What is Data Retrieval?

SDSC SAN DIEGO SUPERCOMPUTER CENTER

What is Data Retrieval?

- **Data retrieval**
  - The way in which the desired data is specified and retrieved from a data store
- **Our focus**
  1. How to specify a data request (query specification method)
     - For static and streaming data
  2. The internal mechanism of data retrieval
     - For large and streaming data

small < big data ★ (focus on this)

What is a Query Language?

- A language to specify the data items you need
- A query language is declarative
  - Specify what you need rather than how to obtain it
  - SQL (Structured Query Language)
- Database programming language
  - Procedural programming language
  - Embeds query operations

(you specify and the system does the rest)

most used query language for relational data

high level

like oracle's PL/SQL, Postgres's PgSQL

SQL

ubiquitous query language when data is structured

has been extended to accommodate other types of data

oracle spark

- The standard for structured data
  - Oracle's SQL to Spark SQL
- Example Database Schema

6 relations of tables

Bars(name, addr, license)

Beers(name, manf) — manufactures

Sells(bar, beer, price) — which bar sells which beer at what price

Drinkers(name, addr, phone)

Frequents(drinker, bar)

Likes(drinker, beer) (knows its customers)

primary keys underlined

↳ attributes that make a record unique

name	addr	license
Great American Bar	363 Main St., SD, CA 92390	41-437844098
Beer Paradise	6450 Mango Drive, SD, CA 92130	41-973428319
Have a Good Time	8236 Adams Avenue, SD, CA 92116	32-032263401

SELECT-FROM-WHERE

- Which beers are made by Heineken?

```
SELECT name
FROM Beers
WHERE manf = 'Heineken'
```

Output attribute(s)

Table(s) to use

* match exactly

↳ single string literal

The condition(s) to satisfy

Strings like 'Heineken' are case-sensitive and are put in quotes

Select $_{manf='Heineken'}$ (Beers)

↓

Project(name)

name
Heineken Lager Beer
Amstel Lager
Amstel Light
...

→ result table with name row

More Example Queries

- **Find expensive beer**
  - SELECT DISTINCT beer, price
  - FROM Sells
  - WHERE price > 15

ensures that result relation will have no duplicate

- **Which businesses have a Temporary License (starts with 32) in San Diego?**
  - SELECT name
  - FROM Bars
  - WHERE addr LIKE '%SD%' **AND** license LIKE '32%' LIMIT 5

limit on number of results to return
* limit clause may vary between DBMS vendors

name	addr	license
Great American Bar	363 Main St., SD, CA 92390	41-437844098
Beer Paradise	6450 Mango Drive, SD, CA 92130	41-973428319
Have a Good Time	8236 Adams Avenue, SD, CA 92116	32-032263401

* satisfies condition 1 but not 2

Select-Project Queries in the Large

- **Large Tables can be partitioned** (millions of entries) — table split over many machines
  - Many partitioning schemes → depending on operation, it can be evaluated in parallel
    - Range partitioning on primary key ~ way of partitioning

