

# Views

DSCI 551

Wensheng Wu

# Example (ETL)

ETL (Extract, Transform, Load)

- insert into W(time, tweet, stock)
- select T.time, tweet\_content, stock \* 200
- from ds1.T join ds2.S on T.time = S.timestamp
- Create materialized view W(time, tweet, stock)
- As select T.time, tweet\_content, stock \* 200
- from ds1.T join ds2.S on T.time = S.timestamp
- <refreshing policy>

\* TIPS:

mysql does NOT support materialized view;

PostgreSQL, Oracle and SQLserver(微软的) supports materialized view

但是你可以直接在mysql写:

create table ...

# Views

- A view is a “virtual table,” a relation that is defined in terms of the contents of other tables and views.
- Declare by:  
`CREATE VIEW <name> AS <query>;`
- In contrast, a relation whose value is really stored in the database is called a *base table*.

# Example: View Definition

- CanDrink(drinker, beer) is a view “containing” the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer.
- Recall Frequents(drinker, bar), Sells(bar, beer, price)

```
CREATE VIEW CanDrink AS
SELECT distinct drinker, beer
FROM Frequents, Sells
WHERE Frequents.bar = Sells.bar;
```

# Example: Accessing a View

- You may query a view as if it were a base table.
  - There is a limited ability to modify views if the modification makes sense as a modification of the underlying base table.
- Example:  

```
select beer from CanDrink  
where drinker = 'Bill';
```

# What Happens When a View Is Used?

- The DBMS starts by interpreting the query as if the view were a base table.
  - Typical DBMS turns the query into something like **relational algebra**.
- The queries defining any views used by the query are also replaced by their algebraic equivalents, and “spliced into” the expression tree for the query.

When a query involves a view, the DBMS has two options:

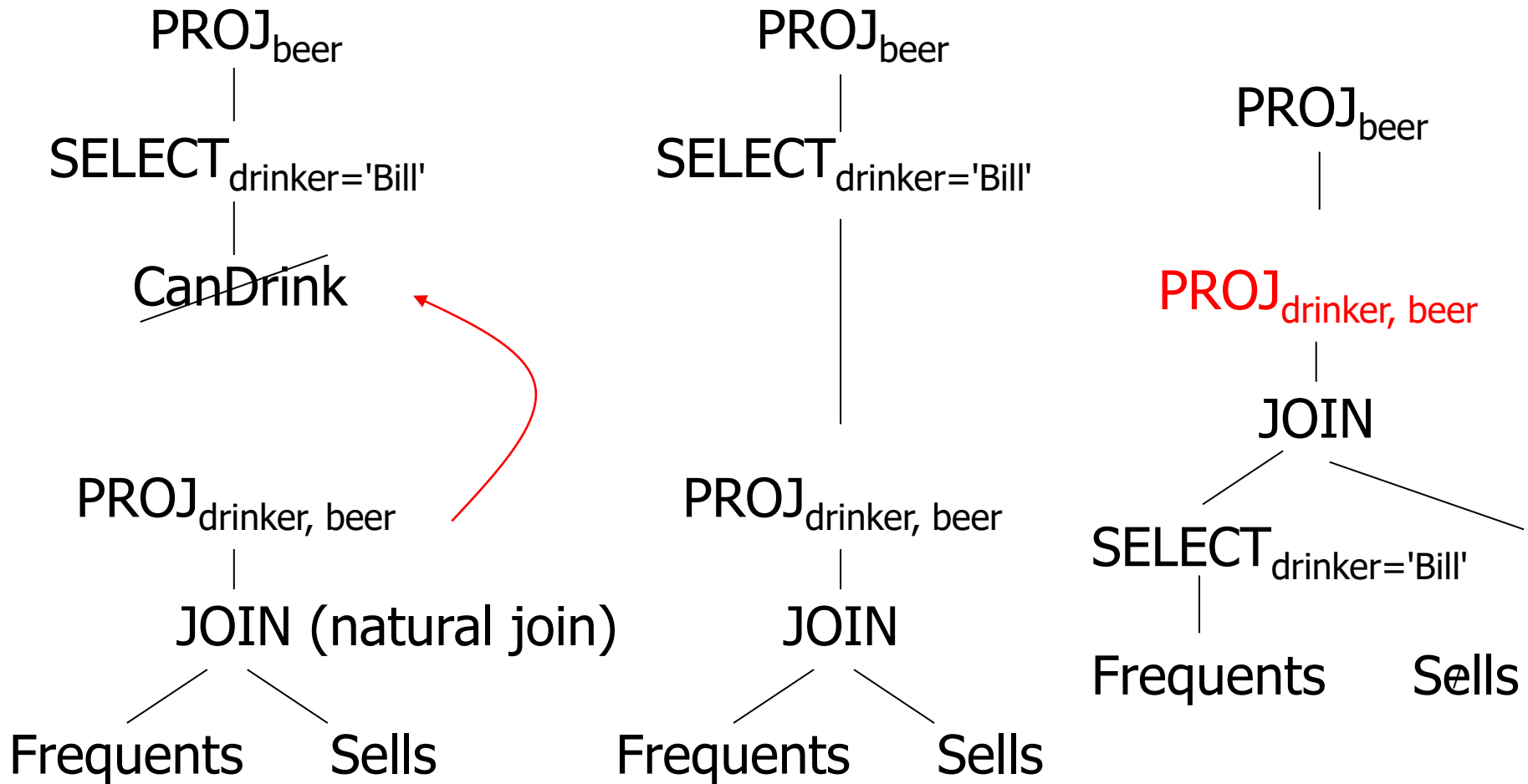
1. Execute the view definition query first, materialize the results, and then execute the outer query on top of the materialized view results.

