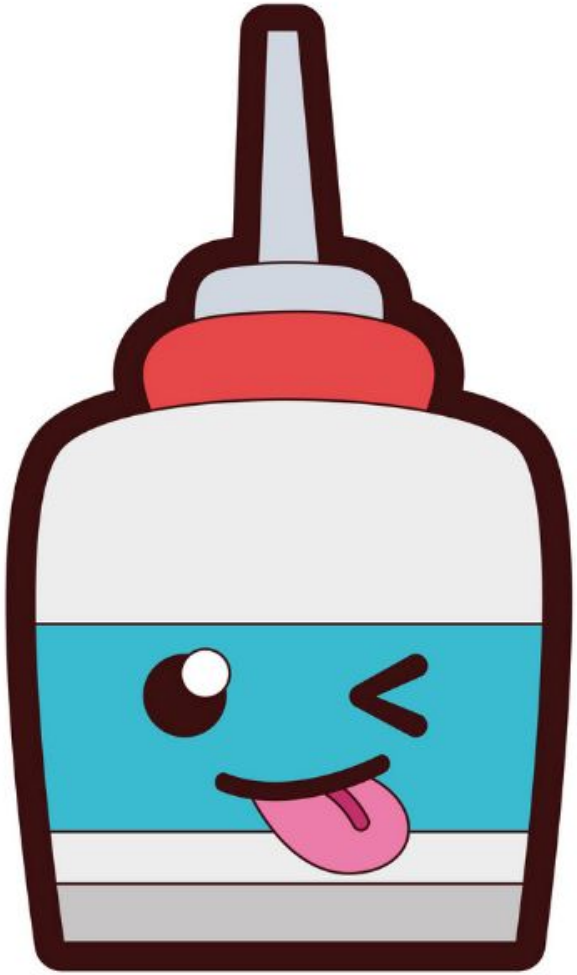

I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else


IT'S
BREAK
TIME

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. **Join-ing tables**
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

