INTRO TO

DATA ANALYTICS

USING SQL

BLACK HILLS
Information Security

WEBCAST

OVERVIEW

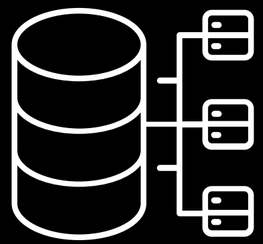
What we'll learn in this webcast

1. **SQL**
   What is SQL and why use it?

2. **Databases**
   Types of databases.

3. **Data Exploration**
   Exploratory Data Analysis using SQL.

4. **SELECT**
   Anatomy and analysis of a SELECT statement.



Ethan Robish

About Me

- Pentester
- Developer
- BHIS SOC
  - Threat Hunter
  - Detection Engineer
  - (Assistant to the) Platform Engineer

BHIS SOC

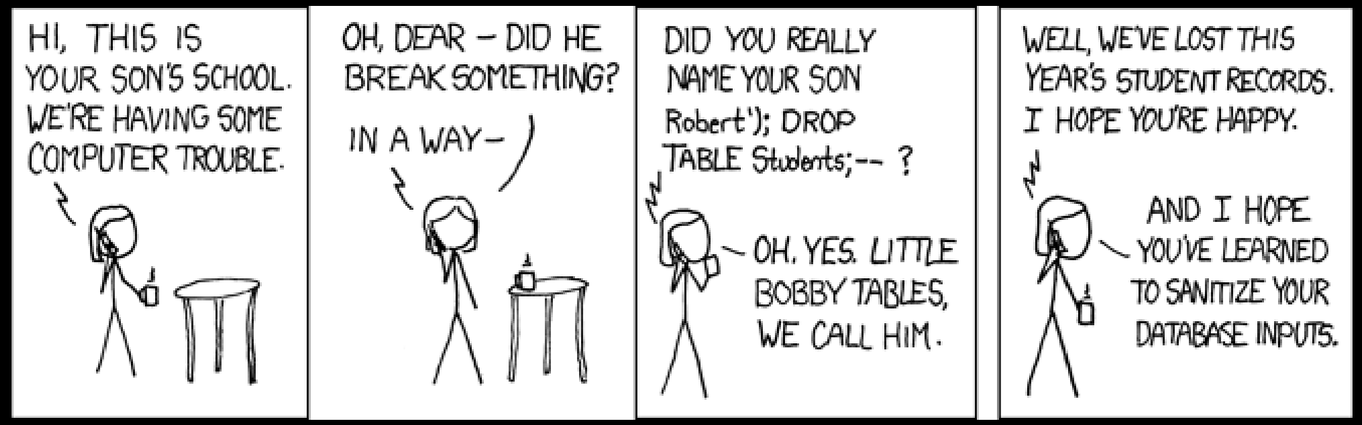
About Us

1. **Continuous Monitoring and Alerting**
   24/7 security log ingestion, risk-based alerting
2. **Adversary Emulation**
   From testing our own rules to pentesting your environment
3. **Forensics and Incident Response**
   Retainer included to ensure we can provide immediate help
4. **Extras**
   Tabletops, Active Directory, Threat Intelligence, and more

SQL

What is SQL?

What do you think of?



https://xkcd.com/327/

1 **DATABASE?**

MS SQL, MySQL, Postgres

2 **INJECTION?**

Every red team's favorite type of SQL

3 **LANGUAGE?**

Structured Query Language

Types of Queries

SQL statements can be categorized based on the way they interact with data.

Acronym	Name	Examples
DDL	Data Definition Language	CREATE, DROP, ALTER, TRUNCATE
DQL	Data Query Language	SELECT
DML	Data Manipulation Language	INSERT, UPDATE, DELETE
DCL	Data Control Language	GRANT, REVOKE
TCL	Transaction Control Language	COMMIT, ROLLBACK

Why use SQL?

Why not Excel, GNU utils, Python, etc. instead?

1 **UBIQUITOUS**
ISO/ANSI Standard

2 **DECLARATIVE**
Say what you want not how to get it

3 **INTEROPERABLE**
Bring results into any programming language

4 **POWERFUL**
Aggregations, JOINS, and more

MariaDB
PostgreSQL
MySQL
Microsoft SQL Server
Velociraptor
osquery
elastic
OpenSearch

Why NOT use SQL?

When it's the wrong tool for the job

1. **UNSTRUCTURED DATA**
   Always a pain but especially with SQL

2. **EXISTING DATA STORE**
   Non-SQL language

3. **COMPLEX* ANALYSIS**
   SQL is not meant to be a full-fledged programming language

Easy to Learn, Hard to Master

https://x.com/largedatabank/status/1559651463919452161

Databases

Database Types

Online Transactional Processing (OLTP) vs Online Analytical Processing (OLAP)

OLTP - Transactional

Great for updates and fast single record retrieval.