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1 — A, B

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2 — C, D

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5 — H

…

Select-Project Queries in the Large

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

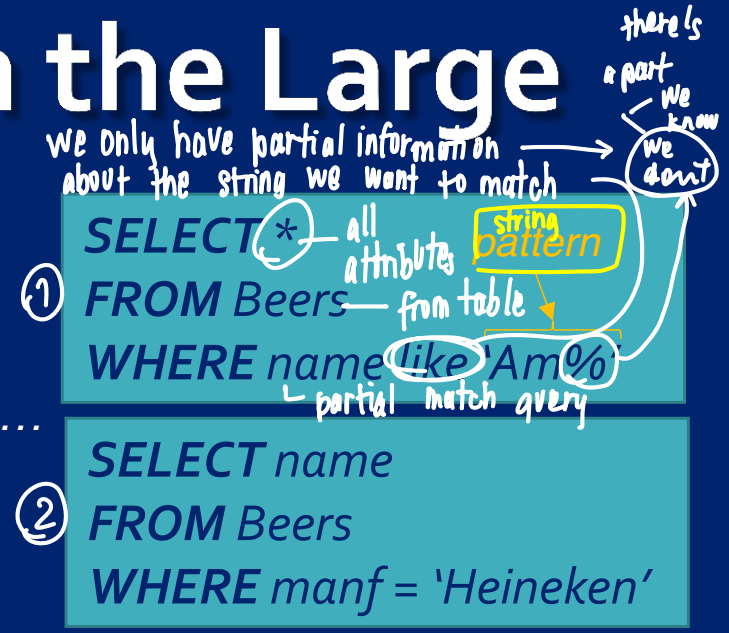
Machine 2

…

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

…



- **Two queries**
  - ① Find records for beers whose name starts with 'Am'
  - ② Which beers are made by Heineken?

Evaluating SP Queries for Large Data

* do we need to touch all partitions to answer the query? NO! name is arranged and partitioned based on it

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

…

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

…

```
SELECT *
FROM Beers
WHERE name like 'Am%'
```

evaluation process should only access machine 1 because no other machine will have records for names starting with A

* so long as system knows partitioning strategy, it can make its job more efficient if it matters

- **A query processing trick**
  - Use the partitioning information
    - Just use partition 1!!

Evaluating SP Queries for Large Data

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

```
SELECT name
FROM Beers
WHERE manf = 'Heineken'
```

Broadcast query

executed in parallel

In each machine in parallel:
Select $_{manf='Heineken'}$ (Beers)
Project(name)

Gather Partial Results
Union
Return

(Essence of dealing with large data)

1. query broadcasted from primary machine to all machines
2. run in local machine
3. results back to primary
4. unioned together
5. return results to client

Cant get away by using partitioning information because partitioning activity is different from attribute on which the query is applied

this will need to go to all partitions

Local and Global Indexing

- **What if a machine does not have any data for the query attributes?**
- **Index structures** — index = reverse table
  - Given value, return records
  - Several solutions
    - Use local index on each machine
      main query will go to all machines but lookup will be instant
    - Use a machine index for each value
      keeps an account of the machine that contains the record with that value
    - Use a combined index in a global index server
      ↳ use both schemes → uses more space, queries are faster

* using an index speeds up query processing

manf	RecordIDs
…	…
MillerCoors	34, 35, 87, 129, …
Duvel Moortgat	5, 298, 943, 994,…
Heineken	631, 683, 882,…
…	…

manf	machineIDs
…	…
MillerCoors	10
Duvel Moortgat	3, 4
Heineken	1, 3, 5
	↳ heineken is in 3 machines

Pause

QUERIES FOR TWO TABLES

Querying Two Relations

- **Often we need to combine two relations for queries**
  - Find the beers liked by drinkers who frequent The Great American Bar
- **In SQL**

```
SELECT DISTINCT beer
FROM Likes L, Frequents F
WHERE bar = 'The Great American Bar' AND
      F.drinker = L.drinker
```

common attribute

avoids duplicates

shortcut from this

(attrs are unique)

