hi, i’m lloyd

1987

1994

2003

2012

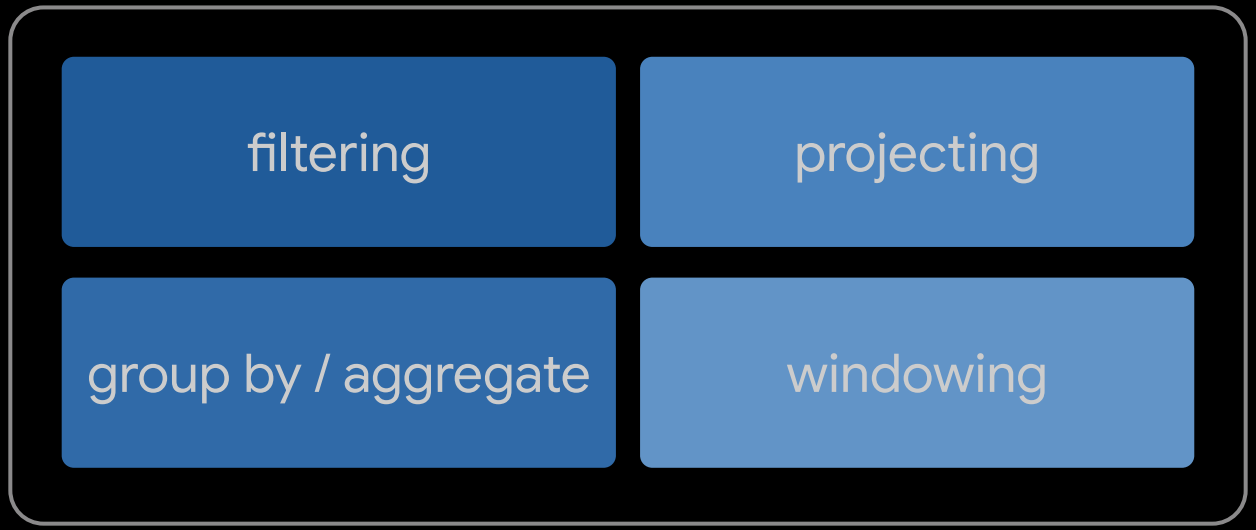
2023

Data is Rectangular

(and other limiting misconceptions)

Humans think in rectangular calculations

Operations within the Rectangle



In SQL Joins, produce a new rectangle

orders

order_id	order_date	shipping_cost	user_id
1	2022-01-01	2	1
2	2022-01-01	3	2
3	2022-01-02	1	1
4	2022-01-02	2	3

orders

order_id	order_date	shipping_cost	user_id
1	2022-01-01	2	1
2	2022-01-01	3	2
3	2022-01-02	1	1
4	2022-01-02	2	3

order_items

item_id	order_id	item	price
1	1	Chocolate	2
2	1	Twizzler	1
3	2	Chocolate	2
4	2	M and M	1
5	3	Twizzler	1
6	4	Fudge	3
7	4	Skittles	1

Let’s measure two things, from sales...

total_shipping

total_revenue

total_shipping

```
SELECT
   SUM(shipping_cost) AS total_shipping
FROM ‘orders.csv
```

total_shipping
8

total_revenue

```
SELECT
   SUM(price) AS total_revenue
FROM ‘items.csv’;
```

total_revenue
11

total_shipping by date

```
SELECT
   order_date,
   SUM(shipping_cost) AS total_shipping
FROM ‘orders.csv’
GROUP BY 1
ORDER BY 1
```

order_date	total_shipping
2022-01-01	5
2022-01-02	3

total_revenue by date

```
SELECT
   order_date,
   sum(price) AS total_revenue
FROM ‘orders.csv’ AS orders
JOIN ‘items.cvs’ AS items on
   orders.order_id = items.order_id
GROUP BY 1
ORDER BY 1
```

order_date	total_revenue
2022-01-01	6
2022-01-02	5

How does revenue relate to shipping?