- Data Model: Normalized
- Data Size: Smaller
- Common Traits: Row store, fast updates of few records

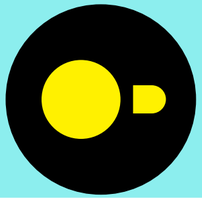
Examples: MSSQL, MariaDB, Postgres, SQLite

OLAP - Analytical

Great for aggregations and large amounts of data.

- Data Model: Denormalized
- Data Size: Larger
- Common Traits: Columnar store, high throughput ingest

Examples: ClickHouse, DuckDB, Spark, Snowflake



DuckDB

"DuckDB is a portable, open source, fast, in-process, analytical database system."

FAST & SIMPLE

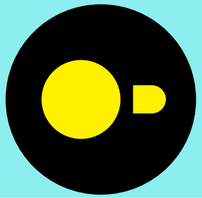
Single binary, no server, parallel vectorized execution,
larger-than-memory workloads, columnar store

STABLE & FEATUREFUL

June 2019 - First release
June 2024 - v1.0 release stable API & file format
Large function library, advanced SQL features, friendlier SQL

EXTENSIONS

Reads and writes many file formats, read directly from HTTP or S3, 3rd party APIs, custom algorithms,
language dialects, LLM integration, WebAssembly



DuckDB

Running commands

```
$ duckdb

v1.1.2 f680b7d08f
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent
database.
D SELECT 'Hello' AS msg, version() AS version;
```

msg	version
varchar	varchar
Hello	v1.1.2

```
$ duckdb -csv -s "SELECT 'Hello' AS msg, version() AS vers"
msg,vers
Hello,v1.1.2

$ duckdb -json -s "SELECT 'Hello' AS msg, version() AS vers"
[{"msg":"Hello","vers":"v1.1.2"}]

$ duckdb -box -s "SELECT 'Hello' AS msg, version() AS vers"
```

msg	vers
Hello	v1.1.2

Formatting

Which is easier to read?

```
select name, id from original where name
not in (select name from current) and name
not like 'Short %' order by name limit 10;
```

```
SELECT
    name,
    id
FROM original
WHERE
    name NOT IN (SELECT name FROM current)
    AND name NOT LIKE 'Short %'
ORDER BY name
LIMIT 10;
```

Data Exploration

The Dataset

Honeypot network data from 2013. Filename : marx-geo.csv

Time Series

Every entry has a timestamp

Network and Geo Info

IP, port, protocol, geolocation

Data Driven Security

https://datadrivensecurity.info/blog/posts/2014/Jan/blander-part1/
https://datadrivensecurity.info/blog/posts/2014/Jan/blander-part2/
Cleaned using: `sed -i 's/Washington, D.C./Washington D.C./g' marx-geo.csv`

Data Exploration

SELECT

```
SELECT * FROM 'marx-geo.csv';
```

datetime timestamp	host varchar	…	latitude double	longitude double
2013-03-03 21:53:59	groucho-oregon	…	28.55	115.9333
2013-03-03 21:57:01	groucho-oregon	…	51.0	9.0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
2013-09-08 05:46:42	groucho-oregon	…	26.0614	119.3061
2013-09-08 05:48:42	groucho-sa	…	43.6525	-79.3686

451581 rows (4 shown) 15 columns (4 shown)

Data Exploration

LIMIT

```
SELECT * FROM 'marx-geo.csv';

SELECT * FROM 'marx-geo.csv' LIMIT 8;
```

host varchar	srcstr varchar	dpt int64	country varchar
groucho-oregon	61.131.218.218	1433	China
groucho-oregon	80.86.82.58	5060	Germany
groucho-oregon	175.180.184.106	1080	Taiwan
groucho-us-east	50.45.128.28	1900	United States
groucho-singapore	213.215.43.23	80	France
groucho-tokyo	198.20.69.98	2323	United States
groucho-oregon	222.89.164.247	1433	China
groucho-singapore	222.214.218.109	3306	China

Data Exploration

DESCRIBE

```
SELECT * FROM 'marx-geo.csv';
SELECT * FROM 'marx-geo.csv' LIMIT 8;

DESCRIBE SELECT * FROM 'marx-geo.csv';
```

column_name varchar	column_type varchar	null varchar
datetime	TIMESTAMP	YES
host	VARCHAR	YES
proto	VARCHAR	YES
type	BIGINT	YES
spt	BIGINT	YES
dpt	BIGINT	YES
srcstr	VARCHAR	YES
cc	VARCHAR	YES
country	VARCHAR	YES
locale	VARCHAR	YES
15 rows		3 columns