alias

alias

single table condition

conditions

joined condition

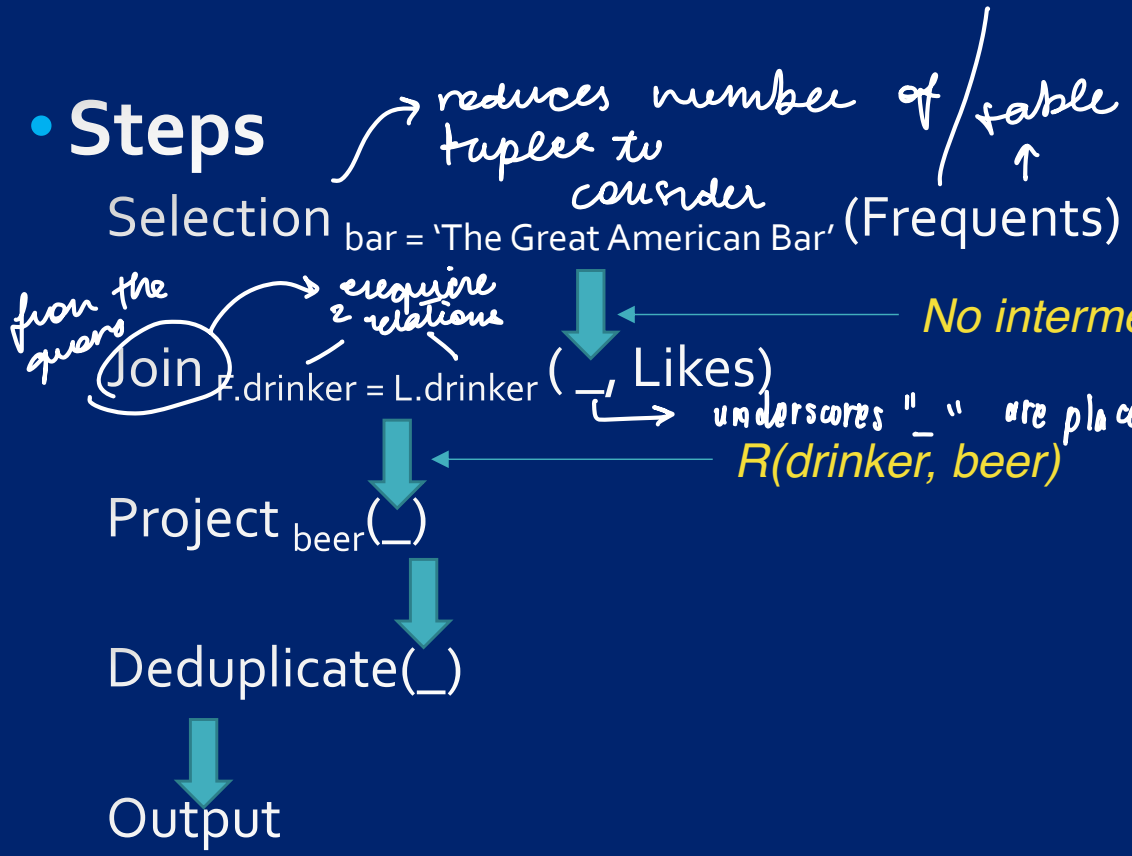
SPJ Queries

Frequents(drinker, bar)
Likes(drinker, beer)

```
SELECT DISTINCT beer
FROM Likes L, Frequents F
WHERE bar = 'The Great American Bar'
      AND F.drinker = L.drinker
```

↓ the selection is piped into the join operation

- **Steps**



Join in a Distributed Setting

Frequents(drinker, bar)
Likes(drinker, beer)