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

```
SELECT
   orders.order_date,
   SUM(items.price) AS total_revenue,
   SUM(orders.shipping_cost) AS total_shipping
FROM ‘orders.csv’ AS orders
JOIN ‘items.cvs’ AS items ON orders.order_id = items.order_id
GROUP BY 1
ORDER BY 1
```

order_date	total_revenue	total_shipping
2022-01-01	6	10
2022-01-02	5	5

```
SELECT
   orders.order_date,
   SUM(items.price) AS total_revenue,
   SUM(orders.shipping_cost) AS total_shipping
FROM ‘orders.csv’ AS orders
JOIN ‘items.cvs’ AS items ON orders.order_id = items.order_id
GROUP BY 1
ORDER BY 1
```

order_date	total_revenue	total_shipping
2022-01-01	6	10
2022-01-02	5	5

WRONG

```
SELECT *
FROM ‘orders.csv’ orders
LEFT JOIN ‘items.csv’ AS items ON orders.order_id = items.order_id
```

order_id	order_date	shipping_cost	user_id	item_id	order_id	item	price
1	2022-01-01	2	1	2	1	Twizzler	1
2	2022-01-01	3	2	4	2	M and M	1
3	2022-01-02	1	1	5	3	Twizzler	1
4	2022-01-02	2	3	7	4	Skittles	1
1	2022-01-02	2	1	1	1	Chocolate	2
2	2022-01-02	3	2	3	2	Chocolate	2
4	2022-01-02	2	3	6	4	Fudge	3

```
SELECT *
FROM ‘orders.csv’ orders
LEFT JOIN ‘items.csv’ AS items ON orders.order_id = items.order_id
```

order_id	order_date	shipping_cost	user_id	item_id	order_id	item	price
1	2022-01-01	2	1	2	1	Twizzler	1
2	2022-01-01	3	2	4	2	M and M	1
3	2022-01-02	1	1	5	3	Twizzler	1
4	2022-01-02	2	3	7	4	Skittles	1
1	2022-01-02	2	1	1	1	Chocolate	2
2	2022-01-02	3	2	3	2	Chocolate	2
4	2022-01-02	2	3	6	4	Fudge	3

Order rows are duplicated by the JOIN so computation is overstated.

```
SELECT *
FROM ‘orders.csv’ orders
LEFT JOIN ‘items.csv’ AS items ON orders.order_id = items.order_id
```

order_id	order_date	shipping_cost	user_id	item_id	order_id	item	price
1	2022-01-01	2	1	2	1	Twizzler	1
2	2022-01-01	3	2	4	2	M and M	1
3	2022-01-02	1	1	5	3	Twizzler	1
4	2022-01-02	2	3	7	4	Skittles	1
1	2022-01-02	2	1	1	1	Chocolate	2
2	2022-01-02	3	2	3	2	Chocolate	2
4	2022-01-02	2	3	6	4	Fudge	3

Order rows are duplicated by the JOIN so computation is overstated.

```
SELECT *
FROM ‘orders.csv’ orders
LEFT JOIN ‘items.csv’ AS items ON orders.order_id = items.order_id
```

order_id	order_date	shipping_cost	user_id	item_id	order_id	item	price
1	2022-01-01	2	1	2	1	Twizzler	1
2	2022-01-01	3	2	4	2	M and M	1
3	2022-01-02	1	1	5	3	Twizzler	1
4	2022-01-02	2	3	7	4	Skittles	1
1	2022-01-02	2	1	1	1	Chocolate	2
2	2022-01-02	3	2	3	2	Chocolate	2
4	2022-01-02	2	3	6	4	Fudge	3

Order rows are duplicated by the JOIN so computation is overstated.