Data Exploration

SUMMARIZE

```
SELECT * FROM 'marx-geo.csv';
SELECT * FROM 'marx-geo.csv' LIMIT 8;
DESCRIBE SELECT * FROM 'marx-geo.csv';

SUMMARIZE SELECT * FROM 'marx-geo.csv';
```

column_name	column_type	min	max	approx_unique	avg	null_percentage
datetime	TIMESTAMP	2013-03-03 21:53:59	2013-09-08 05:55:13	332707		0.00
host	VARCHAR	groucho-eu	zeppo-norcal	10		0.00
proto	VARCHAR	ICMP	UDP	3		0.00
type	BIGINT	0	13	7	7.514895896096941	90.08
dpt	BIGINT	0	65500	4462	6684.258212257541	9.92
srcstr	VARCHAR	1.0.0.38	99.99.34.249	80328		0.00

SELECT Query

Querying Data

SELECT

```sql
-- literal value
SELECT 'Welcome to the webcast';

-- assign column name / alias
SELECT 'Welcome to the webcast' AS msg;

-- return all columns
SELECT * FROM ...;

-- return specified columns
SELECT datetime, host, type FROM ...;

-- exclude specified columns
SELECT * EXCLUDE (type, datetime) FROM ...;
```

```sql
-- return columns with special characters
SELECT "Full Name", "Null" FROM ...;

-- return total number of rows
SELECT count(*) FROM ...;

-- perform math or runtime operations
SELECT a + b AS sum FROM ...;

-- return unique rows
SELECT DISTINCT srcstr, dpt FROM ...;
```

Querying Data From Files

FROM

```sql
-- insert data into a table
INSERT INTO tbl VALUES (1, 2, 3);

-- query from table
SELECT * FROM tbl;
```

```sql
-- FROM first syntax
FROM tbl;

-- auto-detect type from file extension
SELECT * FROM 'flights.csv';
SELECT * FROM read_csv('flights.csv');

-- explicitly tune input settings
SELECT * FROM read_csv('flights.csv',
                       union_by_name = true);

-- same with json, parquet, others
SELECT * FROM 'flights.json';
SELECT * FROM read_json('flights.json');
```

Organizing Data

ORDER BY

```
-- sort by datetime field, oldest to newest
SELECT * FROM 'marx-geo.csv'
ORDER BY datetime;

SELECT * FROM 'marx-geo.csv'
ORDER BY datetime ASC;

-- sort by newest to oldest
SELECT * FROM 'marx-geo.csv'
ORDER BY datetime DESC;

-- multi-level sorting
SELECT datetime, srcstr, dpt FROM 'marx-geo.csv'
ORDER BY dpt ASC, src DESC;
```

datetime	srcstr	dpt
2013-08-18 17:02:28	213.157.218.54	0
2013-09-01 03:22:28	201.238.247.11	0
2013-09-01 03:22:29	201.238.247.11	0
2013-08-22 17:00:15	80.82.64.242	0
2013-07-29 19:09:58	219.149.138.230	1
2013-03-06 18:11:23	219.149.138.230	1
2013-08-31 20:11:39	218.87.16.135	1
2013-04-12 08:25:16	218.87.16.135	1
2013-05-07 03:23:32	218.87.16.135	1
2013-04-18 12:22:57	118.249.87.13	1
2013-04-18 12:22:58	113.240.176.175	1

Filtering Data: Part 1

WHERE

```
-- strings are single quoted, = for equality
SELECT * FROM 'marx-geo.csv'
WHERE country = 'United States';

-- AND, OR, NOT
SELECT * FROM 'marx-geo.csv'
WHERE NOT country = 'United States';

SELECT * FROM 'marx-geo.csv'
WHERE country = 'China'
   OR country = 'Russia';

SELECT * FROM 'marx-geo.csv'
WHERE dpt = 1433 AND
  (country = 'China' OR country = 'Russia');
```

```
-- IN lets you check a set of values
SELECT * FROM 'marx-geo.csv'
WHERE dpt IN (135, 137, 445);

-- BETWEEN allows using a range
SELECT * FROM 'marx-geo.csv'
WHERE dpt BETWEEN 1 AND 1025;

-- BETWEEN works for timestamps
SELECT * FROM 'marx-geo.csv'
WHERE datetime BETWEEN '2013-08-01'
                   AND '2013-09-01';
```

Aggregating Data

Aggregate Functions

```
-- hint: implicit GROUP BY ALL
SELECT min(datetime), max(datetime), count(*) FROM 'marx-geo.csv';
```

min(datetime) timestamp	max(datetime) timestamp	count_star() int64
2013-03-03 21:53:59	2013-09-08 05:55:13	451581