two different machines

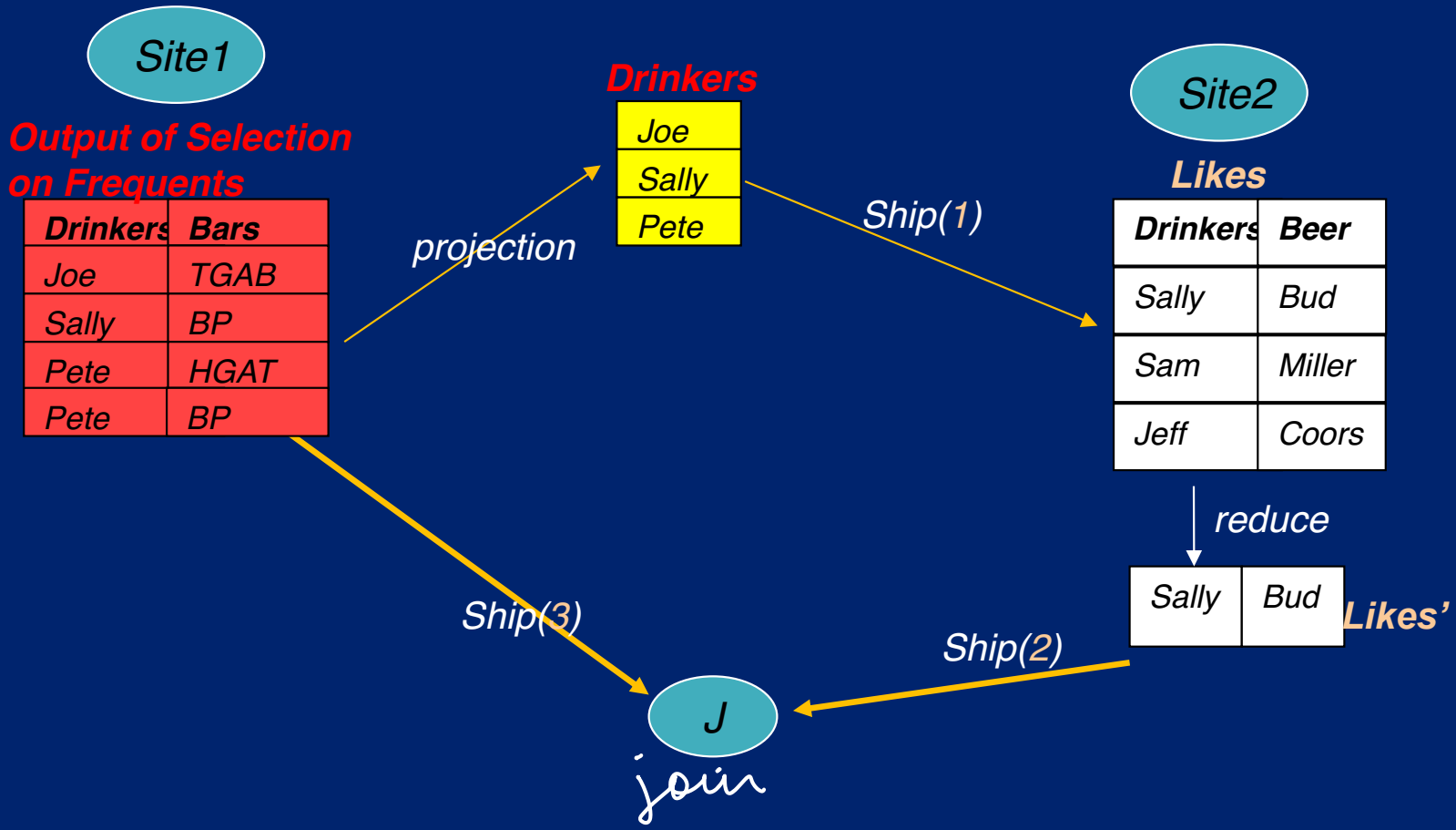
- **Semijoin** operation — move data from one machine to another
  - A semijoin from R to S on attribute is used to reduce the data transmission cost
  - Computing steps:
    1. **Project** R on attribute A and call it (R[A]) – the Drinkers column
    2. **Ship** this projection (a semijoin projection) from the site of R to the site of S
    3. **Reduce** S to S′ by eliminating tuples where attribute A are not matching any value in R[A]