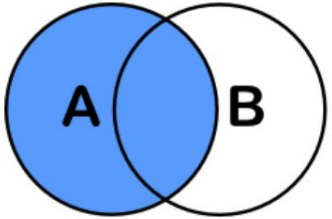
SQL MasterClass

Barcelona Data Institute

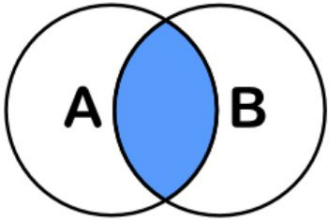


CHEATSHEET

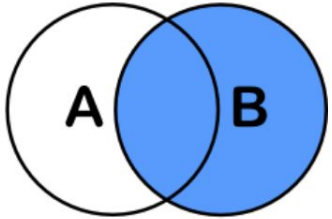
SQL JOINS



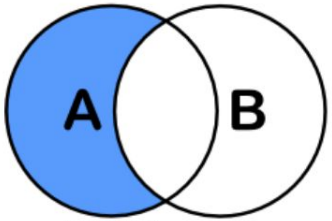

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```



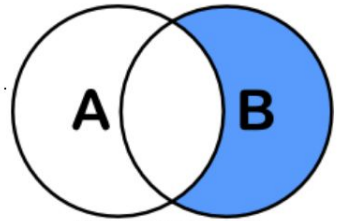

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



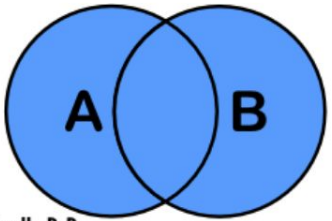

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



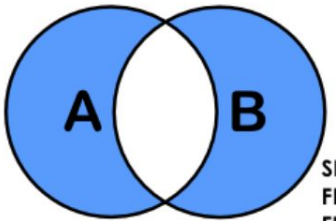

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```



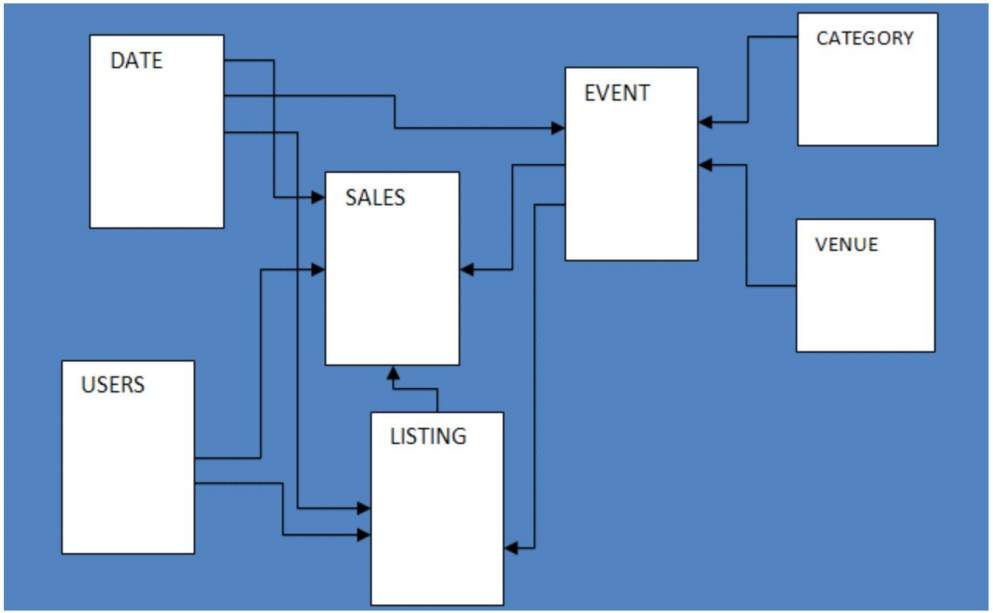

```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

our DataBase

This sample database application helps analysts track sales activity for the fictional TICKIT web site, where users buy and sell tickets online for sporting events, shows, and concerts. In particular, analysts can identify ticket movement over time, success rates for sellers, and the best-selling events, venues, and seasons. Analysts can use this information to provide incentives to buyers and sellers who frequent the site, to attract new users, and to drive advertising and promotions.



Joining (1)

Count number of events per category group

```
select
    catgroup,
    count(distinct eventid) as no_events
from
    sql_masterclass.event e
inner join
    sql_masterclass.category c on e.catid=c.catid
group by 1;
```

Result

select catgroup, count(distinct ever | Enter a SQL expression t

	catgroup	no_events
1	Shows	3,800
2	Concerts	4,998

```
select
    catgroup,
    count(distinct eventid) as no_events
from
    sql_masterclass.event e
right join
    sql_masterclass.category c on e.catid=c.catid
group by 1;
```

Result

select catgroup, count(distinct ever | Enter a SQL expression t

	catgroup	no_events
1	Shows	3,800
2	Concerts	4,998
3	Sports	0

Joining (2)

Get username of the 3 sellers that have made more commission with their sales in October 2008

```
select
    username,
    sum(commission)
from
    sql_masterclass.sales s
inner join
    sql_masterclass.users u on s.sellerid=u.userid
where
    s.saletime>='2008-10-01' and s.saletime<'2008-11-01'
group by 1
order by 2 desc
limit 3;
```

Result

select username, sum(commission) from s | Enter a SQL expression to

	username	sum
1	DVD32FRV	2,427.6
2	BEL95AGA	2,239.2
3	OSU30UYT	2,151.45

Joining (3)

How many users from 'Medford' city bought tickets for 'Spring Awakening' event

```
select
    count(distinct u.userid)
from
    sql_masterclass.event e
inner join
    sql_masterclass.sales s on e.eventid=s.eventid
inner join
    sql_masterclass.users u on u.userid=s.buyerid
where
    e.eventname='Spring Awakening' and
    u.city='Medford';
```

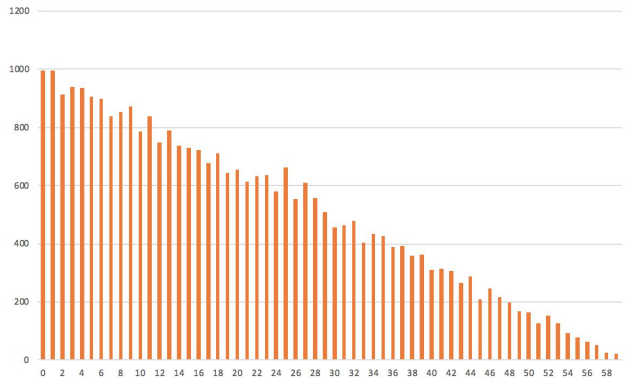
Result

select count(distinct u.userid) from sql_ma | Enter a SQL expres

	count
1	5

Joining (4)

Get distribution of how many days in advance people bought tickets for events in venues in NY state