Aggregating Data: Error

Aggregate Functions

```
SELECT min(datetime), max(datetime), count(*) FROM 'marx-geo.csv';

-- hint: GROUP BY is required when non-aggregate present
SELECT host, count(*) FROM 'marx-geo.csv';   -- Error!
```

```
Binder Error: column "host" must appear in the GROUP BY clause
or must be part of an aggregate function.

Either add it to the GROUP BY list, or use "ANY_VALUE(host)"
if the exact value of "host" is not important.

LINE 1: SELECT host, count(*) FROM 'marx-geo.csv';
               ^
```

Aggregating Data

GROUP BY

```
-- Error!
SELECT host, count(*) FROM 'marx-geo.csv';

-- GROUP BY is required
-- when non-aggregate present
SELECT host, count(*) FROM 'marx-geo.csv'
GROUP BY host;
```

host varchar	count_star() int64
groucho-oregon	94076
groucho-norcal	24566
groucho-eu	23954
groucho-sa	24316
groucho-us-east	31779
zeppo-norcal	24094
groucho-sydney	24456
groucho-singapore	78151
groucho-tokyo	126189

Aggregating Data

Implicit GROUP BY ALL

```
-- implicit GROUP BY ALL
SELECT min(datetime), max(datetime), count(*) FROM 'marx-geo.csv';
-- both commands are equivalent
SELECT min(datetime), max(datetime), count(*) FROM 'marx-geo.csv' GROUP BY ALL;
```

min(datetime) timestamp	max(datetime) timestamp	count_star() int64
2013-03-03 21:53:59	2013-09-08 05:55:13	451581

Filtering Data: Part 2

```
SELECT host, count(*) FROM 'marx-geo.csv'
GROUP BY ALL;

SELECT host, count(*) AS cnt
FROM 'marx-geo.csv'
WHERE cnt > 100000   -- Error!
GROUP BY ALL;
```

```
Binder Error: WHERE clause cannot contain
aggregates!
LINE 1: SELECT host, count(*) AS cnt
                             ^
```

Filtering Data: Part 2

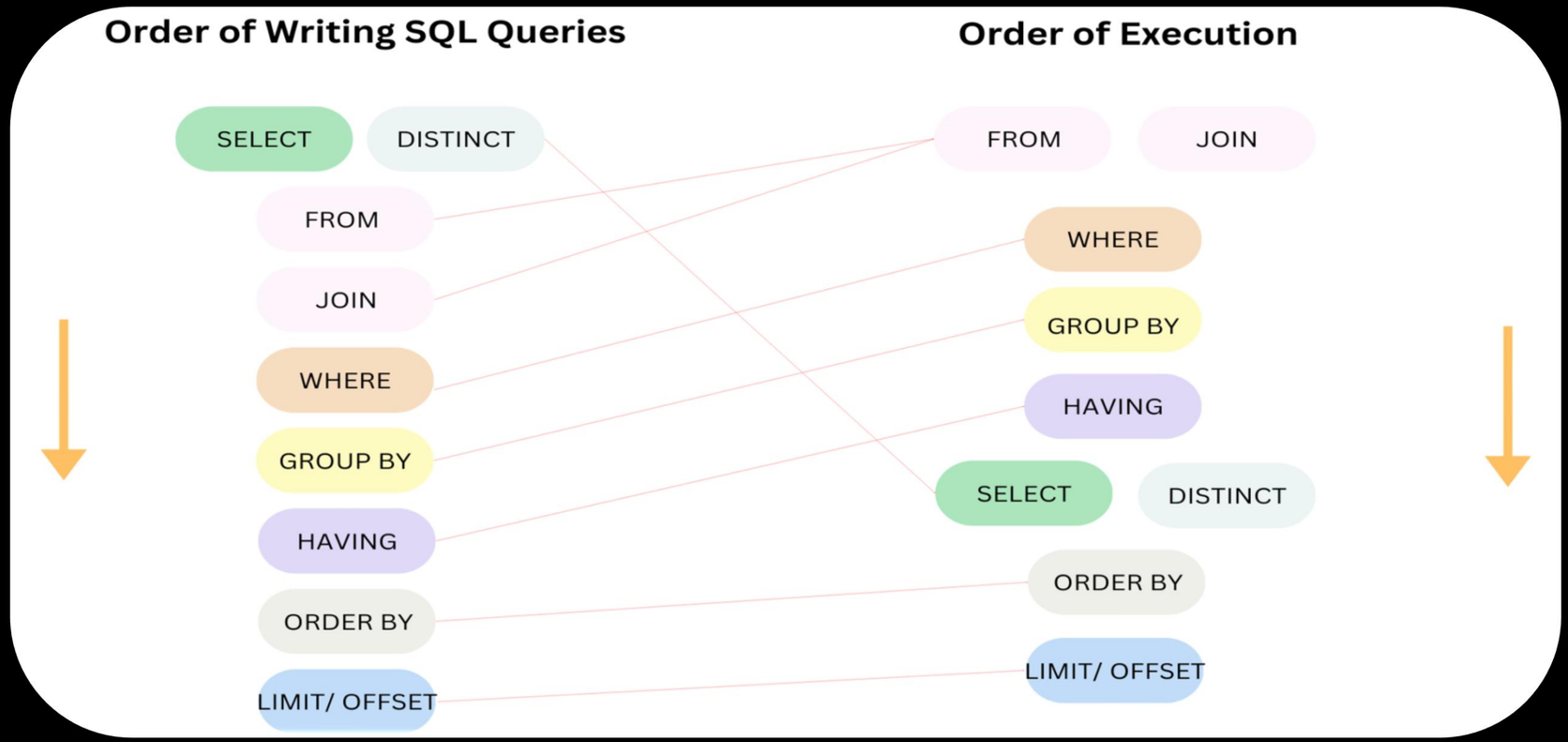
HAVING

```
SELECT host, count(*) FROM 'marx-geo.csv'
GROUP BY ALL;

SELECT host, count(*) AS cnt
FROM 'marx-geo.csv'
WHERE cnt > 100000   -- Error!
GROUP BY ALL;

SELECT host, count(*) AS cnt
FROM 'marx-geo.csv'
GROUP BY ALL
HAVING cnt > 100000; -- Correct
```