Combine Result Rectangles

(Traditional data warehousing)

```
WITH orders_date AS (
   SELECT
      order_date,
      sum(shipping_cost) AS total_shipping
   FROM 'orders.csv'
   GROUP BY 1
),
```

order_date	total_shipping
2022-01-01	5
2022-01-02	3

```
WITH items_date AS (
   SELECT
      order_date,
      sum(price) AS total_revenue
   FROM ‘orders.csv’ AS orders
   JOIN ‘items.csv’ AS items
     ON orders.order_id = items.order_id
   GROUP BY 1
)
```

order_date	total_revenue
2022-01-01	6
2022-01-02	5

```
SELECT
   orders_date.order_date,
   total_revenue,
   total_shipping
FROM orders_date
JOIN items_date
  ON orders_date.order_date =
   items_date.order_date
```

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

```
WITH orders_date AS (
  SELECT
    order_date,
    sum(shipping_cost) AS total_shipping
  FROM 'orders.csv'
  GROUP BY 1
),
```

order_date	total_shipping
2022-01-01	5
2022-01-02	3

```
WITH items_date AS (
  SELECT
    order_date,
    sum(price) AS total_revenue
  FROM 'orders.csv' AS orders
  JOIN 'items.csv' AS items
    ON orders.order_id = items.order_id
  GROUP BY 1
)
```

order_date	total_revenue
2022-01-01	6
2022-01-02	5

```
SELECT
  orders_date.order_date,
  total_revenue,
  total_shipping
FROM orders_date
JOIN items_date
  ON orders_date.order_date =
    items_date.order_date
```

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

user_id	total_revenue	total_shipping
1	4	3
2	3	3
3	4	2

```
WITH orders_date AS (
  SELECT
    order_date,
    sum(shipping_cost) AS total_shipping
  FROM ‘orders.csv’
  GROUP BY 1
),

WITH items_date AS (
  SELECT
    order_date,
    sum(price) AS total_revenue
  FROM ‘orders.csv’ AS orders
  JOIN ‘items.csv’ AS items
    ON orders.order_id = items.order_id
  GROUP BY 1
)

SELECT
  orders_date.order_date,
  total_revenue,
  total_shipping
FROM orders_date
JOIN items_date
  ON orders_date.order_date =
    items_date.order_date
```

```
WITH orders_date AS (
  SELECT
    order_date,
    sum(shipping_cost) AS total_shipping
  FROM ‘orders.csv’
  GROUP BY 1
),

WITH items_date AS (
  SELECT
    order_date,
    sum(price) AS total_revenue
  FROM ‘orders.csv’ AS orders
  JOIN ‘items.csv’ AS items
    ON orders.order_id = items.order_id
  GROUP BY 1
)

SELECT
  orders_date.order_date,
  total_revenue,
  total_shipping
FROM orders_date
JOIN items_date
  ON orders_date.order_date =
    items_date.order_date
```

```
WITH orders_user_id AS (
  SELECT
    user_id,
    sum(shipping_cost) AS total_shipping
  FROM 'orders.csv'
  GROUP BY 1
),

WITH items_user_id AS (
  SELECT
    user_id,
    sum(price) AS total_revenue
  FROM 'orders.csv' AS orders
  JOIN 'items.csv' AS items
    ON orders.order_id = items.order_id
  GROUP BY 1
)

SELECT
  order_user_id.user_id,
  total_revenue,
  total_shipping
FROM orders_user_id
JOIN items_user_id
  ON orders_user_id.user_id =
   items_user_id.user_id
```