```
select
  datediff(day, saletime, starttime),
  count(distinct salesid)
from
  sql_masterclass.sales s
inner join
  sql_masterclass.event e on s.eventid=e.eventid
inner join
  sql_masterclass.venue v on v.venueid=e.venueid
where
  v.venuestate='NY' and
  e.starttime>='2008-07-01' and
  e.starttime<'2009-01-01'
group by 1
```

Result

select datediff(day, saletime, starttime), cc | Enter a SQL expr

	date_diff	count
1	0	996
2	1	994
3	2	914
4	3	940
5	4	936
6	5	905
7	6	897
8	7	840

Homework 3

Task:

- Get total number of unsold tickets per venue in Ohio state
- Save the query as homework3.sql in GitHub in sql_masterclass folder

Tip:

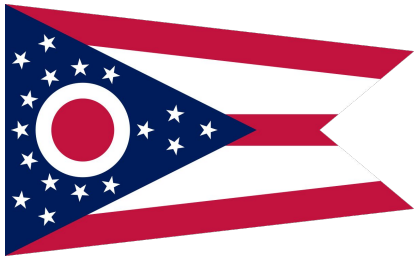
- Use subqueries to avoid duplication of rows

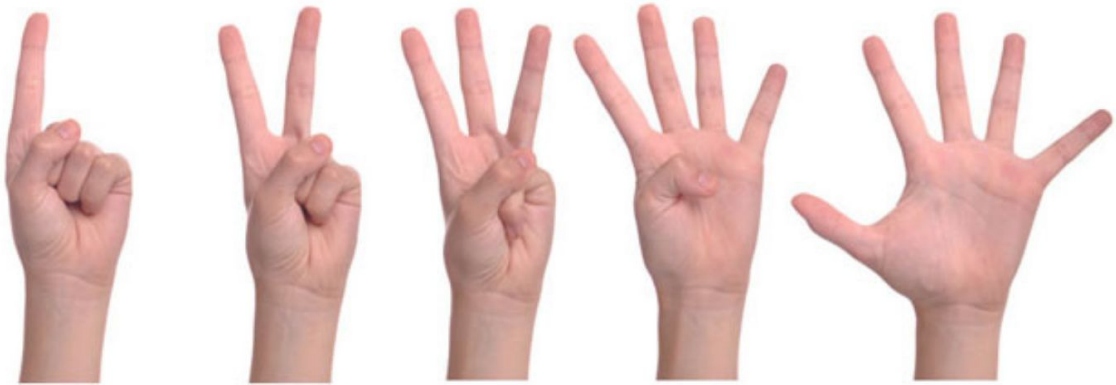
```
select
    eventname
from
    sql_masterclass.event
where
    venueid IN (select distinct venueid from sql_masterclass.venue where venuestate='OH');
```

Result

select eventname from sql_masterclass.ev | Enter a SQL expression to filter results (use Ctrl+Space)

	eventname
1	Smashing Pumpkins
2	Gogol Bordello
3	Carrie Underwood
4	Gavin DeGraw
5	Billy Joel




I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. **Window Functions**
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

Barcelona Data Institute

Window Functions in Redshift

Aggregate Functions

- AVG
- COUNT
- CUME_DIST
- FIRST_VALUE
- LAG
- LAST_VALUE
- LEAD
- MAX
- MEDIAN
- MIN
- NTH_VALUE
- PERCENTILE_CONT
- PERCENTILE_DISC
- RATIO_TO_REPORT
- STDDEV_POP
- STDDEV_SAMP (synonym for STDDEV)
- SUM
- VAR_POP
- VAR_SAMP (synonym for VARIANCE)

Ranking Functions

- DENSE_RANK
- NTILE
- PERCENT_RANK
- RANK
- ROW_NUMBER

Wow, I didn´t know SQL could do that

Window functions are a special class of analytic functions that are applied to windows of rows

function **(*expression*) OVER (**

[PARTITION BY *expr_list*]

[ORDER BY *order_list* [*frame_clause*]])

Adding a new tables to our dataset