host varchar	cnt int64
groucho-tokyo	126189

Order of Execution

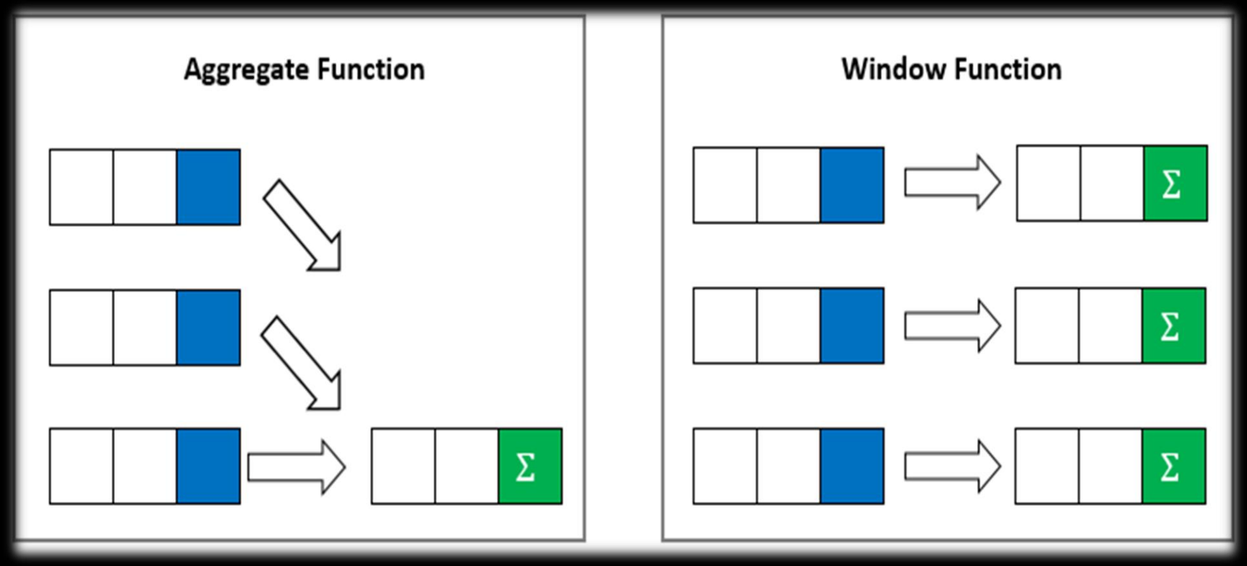
Lexical vs Logical Ordering



https://www.thedataschool.co.uk/le-luu/order-of-operations-and-order-of-execution-in-sql/

Window Functions

← Not *that* Windows



https://www.sqltutorial.org/sql-window-functions/

1. **Running Total**
   Each value builds on the previous
2. **Rolling Average**
   Values expire as they fall out of the frame
3. **Percentage of the Whole**
   Current value portion of the total
4. **Top Values per Group**
   Like a LIMIT but per paritition