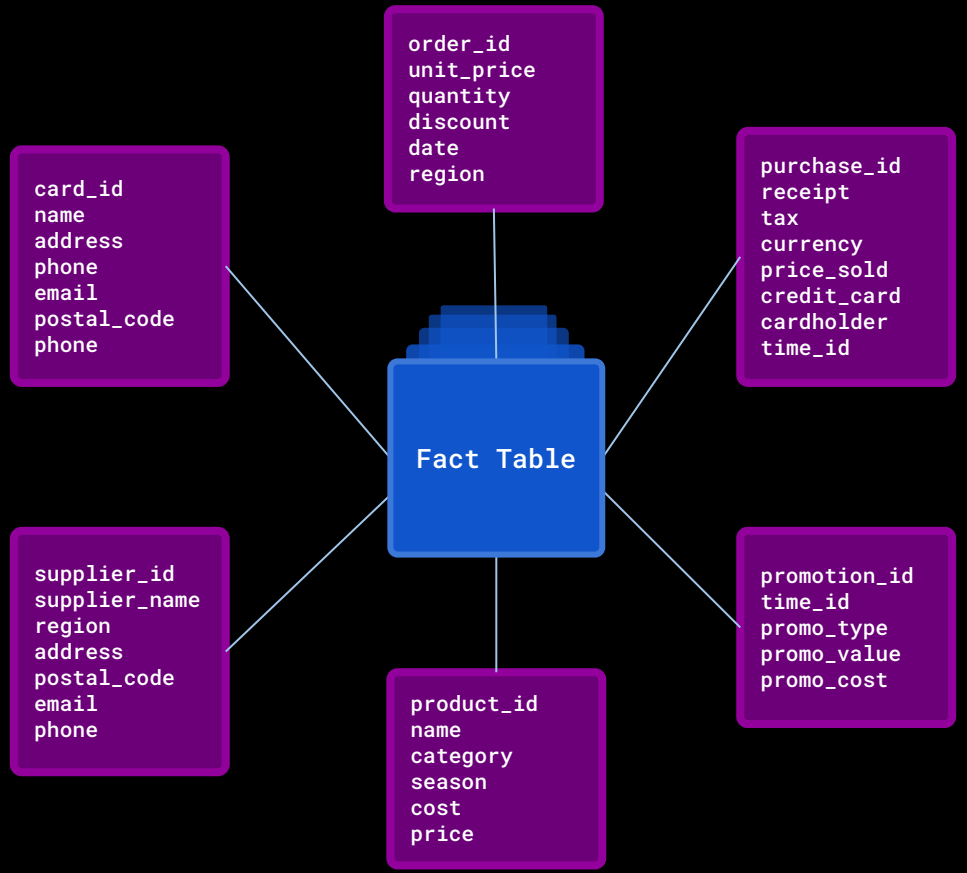
```
WITH orders_user_id AS (
  SELECT
    user_id,
    sum(shipping_cost) AS total_shipping
  FROM 'orders.csv'
  GROUP BY 1
),

WITH items_user_id AS (
  SELECT
    user_id,
    sum(price) AS total_revenue
  FROM 'orders.csv' AS orders
  JOIN 'items.csv' AS items
    ON orders.order_id = items.order_id
  GROUP BY 1
)

SELECT
  order_user_id.use_id,
  total_revenue,
  total_shipping
FROM orders_user_id
JOIN items_user_id
  ON orders_user_id.user_id =
   items_user_id.user_id
```

user_id	total_shipping
1	3
2	3
3	2

user_id	total_revenue
1	4
2	3
3	4

user_id	total_revenue	total_shipping
1	4	3
2	3	3
3	4	2

Traditional Data Warehousing
Star Schema



Enter Malloy

Malloy makes the promise that join relations won’t affect aggregate calculations.

Malloy makes the promise that join relations won't affect aggregate calculations.

Join data in a similar way to SQL.

Malloy makes the promise that join relations won't affect aggregate calculations.

Join data in a similar way to SQL.

Write aggregate calculations with pathing to node in the network.

Malloy makes the promise that join relations won't affect aggregate calculations.

Join data in a similar way to SQL.

Write aggregate calculations with pathing to node in the network.

Aggregate calculations are always correct

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

SOURCE

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

LOCAL TO ITEMS

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

LOCAL TO ORDERS

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

Malloy

```
run: table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
}
-> {
  group_by: user_id
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

user_id	total_revenue	total_shipping
1	4	3
2	3	3
3	4	2

Dimensional Freedom

Produce results from anywhere in the join network