```
create table sql_masterclass.winsales(
salesid int,
dateid date,
sellerid int,
buyerid char(10),
qty int,
qty_shipped int);

insert into sql_masterclass.winsales values
(30001, '8/2/2003', 3, 'b', 10, 10),
(10001, '12/24/2003', 1, 'c', 10, 10),
(10005, '12/24/2003', 1, 'a', 30, null),
(40001, '1/9/2004', 4, 'a', 40, null),
(10006, '1/18/2004', 1, 'c', 10, null),
(20001, '2/12/2004', 2, 'b', 20, 20),
(40005, '2/12/2004', 4, 'a', 10, 10),
(20002, '2/16/2004', 2, 'c', 20, 20),
(30003, '4/18/2004', 3, 'b', 15, null),
(30004, '4/18/2004', 3, 'b', 20, null),
(30007, '9/7/2004', 3, 'c', 30, null);
```

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

SUM

Add an extra column to "winsales" table with total "qty" per seller

```
select
    w.*,
    s.qty_by_seller
from
    sql_masterclass.winsales w
inner join
    (select
        sellerid,
        sum(qty) qty_by_seller
    from
        sql_masterclass.winsales
    group by 1) s on w.sellerid=s.sellerid;
```

Result

select *, sum(qty) over (partition by selleric | Enter a SQL expression to filter results (use Ctrl+Space)

	salesid	dateid	sellerid	buyerid	qty	qty_shipped	qty_by_seller
1	10,006	2004-01-18	1	c	10	[NULL]	50
2	10,005	2003-12-24	1	a	30	[NULL]	50
3	10,001	2003-12-24	1	c	10	10	50
4	20,002	2004-02-16	2	c	20	20	40
5	20,001	2004-02-12	2	b	20	20	40
6	30,001	2003-08-02	3	b	10	10	75
7	30,003	2004-04-18	3	b	15	[NULL]	75
8	30,007	2004-09-07	3	c	30	[NULL]	75
9	30,004	2004-04-18	3	b	20	[NULL]	75
10	40,001	2004-01-09	4	a	40	[NULL]	50
11	40,005	2004-02-12	4	a	10	10	50

```
select
    *,
    sum(qty) over (partition by sellerid) as qty_by_seller
from
    sql_masterclass.winsales;
```

Result

select *, sum(qty) over (partition by selleric | Enter a SQL expression to filter results (use Ctrl+Space)

	salesid	dateid	sellerid	buyerid	qty	qty_shipped	qty_by_seller
1	10,006	2004-01-18	1	c	10	[NULL]	50
2	10,005	2003-12-24	1	a	30	[NULL]	50
3	10,001	2003-12-24	1	c	10	10	50
4	20,002	2004-02-16	2	c	20	20	40
5	20,001	2004-02-12	2	b	20	20	40
6	30,001	2003-08-02	3	b	10	10	75
7	30,003	2004-04-18	3	b	15	[NULL]	75
8	30,007	2004-09-07	3	c	30	[NULL]	75
9	30,004	2004-04-18	3	b	20	[NULL]	75
10	40,001	2004-01-09	4	a	40	[NULL]	50
11	40,005	2004-02-12	4	a	10	10	50

First Value & Last Value

Add an extra column to "winsales" with the first "saleid" per seller

```
select
    *,
    first_value(salesid) over (partition by sellerid order by dateid)
from
    sql_masterclass.winsales;
```

Query execution failed

Reason:
[Amazon](500310) Invalid operation: Aggregate window functions with an ORDER BY clause require a frame clause;

Frame clause does not apply to ranking functions and is not required when no ORDER BY clause is used in the OVER clause for an aggregate function

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required

Frame Clause in Window Functions

It provides the ability to include or exclude sets of rows within the ordered result

{ ROWS | RANGE } BETWEEN *window_start* AND *window_end*

window_start

= { UNBOUNDED PRECEDING
| CURRENT ROW
| *constant-value* { PRECEDING | FOLLOWING } }

window_end

= { UNBOUNDED FOLLOWING
| CURRENT ROW
| *constant-value* { PRECEDING | FOLLOWING } }

First Value & Last Value (II)

Add an extra column to "winsales" with the first "saleid" per seller