Window Functions

1. **OVER**
   Aggregate function

2. **Partitioning**
   Creates independent windows

3. **Ordering**
   Sorts data within the window

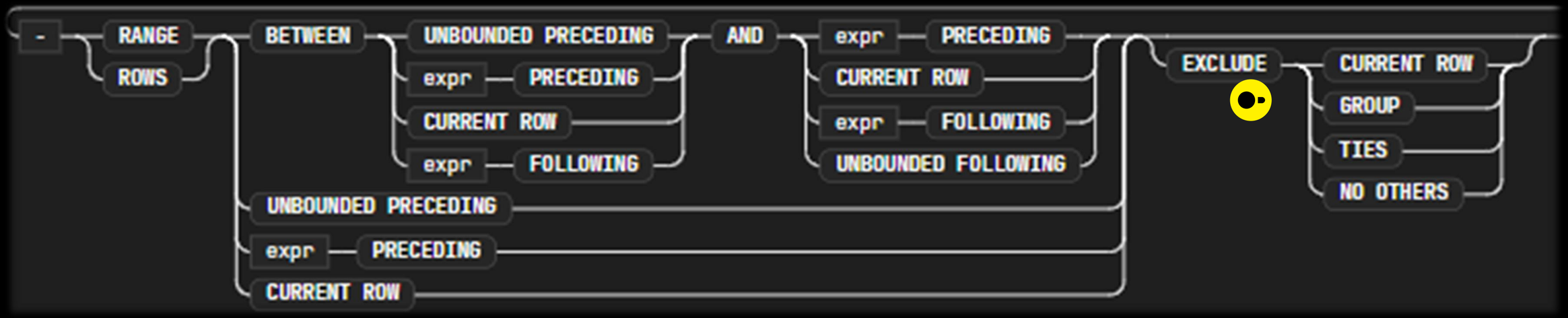
4. **Frame**
   Sliding subset of rows within the window

```
-- syntax
window_function_name ( expression ) OVER (
    [ PARTITION BY clause ]
    [ ORDER BY clause ]
    [ ROWS/RANGE frame clause ]
)
```

```
-- example: running total
SELECT host, spt,
  sum(spt) OVER (
    PARTITION BY host
    ORDER BY datetime ASC
    ROWS BETWEEN UNBOUNDED PRECEDING
             AND CURRENT ROW
  ) AS running_total
FROM 'marx-geo.csv'
ORDER BY datetime;
```

Window Functions

Frame Clause
Syntax



https://duckdb.org/docs/sql/functions/window_functions#syntax

Window Functions

Running Total

```sql
-- running total
SELECT host, spt,
  sum(spt) OVER (
    PARTITION BY host
    ORDER BY datetime ASC
) AS running_total
FROM 'marx-geo.csv'
ORDER BY datetime;
```

host	spt	running_total
groucho-oregon	6000	6000
groucho-oregon	5270	11270
groucho-oregon	2489	13759
groucho-singapore	56577	56577
groucho-tokyo	32628	32628
groucho-oregon	6000	19759
groucho-singapore	6000	62577
groucho-oregon	6000	25759
groucho-singapore	6000	68577
groucho-tokyo	6000	38628
.	.	.

Window Functions

Rolling Average

```
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Rolling Average

```sql
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Rolling Average

```
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Rolling Average

```sql
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Rolling Average

```sql
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Rolling Average

```
WITH sample_data AS (
  SELECT 10 * (range+1) AS val
  FROM range(10)
)
-- rolling average
SELECT val,
  avg(val) OVER frame AS rolling_avg,
  sum(val) OVER frame AS rolling_sum,
  count(val) OVER frame AS rolling_cnt
FROM sample_data
WINDOW frame AS (
  ROWS 2 PRECEDING
);
```

val	rolling_avg	rolling_sum	rolling_cnt
10	10.0	10	1
20	15.0	30	2
30	20.0	60	3
40	30.0	90	3
50	40.0	120	3
60	50.0	150	3
70	60.0	180	3
80	70.0	210	3
90	80.0	240	3
100	90.0	270	3

Window Functions

Combining GROUP BY and OVER to calculate percentage of the whole

```sql
-- display percentage of total traffic
-- from each country
SELECT
  country,
  -- x * 100 / sum(x)
  count(*) * 100.0 / sum(count(*))
    OVER () AS pct
FROM 'marx-geo.csv'
GROUP BY country
ORDER BY pct DESC
LIMIT 10;
```

country	pct
China	42.38309406285916
United States	19.9259933433869
Japan	3.809726272805986
Iran	2.8880754504728943
Taiwan	2.6901043223696304
Netherlands	2.378089423602853
India	2.0855616157455694
South Korea	2.062974305827747
Vietnam	1.733022425655641
Russia	1.5968342335040668

Filtering Data: Part 3

How do we filter based on window functions? WHERE?