```
SELECT
  base.”order_date” AS “order_date”,
  COALESCE(SUM(items_0.”price”),0) AS “total_revenue”,
  COALESCE((
    SELECT sum(a.val) AS value
    FROM (
      SELECT UNNEST(list(distinct {key:base.”__distinct_key”,
val: base.”shipping_cost”})) a
        )
      ),0) AS “total_shipping”
FROM (SELECT GEN_RANDOM_UUID() AS __distinct_key, * FROM orders.csv
AS x) AS base
LEFT JOIN items.csv AS items_0
  ON base.”order_id”=items_0.”order_id”
GROUP BY 1
ORDER BY 1 ASC NULLS LAST
```

Malloy's reusability is a `source`

```
source: orders_items is table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
  measure:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

Sources are named

```
source: orders_items is table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
  measure:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

Sources describe the join relationships

```
source: orders_items is table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
  measure:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

Sources describe the calculations (aggregate and scalar)

```
source: orders_items is table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
  measure:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

Using a source makes queries very simple

```
run: orders_items -> {
  group_by: order_date
  aggregate: total_revenue, total_shipping
  order_by: 1
}
```

Using a source makes queries very simple

```
run: orders_items -> {
  group_by: order_date
  aggregate: total_revenue, total_shipping
  order_by: 1
}

run: orders_items -> {
  group_by: user_id
  aggregate: total_revenue, total_shipping
  order_by: 1
}
```

Using a source makes queries very simple

```
run: orders_items -> {
  group_by: order_date
  aggregate: total_revenue, total_shipping
  order_by: 1
}

run: orders_items -> {
  group_by: user_id
  aggregate: total_revenue, total_shipping
  order_by: 1
}

run: orders_items -> {
  aggregate: total_revenue
}
```

```
[
  {
    “order_id”: 1,
    “order_date”: “2022-01-01”,
    “shipping_cost”: 2,
    “user_id”: 1,
    “items”: [
      {
        “item_id”: 1,
        “item”: “Chocolate”,
        “price”: 2
      },
      {
        “item_id”: 2,
        “item”: “Twizzler”,
        “price”: 1
      }
    ]
  },
  {
    “order_id”: 2,
    “order_date”: “2022-01-01”.
```


column_name	column_type	null	key
order_id	INTEGER	YES	
order_date	DATE	YES	
shipping_cost	INTEGER	YES	
user_id	INTEGER	YES	
items	STRUCT(item_id INTEGER, item VARCHAR, price INTEGER)[]	YES	

```
run: table('duckdb:orders_items.parquet')
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  order_by: 1
}
```

order_date	total_revenue	total_shipping
2022-01-01	6	5
2022-01-02	5	3