Frequents (~~drinker~~, bar) → projected out

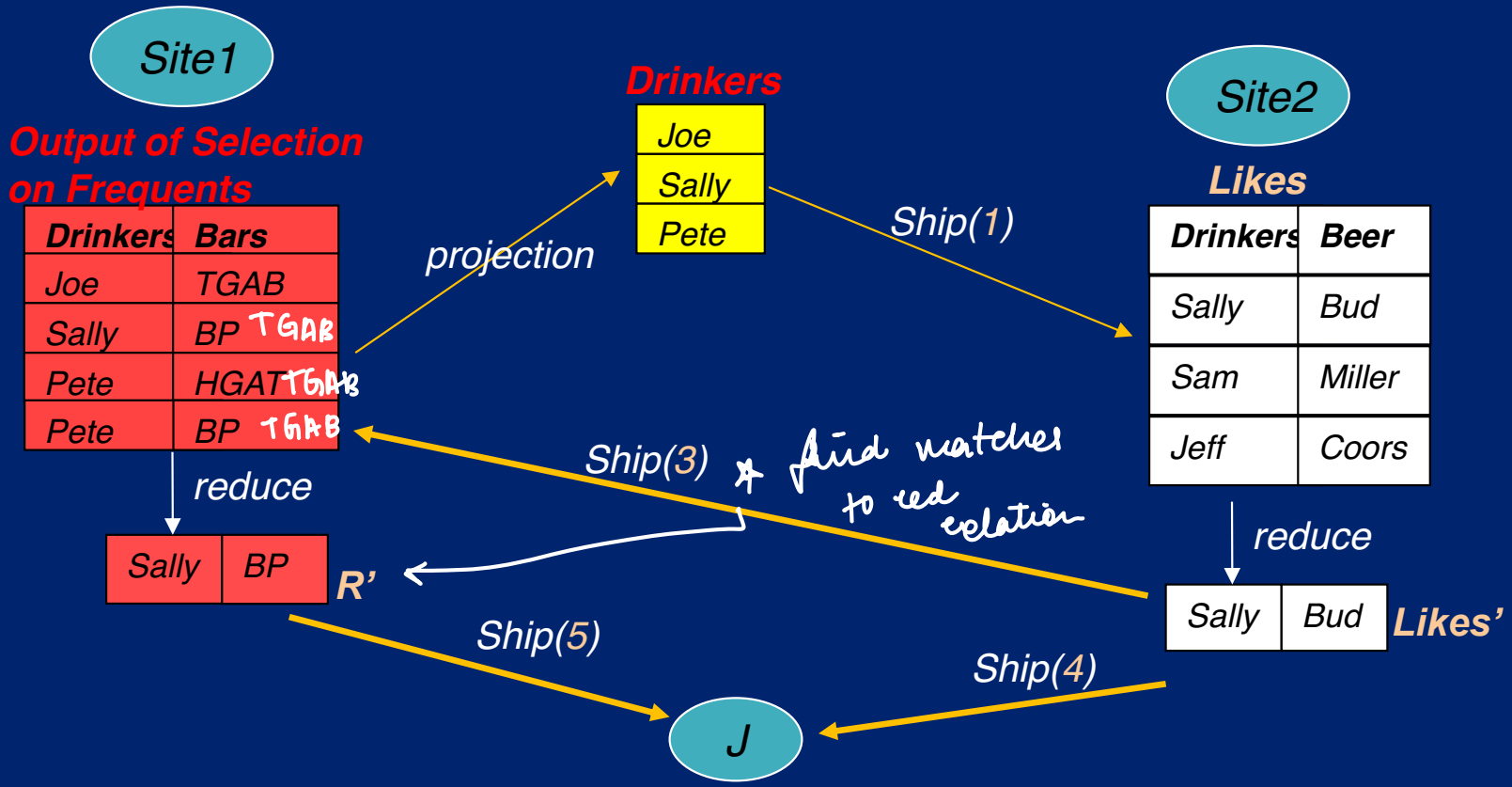
frequents (machine) → likes (machine)

$ (cost is reduced if ship less data

Semijoin s: Frequents—Drinkers→Likes



Semijoin s: Frequents—Drinkers→Likes



Pause

Subqueries

- A slightly complex query
- Find the bars that serve Miller for the same or less price than what TGAB charges for Bud
- We may break it into two queries:
  1. Find the price TGAB charges for Bud
  2. Find the bars that serve Miller at that price

the great american bar

result is fed as parameter to second query

Subqueries in SQL

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price <= (SELECT price
              FROM Sells
              WHERE bar = 'TGAB'
              AND beer = 'Bud');
```

The price at which TGAB sells Bud

subquery (inner query)

- independent
- uncorrelated

Subqueries with IN

- **Find the name and manufacturer of each beer that Fred does not like**
- **Query**

Beers(name, manf)
Likes(drinker, beer)

every name does not appear here

