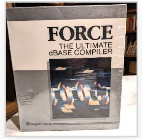
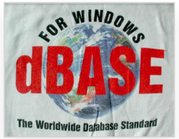
hi, i’m lloyd

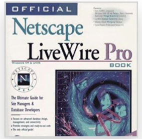
Encore!



1987 - Force



1992 - dBASE



1994 - LiveWire

2003 - LTool (perl)

2007 - LTool (python)

2009 - El Tool (php)

2012 - Looker

2020 - Malloy

Data is Rectangular and other Limiting Misconceptions

Operations within the Rectangle

filtering

group by / aggregate

projecting

windowing

Humans think in rectangular calculations

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.

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	23	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	23	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
```

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

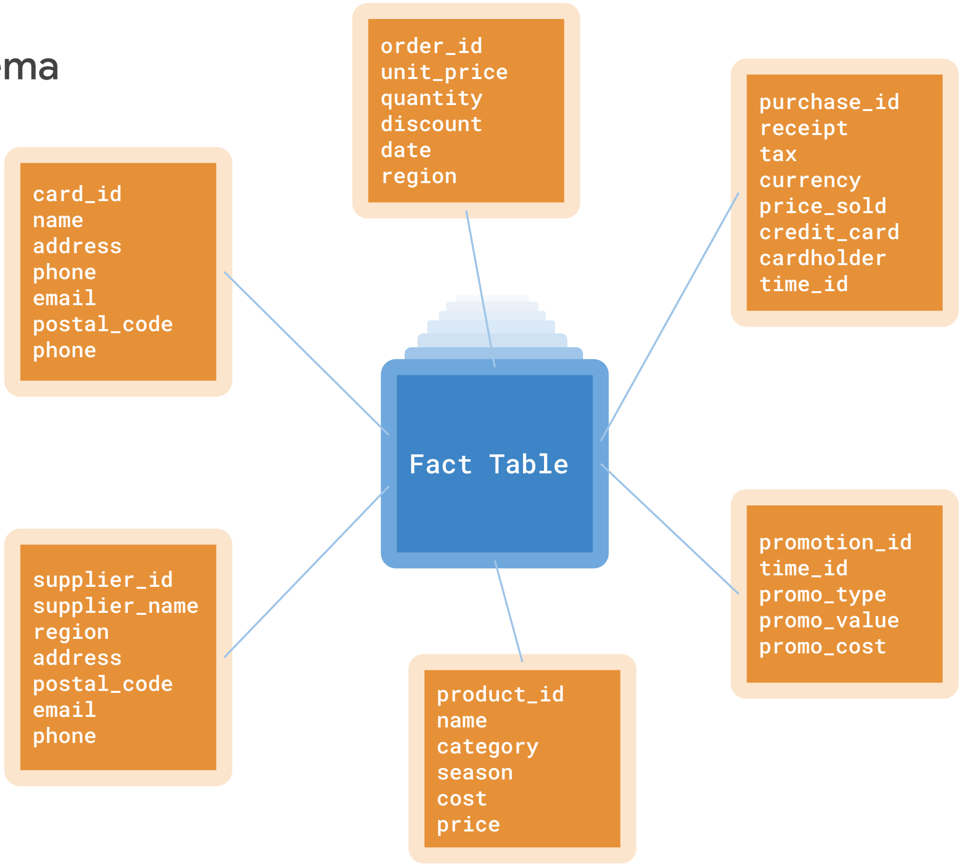
```
WITH orders_user_id AS (
  SELECT
    user_id,
    sum(shipping_cost) AS total_shipping
  FROM ‘orders.csv’
  GROUP BY 1
),

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 orders_user_id
  ON orders_user_id.user_id =
   items_user_id.user_id
```

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

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

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

Traditional data warehouse star schema



Traditional Data Warehousing (Star Schema)

Designed at a time when

- databases were slow
- data was relatively big

Traditional Data Warehousing (Star Schema)

Designed at a time when

- databases were slow
- data was relatively big

Not real time - reporting only

Traditional Data Warehousing (Star Schema)

Designed at a time when

- databases were slow
- data was relatively big

Not real time - reporting only

ETL from storage format to reporting format

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.

Data is first described in a network. The network of joined rectangles is a reusable object called a source.

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

Data is first described in a network. The network of joined rectangles is a reusable object called a source.

In a query operation, aggregate calculations are applied. The aggregate calculations can reference any ‘locality’ in the join network and will compute results correctly.

Malloy

```
query: 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

```
query: 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

```
query: 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

```
query: 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

```
query: 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

```
query: 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

```
query: 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

```
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 orders_user_id
  ON orders_user_id.user_id =
   items_user_id.user_id
```

```
query: 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
  declare:
    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
  declare:
    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
  declare:
    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
  declare:
    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
  declare:
    total_revenue is items.price.sum()
    total_shipping is shipping_cost.sum()
}
```

```
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()
}
```

Using a source makes queries very simple

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

```
  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()
}
```

Using a source makes queries very simple

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

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

```
    total_shipping is shipping_cost.sum()
}
```

Using a source makes queries very simple

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

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

query: 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	

```
query: 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_0 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

http://www.malloydata.dev