2. Replace (inline) the view definition with its equivalent algebraic expression and merge it with the outer query to form a single algebraic expression tree.

This process of inlining or merging the view definitions into the outer query is known as view merging or view flattening. It allows the DBMS to optimize the entire query as a single expression tree, potentially leading to more efficient query execution plans.



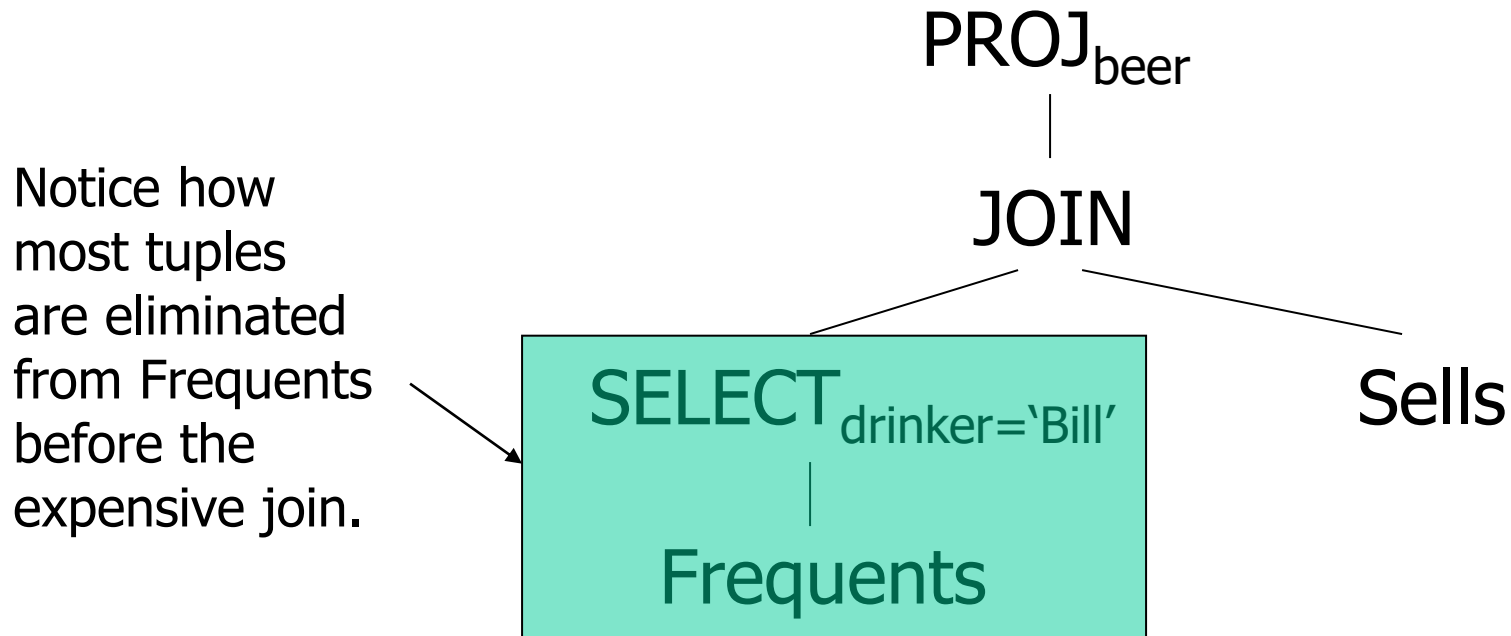
# Example: View Expansion



# DMBS Optimization

- It is interesting to observe that the typical DBMS will then “optimize” the query by transforming the algebraic expression to one that can be executed faster.
- Key optimizations:
  1. Push selections down the tree.
  2. Eliminate unnecessary projections.

# Example: Optimization



# More Examples: Defining Views

Views are relations, except that they are not physically stored.