```sql
-- display percentage of total traffic
-- from each country
SELECT
  country,
  count(*) * 100.0 / sum(count(*))
    OVER () AS pct
FROM 'marx-geo.csv'
WHERE pct > 10   -- Error!
GROUP BY country
ORDER BY pct DESC;
```

```
Binder Error: WHERE clause cannot contain
aggregates!

LINE 3:   count(*) * 100.0 / sum(count(*))
```

Filtering Data: Part 3

How do we filter based on window functions? HAVING?

```sql
-- display percentage of total traffic
-- from each country
SELECT
  country,
  count(*) * 100.0 / sum(count(*))
    OVER () AS pct
FROM 'marx-geo.csv'
GROUP BY country
HAVING pct > 10   -- Error!
ORDER BY pct DESC;
```

```
Binder Error: HAVING clause cannot contain
window functions!

LINE 3:   count(*) * 100.0 / sum(count(*))
```

Filtering Data: Part 3

How do we filter based on window functions? Subquery? CTE?

```sql
WITH results AS (
  SELECT
    country,
    count(*) * 100.0 / sum(count(*))
      OVER () AS pct
  FROM 'marx-geo.csv'
  GROUP BY country
)
SELECT country, pct
FROM results
WHERE pct > 10
ORDER BY pct DESC;
```

country varchar	pct double
China	42.38309406285916
United States	19.9259933433869

Filtering Data: Part 3

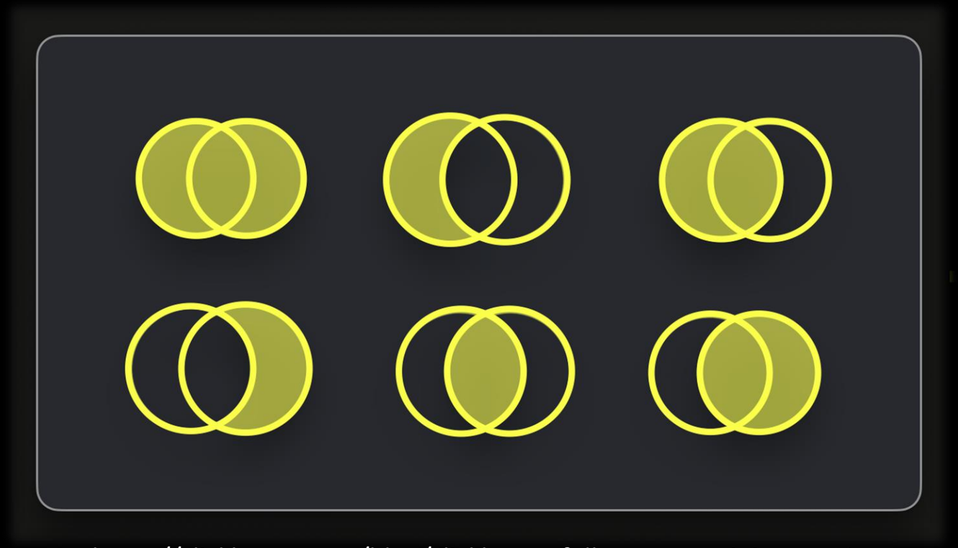
How do we filter based on window functions? QUALIFY?