```
select
    *,
    first_value(salesid) over (partition by sellerid order by dateid rows between unbounded preceding and unbounded following) as first_sale_per_seller
from
    sql_masterclass.winsales;
```

Result

Log

select *, first_value(salesid) over (partition | Enter a SQL expression to filter results (use Ctrl+Space)

	salesid	dateid	sellerid	buyerid	qty	qty_shipped	first_sale_per_seller
1	20,001	2004-02-12	2	b	20	20	20,001
2	20,002	2004-02-16	2	c	20	20	20,001
3	40,001	2004-01-09	4	a	40	[NULL]	40,001
4	40,005	2004-02-12	4	a	10	10	40,001
5	10,001	2003-12-24	1	c	10	10	10,001
6	10,005	2003-12-24	1	a	30	[NULL]	10,001
7	10,006	2004-01-18	1	c	10	[NULL]	10,001

First Value & Last Value (III)

Using transactions from "tangarana" schema, get amount in euros of the last payments made per country

```
select
  distinct user_acquisition_country,
  last_value(transaction_amount*xrate_eur)
    over (partition by user_acquisition_country order by transaction_timestamp
         rows between unbounded preceding and unbounded following) as last_amount_eur
from
  tangarana.payment_transactions t
inner join
  tangarana.users u on t.user_id=u.user_id
where
  t.transaction_type='payment';
```

Result

select distinct user_acquisition_count | Enter a SQL expression to filter results (use Ctrl+Space)

	user_acquisition_country	last_amount_eur
1	BM	41.761895
2	CI	39.336384
3	HT	42.31528
4	JO	42.01652
5	MU	27.863442
6	RU	33.044922
7	UM	48.199675
8	IE	16.979806
9	IR	40.70976
10	KY	27.863442

NO TRANSPARENCY
NO CONSENSUS

Let´s double check…

Peer Reviews are key