```
query:
table('duckdb:orders_items.parquet')
-> {
  group_by: order_date
  aggregate:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
  nest: by_items is {
    group_by: items.item
    aggregate: total_revenue is
      items.price.sum()
  }
  order_by: 1
}
```

order_date	total_revenue	total_shipping	by_items	
2022-01-01	6	5	item	total_revenue
			Chocolate	4
			Twizzler	1
			M and M	1
2022-01-02	5	3	item	total_revenue
			Fudge	3
			Skittles	1
			Twizzler	1

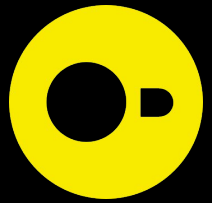
```
WITH stage0 AS (
  SELECT
   group_set,
   CASE WHEN group_set IN (0,1) THEN
     base. "order_date"
     END as "order_date_0"
   CASE WHEN group_set=0 THEN
     COALESCE(SUM(base.items [items_0.__row_id]."price"),0)
     END as "total_revenue_0"
   CASE WHEN group_set=0 THEN
     COALESCE((
        SELECT sum(a.val) as value
        FROM (
           SELECT UNNEST(list(distinct {key:base."__distinct_key" val: base."shipping_cost"})) a
        )
     ),0)
     END as "total_shipping__0",
   CASE WHEN group_set=1 THEN
     base.items[items_0.__row_id]."item"
     END as "item__1",
   CASE WHEN group_set=1 THEN
     COALESCE(SUM(base.items[items_0.__row_id]."price"),0)
     END as "total_revenue__1"
  FROM (SELECT GEN_RANDOM_UUID() as __distinct_key, *
        FROM orders_items.parquet as x) as base
  LEFT JOIN (select UNNEST(generate_series(1,100000,
        -- (SELECT genres_length FROM movies limit 1),
        1)) as __row_id) as items_O ON items_0.__row_id <=
           array_length(base."items")
  CROSS JOIN (SELECT UNNEST (GENERATE_SERIES(0,1,1)) as group_set ) as group_set
  GROUP BY 1,2,5
)
SELECT
  "order_date__0" as "order_date"
  MAX(CASE WHEN group_set=0 THEN total_revenue 0 END) as
     "total revenue",
  MAX(CASE WHEN group_set=0 THEN total_shipping_0 END) as
     "total_shipping",
  COALESCE(LIST({
     "item": "item__1",
     "total_revenue": "total revenue_1"} ORDER BY
     "total_revenue__1" desc NULLS LAST) FILTER (WHERE group_set=1), []) as "by items"
FROM __stage0
GROUP BY 1
ORDER BY 1 ASC NULLS LAST
```

Demo

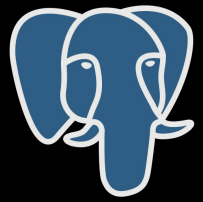
https://github.dev/malloydata/patterns

Malloy supports Databases

BigQuery



DuckDB



Postgres

The Malloy Language

Window Functions

Sampled Dimensional Indexes

One Malloy is One SQL query

Nested Queries

Semantic data modeling

Automatic modeling of nested sources

Aggregate locality

Level of detail Calculations
(ungrouped aggregates)

Filtered Aggregates

Annotations

Transformation
Malloy in SQL/SQL in Malloy

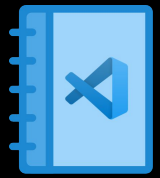
Partial relational expressions

Standard Cross SQL function library

Pipelined queries
(even when nested)

Specialized Nested Renderer

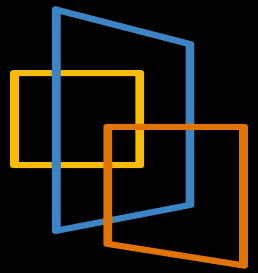
Malloy runs in / as a


http://www.malloydata.dev

ltabb@google.com

Thanks!

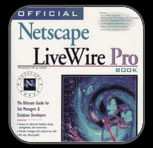


Data is Rectangular

(and other limiting misconceptions)

Force
1987

dBASE
1992



LiveWire
1994

liveops

LTool (perl)
2003

LTool (python)
2007

Readyforce

El Tool (php)
2009

looker

Looker
2012

Malloy
2020

In SQL Joins, produce a new rectangle

In SQL joins produce a new rectangle.

FIRST: Joins tables expand rows to first produce a new rectangle

THEN: perform Rectangular operations up on the new rectangle.

Sources describe the calculations (aggregate and scalar)

```
source: orders_items is table('duckdb:orders.csv') + {
  join_many: items is table('duckdb:items.csv')
    on order_id = items.order_id
  declare:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