Can be used for presenting different information to different users

**Employee**(ssn, name, department, project, salary)

```
CREATE VIEW Developers AS  
  SELECT name, project  
  FROM Employee  
  WHERE department = 'Development'
```

Payroll has access to all **Employees**, others only to **Developers**

# A Different View

Purchase(buyer, seller, product, store, price) 1000 rows

Product(name, maker, category) 100 rows

Person(name, city, phone)

|Purchase **Join** Product| = 900?

```
CREATE VIEW LA-view AS
```

```
SELECT buyer, seller, product, store
```

```
FROM Person, Purchase
```

```
WHERE Person.city = 'LA' AND  
       Person.name = Purchase.buyer
```

We have a new virtual table:

LA-view(buyer, seller, product, store)

# A Different View

LA-view(buyer, seller, product, store)

We can later use the view:

```
SELECT name, store
FROM LA-view, Product
WHERE LA-view.product = Product.name AND
      Product.category = 'shoes'
```

# What Happens When We Query a View ?

Recall: LA-view(buyer, seller, product, store)

```
SELECT name, LA-view.store
FROM   LA-view, Product
WHERE  LA-view.product = Product.name AND
       Product.category = 'shoes'
```



View expansion

```
SELECT name, Purchase.store
FROM   Person, Purchase, Product
WHERE  Person.city = 'LA' AND
       Person.name = Purchase.buyer AND
       Purchase.product = Product.name AND
       Product.category = 'shoes'
```

# Types of Views

- Virtual views:
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Precomputed offline – fast at runtime
  - Common in data warehouses (data cube)
    - Fact table + dimension tables
  - May have stale data



what is OLA?  
On-Line Analytical.

# Reusing a Materialized View

- Suppose I have **only** the result of LAView:

```
SELECT buyer, seller, product, store
FROM   Person, Purchase
WHERE  Person.city = 'LA'   AND
       Person.name = Purchase.buyer
```

- and I want to answer the query

```
SELECT buyer, seller
FROM   Person, Purchase
WHERE  Person.city = 'LA'   AND
       Person.name = Purchase.buyer AND
       Purchase.product='gizmo'
```

Can I answer the query using only the view?

# Query Rewriting Using Views

Rewritten query:

```
SELECT buyer, seller
FROM    LAView
WHERE   product= 'gizmo'
```

Original query:

```
SELECT buyer, seller
FROM    Person, Purchase
WHERE   Person.city = 'LA'   AND
        Person.name = Purchase.buyer AND
        Purchase.product='gizmo'.
```



# Another Example

- I still have **only** the result of LAView:

```
SELECT buyer, seller, product, store
FROM   Person, Purchase
WHERE  Person.city = 'LA'   AND
       Person.name = Purchase.buyer
```

- but I want to answer the query

```
SELECT buyer, seller
FROM   Person, Purchase
WHERE  Person.city = 'LA'   AND
       Person.name = Purchase.buyer AND
       Person.phone LIKE '206 543 %'
```

no you cannot

# And Now?

- I still have **only** the result of (slightly different) LAView:

```
SELECT buyer, seller, product, store
FROM Person, Purchase, Product
WHERE Person.city = 'LA' AND
      Person.name = Purchase.buyer AND
      Purchase.product = Product.name
```

- but I want to answer the query

```
SELECT buyer, seller
FROM Person, Purchase
WHERE Person.city = 'LA' AND
      Person.name = Purchase.buyer
```

No.

Because it's possible that  
by joining the Product table, the view created  
will lose some data.  
(like when `Product.name != Purchase.product`)

But when `Product.name` is the foreign key of  
`Purchase.product`, no data will be lost. It will  
work.

In summary, it depends on whether  
`Product.name` is a foreign key or not.

# And Now?

the name of the view is SBS



- I still have **only** the result of view SBS:

SELECT seller, buyer, Sum(Price) sp

FROM Purchase

WHERE Purchase.store = 'The Bon'

Group By seller, buyer

- but I want to answer the query

SELECT seller, Sum(Price)

FROM Purchase

WHERE Purchase.store = 'The Bon'

Group By seller

Answer:

Select seller, sum(sp)

From SBS

Group by seller

And what if it's the other way around?

# Example (OLAP)

Materialized view SBS(seller, buyer, sp)

Seller	Buyer	Sum(price) sp
David	Bill	10
David	Jennifer	20
David	Steve	10
Bill	David	20
Bill	Mary	10

Roll-up

Query:

Seller	Sum(price)
David	? 40
Bill	? 30



# Finally...

- I still have **only** the result of:

```
SELECT seller, buyer, Count(*) cnt
FROM Purchase
WHERE Purchase.store = 'The Bon'
Group By seller, buyer
```

- but I want to answer the query

```
SELECT seller, Count(*)
FROM Purchase
WHERE Purchase.store = 'The Bon'
Group By seller
```

# Example

View SBC(seller, buyer, cnt)

Seller	Buyer	count(*) cnt
David	Bill	2
David	Jennifer	4
David	Steve	2
Bill	David	5
Bill	Mary	2

Query:

Seller	count(*)
David	?? 8
Bill	? 7

Answer:  
select seller, sum(cnt)  
from SBC  
group by seller

Select se  
From SB  
Group by