```
select
  transaction_timestamp,
  transaction_amount,
  xrate_eur
from
  tangarana.payment_transactions
where
  transaction_type='payment' and
  user_id IN (select user_id from tangarana.users where user_acquisition_country='BM')
order by 1 desc;
```

Result

select transaction_timestamp, transac | Enter a SQL expression to filter results (use Ctrl+Space)

	transaction_timestamp	transaction_amount	xrate_eur
1	2017-11-29 21:50:52	33.95	1.2301
2	2017-11-21 21:28:12	33.95	1.2409
3	2017-06-12 04:47:46	33.95	1.206
4	2017-05-27 17:54:59	33.95	1.1808
5	2017-05-19 14:12:22	80.95	1.1697

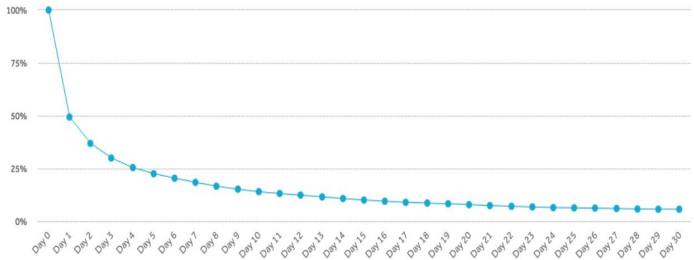
Homework 4

Task:

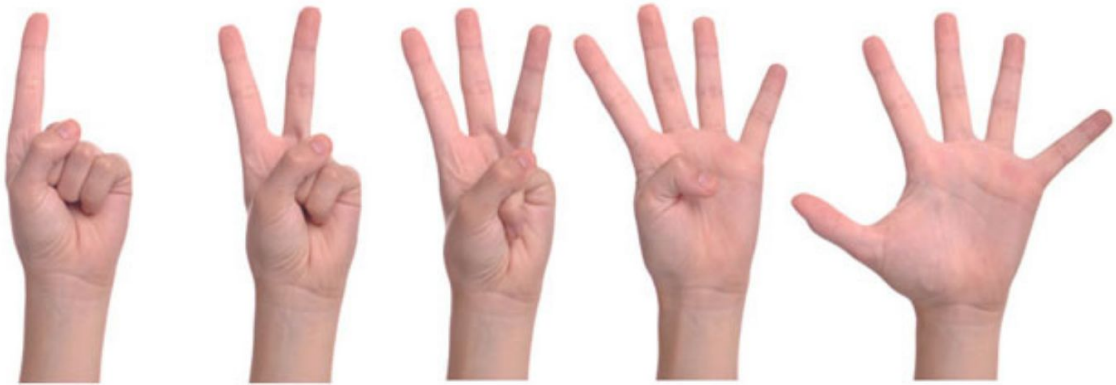
- Calculate 30 days Daily Retention for those who transact for the first time in the first half of 2017
- Transaction Retention measures how often users come back to transact after the first transaction
- Save the query as homework4.sql in GitHub in sql_masterclass folder

Tips:

- Get first users that transact for the first time in the first half of 2017
- Out of them, calculate the volume of those users that transact on day0, day2, day3, ... as shown on screenshot on the right
- Use the function datediff to calculate the difference between dates



	datediff	count
1	0	14,925
2	1	2,198
3	2	1,253
4	3	1,315
5	4	1,295
6	5	785
7	6	553
8	7	493
9	8	377
10	9	381


I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

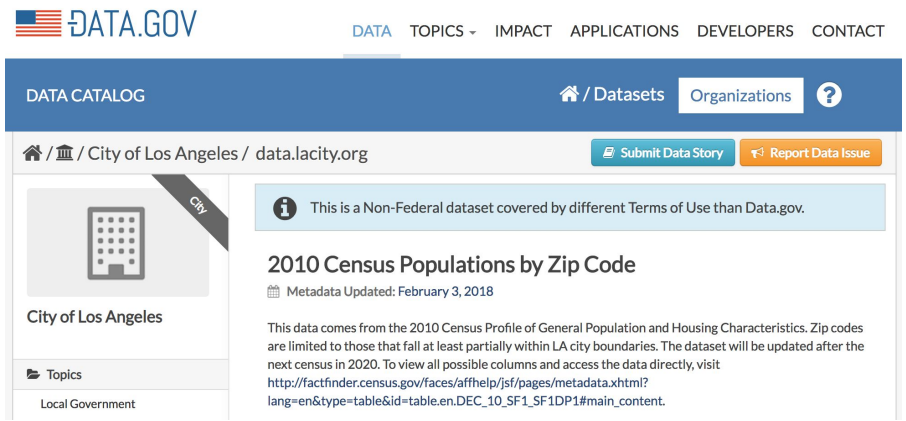
SQL MasterClass

Barcelona Data Institute

Loading Data in Redshift

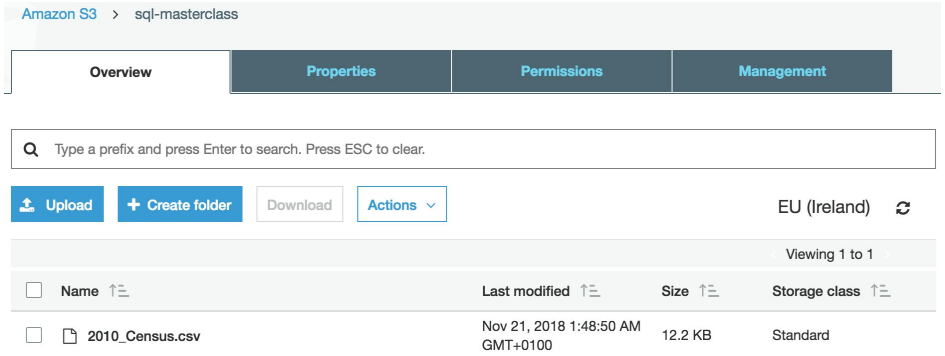
1. Get the data you want to load
2. Upload data file into AWS S3
3. Create table in RedshifT to store that data
4. Load data using copy command

https://catalog.data.gov/dataset/2010-census-populations-by-zip-code



Loading Data in Redshift (II)

1. Get the data you want to load
2. **Upload data file into AWS S3**
3. Create table in RedshifT to store that data
4. Load data using COPY command



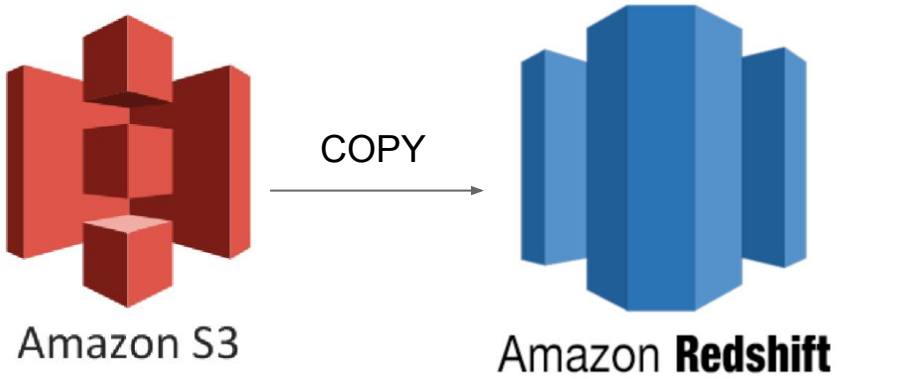
Loading Data in Redshift (III)

1. Get the data you want to load
2. Upload data file into AWS S3
3. **Create table in Redshift to store that data**
4. Load data using COPY command