```
SELECT *
FROM Beers
WHERE name NOT IN
    ( SELECT beer
      FROM Likes
      WHERE drinker = 'Fred');
```

uncorrelated

Correlated Subqueries

- **Find the name and price of each beer that is more expensive than the average price of beers sold in the bar**

```
SELECT beer, price
FROM  Sells s1
WHERE price >
    (SELECT AVG(price)
     FROM Sells s2
     WHERE s1.bar = s2.bar)
```

Bar	Beer	Price
HGAT	Bud	5
BP	Michelob	4
TGAB	Heineken	6
HGAT	Guinness	10

① average of every bar
② compare price of each beer to average

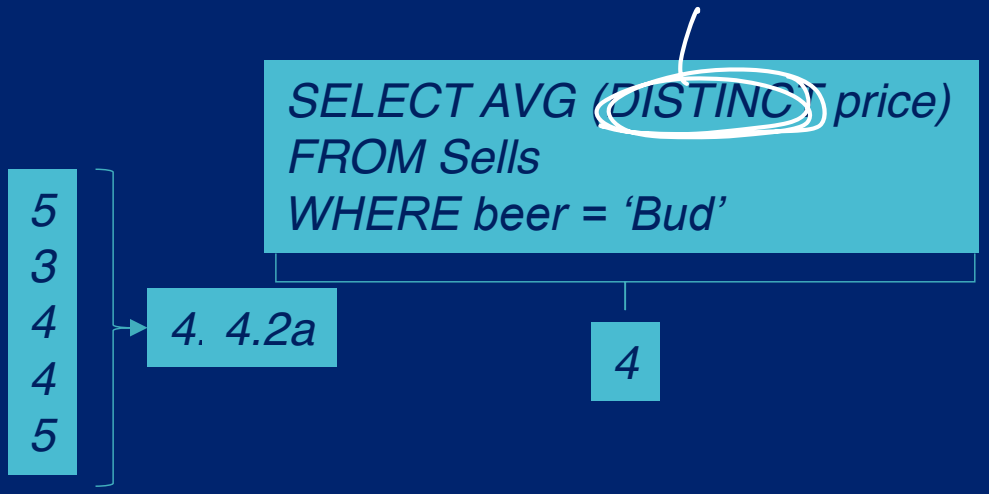
for every tuple processed by the outer query, one needs to compute the inner query for that bar

↓
correlated

stores average once computed and reuses it

Aggregate Queries

- **Example**
  - Find the average price of Bud:
  - SELECT AVG(price)
  - FROM Sells
  - WHERE beer = 'Bud';
- **Other aggregate functions**
  - SUM, MIN, MAX, COUNT, ...



GROUP BY Queries

- Find for each drinker the average price of Bud at the bars they frequent

group by — aggregate

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

used for selection

join attribute

grouping variable

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Joe	TGAB	6
Joe	HGAT	5

Drinker	Price
Pete	4.5
Joe	5.5

statistical aggregate by groups

: creates one result row for each drinker and place average price

Grouping Aggregates over Partitioned Data

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Joe	TGAB	6
John	HGAT	5

result of selection in two different machines

Drinker	Bar	Price
Pete	BO	6
John	BP	4
Sally	TGAB	6
Sally	HGAT	5

① data gets repartitioned by grouping attribute (drinker)

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Pete	BO	6
Joe	TGAB	6

Drinker	Bar	Price
John	HGAT	5
John	BP	4
Sally	TGAB	6
Sally	HGAT	5

② aggregate function computed locally

Drinker	Price
Pete	5
Joe	6

Drinker	Price
John	4.5
Sally	5.5

each machine groups its own data locally

* there are variants