Malloy runs in / as a

VS Code Dev Environment

VS Code Notebooks

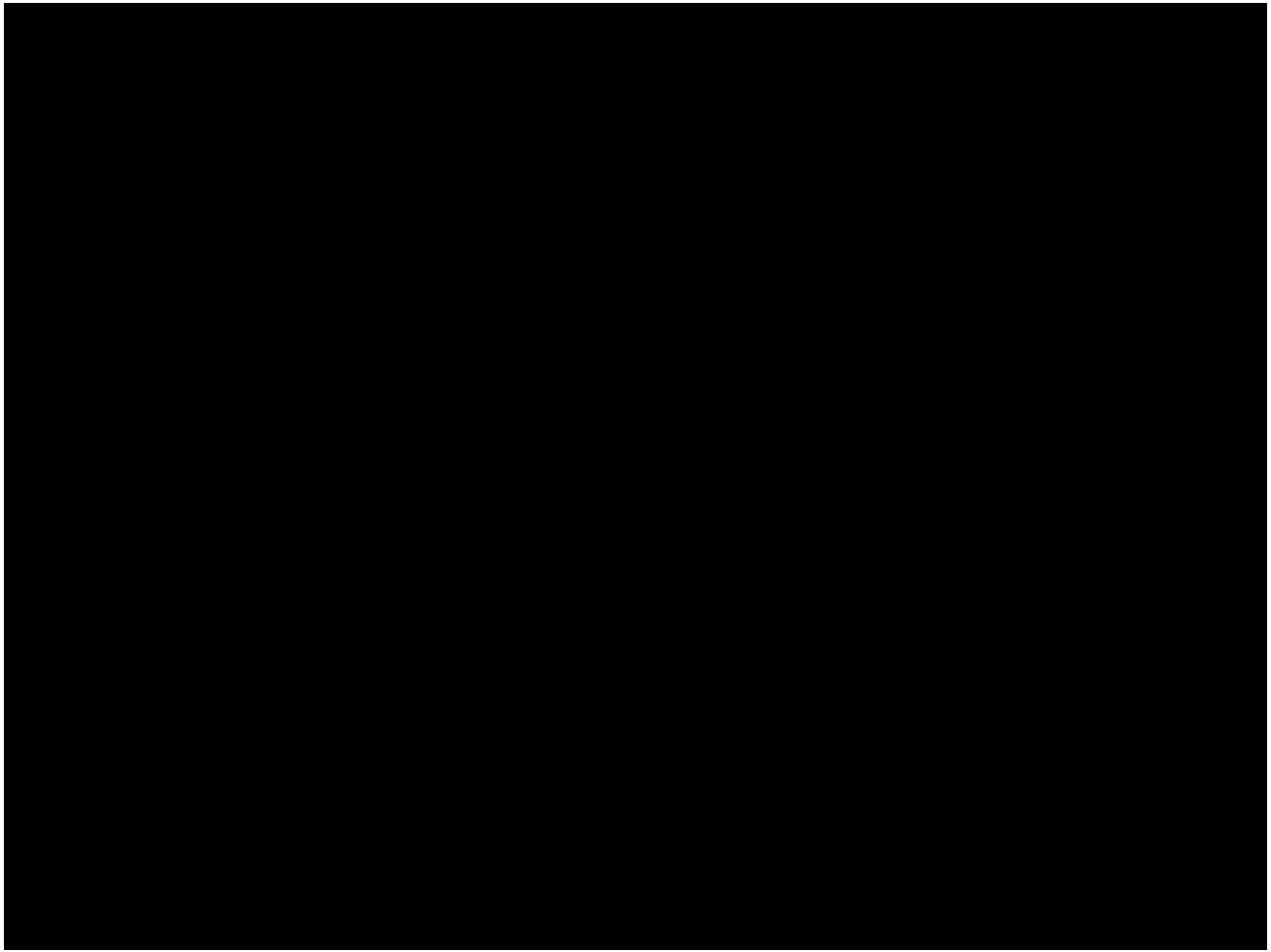
Command Line

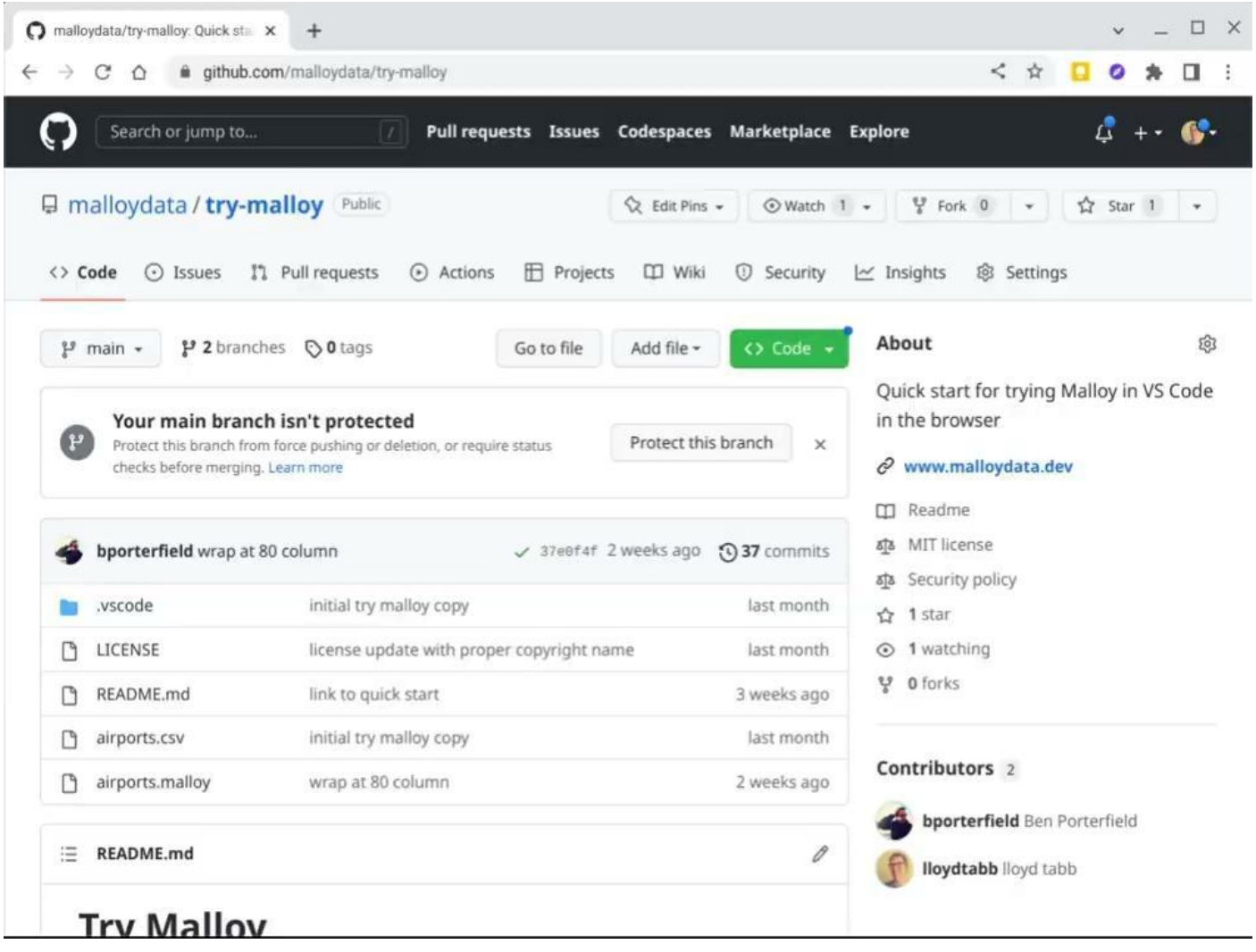
NPM Library

Python Library

Jupyter Notebooks

Malloy Composer




malloydata/try-malloy: Quick sta
github.com/malloydata/try-malloy
Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
malloydata / try-malloy
Public
Edit Pins
Watch 1
Fork 0
Star 1
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
2 branches
0 tags
Go to file
Add file
Code
Your main branch isn't protected
Protect this branch from force pushing or deletion, or require status checks before merging. Learn more
Protect this branch
bporterfield wrap at 80 column
37e0f4f 2 weeks ago
37 commits
.vscode initial try malloy copy last month
LICENSE license update with proper copyright name last month
README.md link to quick start 3 weeks ago
airports.csv initial try malloy copy last month
airports.malloy wrap at 80 column 2 weeks ago
README.md
Try Malloy
About
Quick start for trying Malloy in VS Code in the browser
www.malloydata.dev
Readme
MIT license
Security policy
1 star
1 watching
0 forks
Contributors 2
bporterfield Ben Porterfield
lloydtabb lloyd tabb