```
create table sql_masterclass.census(
    "Zip Code" integer not null distkey sortkey,
    "Total Population" integer,
    "Median Age" float,
    "Total Males" integer,
    "Total Females" integer,
    "Total Households" integer,
    "Average Household Size" float);
```

Loading Data in Redshift (IV)

1. Get the data you want to load
2. Upload data file into AWS S3
3. Create table in RedshifT to store that data
4. Load data using copy command

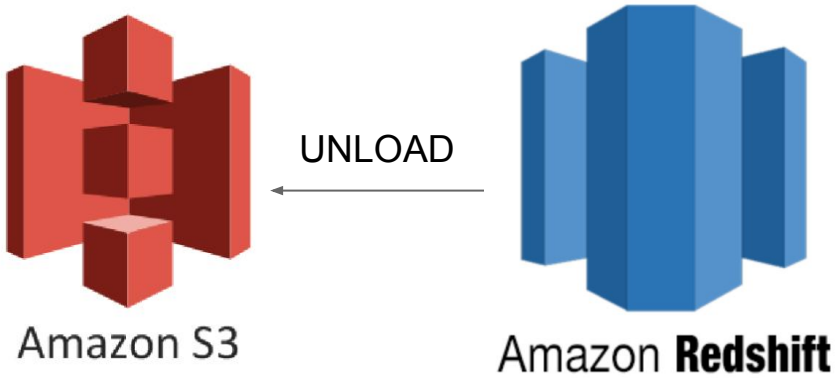



```
copy sql_masterclass.census from 's3://sql-masterclass/2010_Census.csv'
credentials 'aws_iam_role=arn:aws:iam::529806788512:role/RedshiftCopyUnload'
delimiter ',' IGNOREHEADER 1;
```

Unloading Data from Redshift

1. Save query results into S3 by

   using unload command




```
unload ('select
    sum("total population") as total_population,
    sum("total males") as total_males,
    sum("total females") as total_females
from
    sql_masterclass.census')
to 's3://sql-masterclass/unload_census.csv'
iam_role 'aws_iam_role=arn:aws:iam::529806788512:role/RedshiftCopyUnload'
DELIMITER as ','
PARALLEL OFF;
```

Homework 5

Task:

- We have data from visits and purchases from an ecommerce website in the following format:
  - "sql_masterclass.transactions"
  - "sql_masterclass.visits"
- Assign a visit to every transaction
- Save the query as homework5.sql in GitHub in sql_masterclass folder

Tips:

- Use lead or lag window functions to add a calculated column in one of the tables
- Learn how to use the function NVL in Redshift