```sql
SELECT
  country,
  count(*) * 100.0 / sum(count(*))
    OVER () AS pct
FROM 'marx-geo.csv'
GROUP BY country
QUALIFY pct > 10
ORDER BY pct DESC;
```

country varchar	pct double
China	42.38309406285916
United States	19.9259933433869

Joining Data

Putting data together from two or more tables

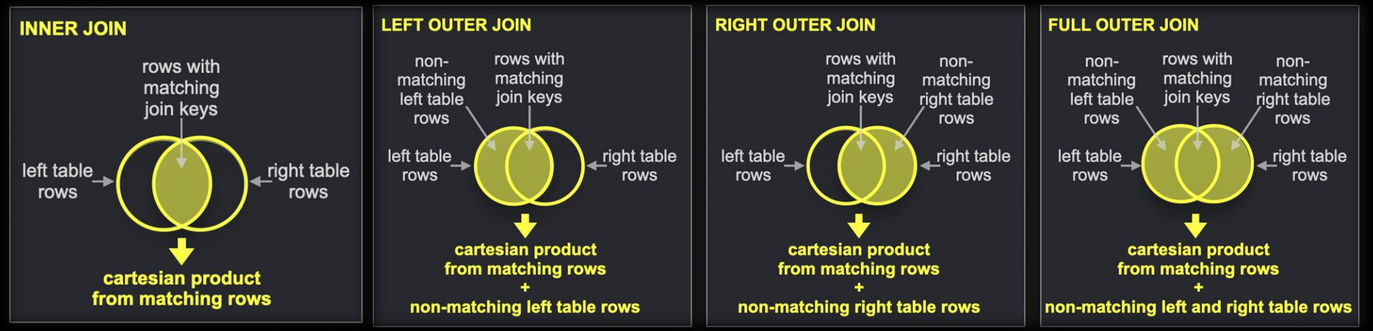


https://clickhouse.com/blog/clickhouse-fully-supports-joins-part1

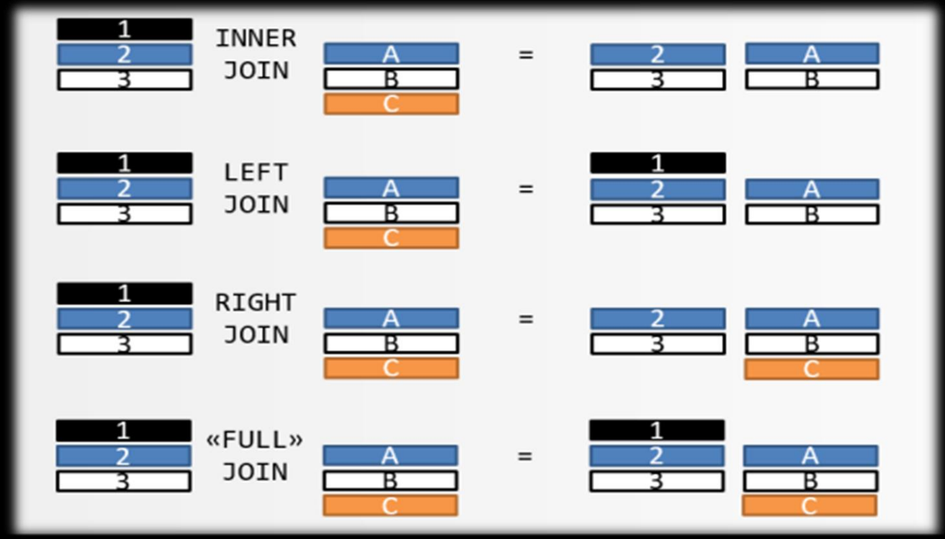
1. **Enrichment**
   Using as a lookup table
2. **Filtering**
   Reduce results based on data in another table
3. **Expanding**
   Create combinations of values

Types of Joins

1. **INNER JOIN**
   INNER is an optional keyword
2. **OUTER JOIN**
   LEFT, RIGHT, FULL
   OUTER is an optional keyword
3. **CROSS JOIN**
   Cartesian product
4. **Special Purpose**
   NATURAL, POSITIONAL, SEMI, ANTI, ASOF, LATERAL



https://clickhouse.com/blog/clickhouse-fully-supports-joins-part1



https://blog.jooq.org/say-no-to-venn-diagrams-when-explaining-joins/

Joining Data

Enrichment

```
SELECT
  marx.datetime,  -- excluded for brevity ⟶
  marx.srcstr ,   -- excluded for brevity ⟶
  marx.proto,
  marx.dpt,
  iana."Service Name" AS service,
  iana.Description AS description
FROM 'marx-geo.csv' AS marx
LEFT JOIN 'iana-port-numbers.csv' AS iana
  -- protocol is uppercase in marx, lower in iana
  ON lower(marx.proto) = iana."Transport Protocol"
    -- typecast port to string to match iana type
    AND marx.dpt::VARCHAR = iana."Port Number";
```

proto varchar	dpt int64	service varchar	description varchar
ICMP			
TCP	1723	pptp	pptp
TCP	8000	irdmi	iRDMI
TCP	3127	ctx-bridge	CTX Bridge Port
UDP	137	netbios-ns	NETBIOS Name Service
TCP	8001	vcom-tunnel	VCOM Tunnel
TCP	143	imap	Internet Message Access Protocol
TCP	11211	memcache	Memory cache service
UDP	16368	netserialext4	Network Serial Extension Ports Four
UDP	16000		Reserved
TCP	222	rsh-spx	Berkeley rshd with SPX auth
UDP	5070	vtsas	VersaTrans Server Agent Service
UDP	4131	stars	Global Maintech Stars
TCP	6670	vocaltec-gold	Vocaltec Global Online Directory
·	·	·	·
·	·	·	·
·	·	·	·

Thank You!

```
/* He's making a database,
   He's checking it twice, */
   SELECT * FROM contacts
   WHERE behavior = 'nice';
/* SQL Clause is coming to town */
```