hi, i’m lloyd
Data is Rectangular
(and other limiting misconceptions)
Humans think in rectangular calculations
Operations within the Rectangle

- filtering
- projecting
- group by / aggregate
- windowing
In SQL Joins, produce a new rectangle
# orders

<table>
<thead>
<tr>
<th>order_id</th>
<th>order_date</th>
<th>shipping_cost</th>
<th>user_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-01-01</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-01</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2022-01-02</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>order_id</td>
<td>order_date</td>
<td>shipping_cost</td>
<td>user_id</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>2022-01-01</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-01</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2022-01-02</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item_id</th>
<th>order_id</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>M and M</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Fudge</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>Skittles</td>
<td>1</td>
</tr>
</tbody>
</table>
Let’s measure two things, from sales...

total_shipping

total_revenue
SELECT SUM(shipping_cost) AS total_shipping FROM 'orders.csv
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

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>3</td>
</tr>
</tbody>
</table>
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 BY 1

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
</tr>
</tbody>
</table>
How does revenue relate to shipping?

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
SELECT
    orders.order_date,
    SUM(items.price) AS total_revenue,
    SUM(orders.shipping_cost) AS total_shipping
FROM 'orders.csv' AS orders
JOIN 'items.csv' AS items ON orders.order_id = items.order_id
GROUP BY 1
ORDER BY 1

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
SELECT
    orders.order_date,
    SUM(items.price) AS total_revenue,
    SUM(orders.shipping_cost) AS total_shipping
FROM `orders.csv` AS orders
JOIN `items.csv` AS items ON orders.order_id = items.order_id
GROUP BY 1
ORDER BY 1

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
SELECT *
FROM 'orders.csv' orders
LEFT JOIN 'items.csv' AS items ON orders.order_id = items.order_id

<table>
<thead>
<tr>
<th>order_id</th>
<th>order_date</th>
<th>shipping_cost</th>
<th>user_id</th>
<th>item_id</th>
<th>order_id</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-01-01</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-01</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>M and M</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2022-01-02</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>Skittles</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2022-01-02</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-02</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>Fudge</td>
<td>3</td>
</tr>
</tbody>
</table>
SELECT *
FROM 'orders.csv' orders
LEFT JOIN 'items.csv' AS items ON orders.order_id = items.order_id

<table>
<thead>
<tr>
<th>order_id</th>
<th>order_date</th>
<th>shipping_cost</th>
<th>user_id</th>
<th>item_id</th>
<th>order_id</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-01-01</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-01</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>M and M</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2022-01-02</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>Skittles</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2022-01-02</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-02</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>Fudge</td>
<td>3</td>
</tr>
</tbody>
</table>

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 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

<table>
<thead>
<tr>
<th>order_id</th>
<th>order_date</th>
<th>shipping_cost</th>
<th>user_id</th>
<th>item_id</th>
<th>order_id</th>
<th>item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022-01-01</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-01</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>M and M</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2022-01-02</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>Twizzler</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>Skittles</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2022-01-02</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2022-01-02</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Chocolate</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2022-01-02</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>Fudge</td>
<td>3</td>
</tr>
</tbody>
</table>

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  
),

<table>
<thead>
<tr>
<th>order_date</th>
<th>totalShipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>3</td>
</tr>
</tbody>
</table>
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  
)

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
</tr>
</tbody>
</table>
SELECT
  orders_date.order_date,
  total_revenue,
  total_shipping
FROM orders_date
JOIN items_date
  ON orders_date.order_date =
      items_date.order_date

<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>order_date</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>order_date</td>
<td>total_revenue</td>
<td>total_shipping</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>2022-01-01</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2022-01-02</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>user_id</th>
<th>total_revenue</th>
<th>total_shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
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 order_id  
)  

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  
    order_id,  
    sum(shipping_cost) AS total_shipping  
FROM 'orders.csv'  
GROUP BY 1  
),  

WITH items_user_id AS (  
SELECT  
    order_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
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
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
}
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
}
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
}
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
}
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
}
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
}
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)

```sql
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

```plaintext
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
}
<table>
<thead>
<tr>
<th>column_name</th>
<th>column_type</th>
<th>null</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_id</td>
<td>INTEGER</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>order_date</td>
<td>DATE</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>shipping_cost</td>
<td>INTEGER</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>user_id</td>
<td>INTEGER</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>items</td>
<td>STRUCT(item_id INTEGER, item VARCHAR, price INTEGER)[]</td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>

```json
[
  {
    "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",
  }
]
```
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
}
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
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)  
),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

- Semantic data modeling
- Nested Queries
- Pipelined queries (even when nested)
- Level of detail Calculations (ungrouped aggregates)
- Annotations
- Window Functions
- Sampled Dimensional Indexes
- Aggregate locality
- Transformation
- Malloy in SQL/SQL in Malloy
- Partial relational expressions
- Standard Cross SQL function library
- Specialized Nested Renderer
- Filtered Aggregates
- One Malloy is One SQL query
- Automatic modeling of nested sources
- Pipelined queries (even when nested)
Malloy runs in / as a

[Images of various software icons, including VS Code, Jupyter, and npm]
Thanks!
Data is Rectangular
(and other limiting misconceptions)
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)

```sql
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
Try Malloy

Your main branch isn't protected
Protect this branch from force pushing or deletion, or require status checks before merging. Learn more

bporterfield wrap at 80 column
37e84f 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

Try Malloy

Quick start for trying Malloy in VS Code in the browser

www.malloydatalocal.dev

Readme
MIT license
Security policy
1 star
1 watching
0 forks

Contributors
bporterfield Ben Porterfield
lloydtabb lloyd tabb