```
select * from sql_masterclass.transactions limit 5;
```

Result

select * from sql_masterclass.transac | Enter a SQL expression to

	transaction_id	user_id	transaction_time
1	8,788,759	672,040	2016-10-03 13:52:45
2	8,708,891	649,612	2016-10-03 17:29:26
3	8,722,370	652,175	2016-10-04 04:34:10
4	8,621,989	985,347	2016-10-04 10:43:12
5	8,712,478	1,289,767	2016-10-05 09:35:57

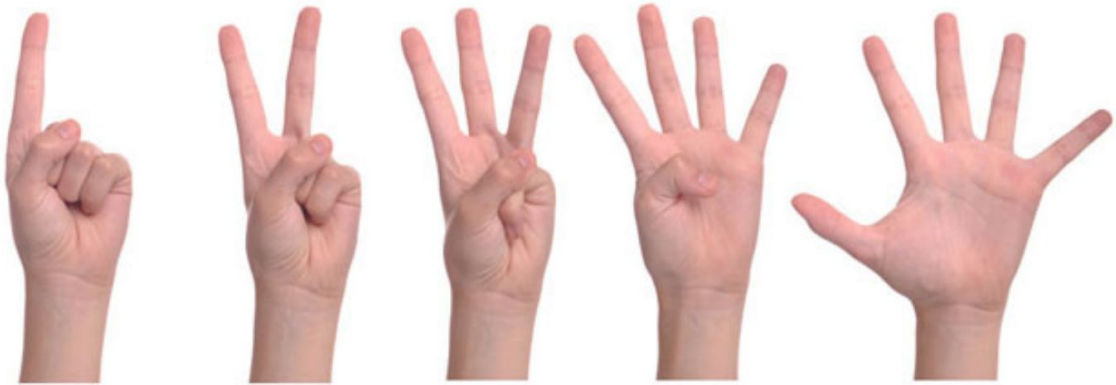
```
select * from sql_masterclass.visits limit 5;
```

Result

select * from sql_masterclass.visits lir | Enter a SQL expres

	visit_id	user_id	visit_time
1	9,994	672,040	2016-10-03 13:28:45
2	432	1,289,767	2016-10-05 09:11:57
3	5,184	665,882	2016-10-10 20:20:17
4	8,053	1,064,435	2016-10-14 09:38:14
5	9,020	666,534	2016-10-17 11:23:00

	user_id	transaction_id	visit_id	transaction_time	visit_time
1	652,910	8,727,907	7,949	2016-10-06 01:33:50	2016-10-06 01:09:50
2	657,037	8,564,577	6,402	2016-10-08 12:59:28	2016-10-08 12:35:28
3	657,037	8,767,214	7,357	2016-12-24 15:01:03	2016-12-02 22:37:03
4	665,882	8,531,769	5,184	2016-10-10 20:44:17	2016-10-10 20:20:17
5	666,302	8,506,137	6,263	2016-10-05 18:48:17	2016-10-05 18:24:17
6	666,534	8,700,906	9,020	2016-10-17 11:47:00	2016-10-17 11:23:00
7	669,008	8,563,949	1,996	2016-10-17 00:33:24	2016-10-15 09:49:24
8	671,735	8,520,433	6,435	2016-12-09 05:06:32	2016-12-07 14:22:32
9	671,998	8,710,930	7,796	2016-10-23 08:26:36	2016-10-23 08:02:36
10	672,640	8,481,493	2,959	2016-10-21 03:18:29	2016-10-19 12:34:29


I don't understand at all
I need to go over this again
I think I got it, but am not completely comfortable
I got it
I can explain it to someone else

1. Introduction to Databases
2. Our Data World
3. Amazon Redshift vs Google BigQuery
4. How to create a Database
5. Basic SQL
6. Join-ing tables
7. Window Functions
8. How to Export and Import data in Redshift
9. QA

SQL MasterClass

Barcelona Data Institute


YOUR
FEEDBACK
MATTERS!

Deepen Learning

- With your super pen and in one post-it, write (2 min):
  - Your 2 main learnings/takeaways
  - Your 2 main challenges for the week
- Find a partner that is not your table mate
- Share and help/support each other (3 min)
- Find a new partner, repeat (3 min)
  - Look at what is different, after the first share

Data Analyst Progress

12%

THANK YOU

SQL MasterClass

Barcelona Data Institute