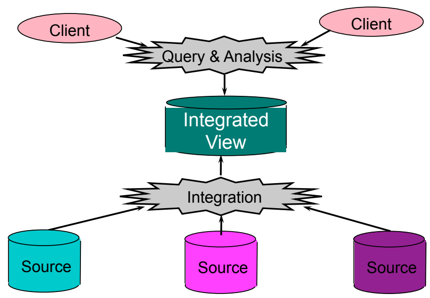
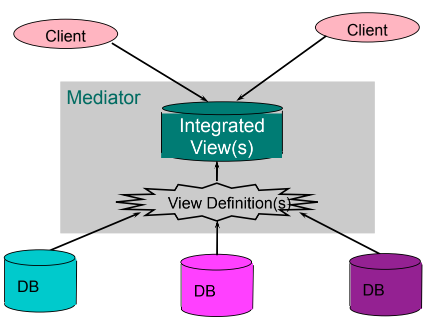
Warehouses & Virtual Databases offer Integrated & Added-Value Views

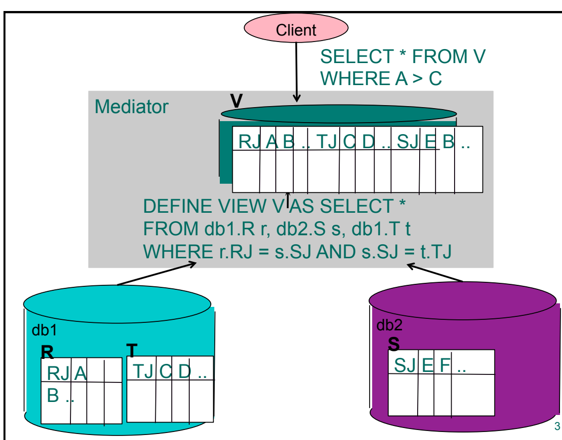


1

Focus: Sources are relational DBs. Integration specified by distributed view definition(s). Clients issue queries on views.

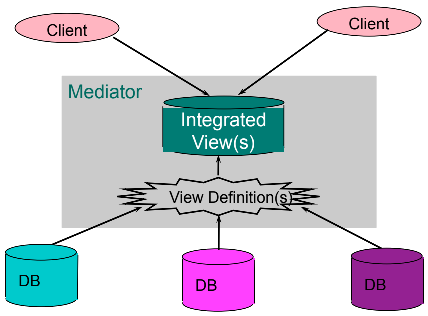


2



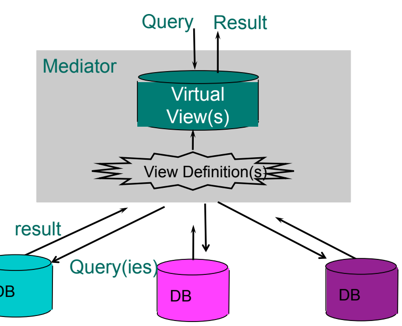
3

Virtual View -> Mediator is a Distributed Query Processor
Materialized view (warehouse) -> Mediator actually stores integrated view



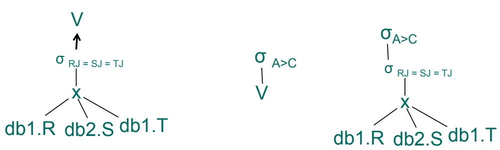
4

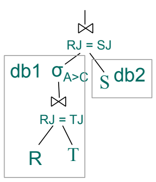
Distributed Query Processing in Mediators



5

Distributed Query processing





Translate subplans to SQL

+ How will the join happen?

- What if the source offers multiple data services instead of JDBC access?

6

Distributed Join Types

- Mediator-based Join
  - Ship results of queries at mediator
- Parameterized Join
  - Right subquery is enhanced with selection on join attribute
  - For each join value of left hand side, execute another right subquery
- Data Ship Join
  - Insert the result of left hand side (lhs) in the db of right hand side (rhs).
  - Execute join at db of right hand side
- Semijoin Reduction Join
  - Send rhs parameters to lhs
  - (Data ship alike variation) Lhs sends to rhs the semijoin of its subquery with the parameters set.
  - Execute join at db of rhs
  - Also, variation that looks like mediator-based join

7

Virtual Views Vs Materialized Views

View kind	CREATE VIEW V AS SELECT G, SUM(A) AS S FROM R GROUP BY G	CREATE MATERIALIZED VIEW V AS SELECT G, SUM(A) AS S FROM R GROUP BY G
Upon Updating R	Database does nothing	(Ideally) Database must refresh V to reflect changes on R
Upon Querying V	Optimize & run $\sigma_{G=5}$ $\gamma_{G;SUM(A)\to S}$ R	SELECT * FROM V WHERE G = 5 $\sigma_{G=5}$ V

8

Recompute Vs Incremental (Materialized View) Maintenance – Informal Example

CREATE MATERIALIZED VIEW V AS
SELECT G, SUM(A) AS S
FROM R
GROUP BY G

(start)
INSERT INTO R (…);
…
INSERT INTO R (…);
…
INSERT INTO R (…);
commit

At the end of the transaction, we want V to reflect the new state of R
Option 1: Delete and Recompute V
Option 2: Incrementally maintain V

$\triangle R^{+}$: the set of tuples inserted in R (obtained by log or other mechanism)

Option 1:

```
DELETE FROM V
 WHERE true;
INSERT INTO V
  (SELECT G, SUM(A) AS S
   FROM R
   GROUP BY G);
```

Option 2:

```
UPDATE V
SET S = S +
   (SELECT SUM(A)
     FROM △R+
     WHERE △R+.G = V. G)
WHERE V.G IN (SELECT G FROM △R+);
INSERT INTO V
 (SELECT G, SUM(A) AS S FROM △R+
  WHERE NOT G IN (SELECT G FROM V)
  GROUP BY G);
```

9

Capturing IVM as computation of $\triangle V^+$, $\triangle V^-$

- ignore (just for simplicity) update commands
  - think of update as delete – insert combo
- input is $\triangle R^+$, $\triangle R^-$
- compute "tuples to be deleted from the view" $\triangle V^-$
- compute "tuples to be inserted in the view" $\triangle V^+$
- delete $\triangle V^-$ from V, insert $\triangle V^+$ in V

```
CREATE MATERIALIZED
  VIEW V AS
SELECT G, SUM(A) AS S
FROM R
GROUP BY G
```

choose(a,b) returns *a* if *a* is NOT NULL, returns *b* if *a* is NULL
n0(a) returns *a* if *a* is NOT NULL, *0* otherwise

```
WITH RGplus AS
( SELECT G, SUM(A) AS S
  FROM △R+ GROUP BY G ),
RGminus AS
 (SELECT G, SUM(A) AS S
  FROM △R- GROUP BY G),
RGnet AS
 (SELECT choose(p.G, m.G) AS G,
         n0(p.S) – n0(m.S) AS S
   FROM RGplus AS p FULL OUTER
     JOIN RGminus AS m ON p.G=m.G
△V- AS
 ( SELECT * FROM V
   WHERE G IN
     (SELECT G FROM RGnet))
△V+ AS
 ( SELECT r.G AS G,
         n0(V.S) + r.S AS S
    FROM (V RIGHT OUTER JOIN
         RGnet AS r ON V.G=r.G)
```

IVM: Incremental (Materialized) View Maintenance. Eager version.

Snapshot 0

Problem: Find efficient view updates
$\triangle \mathbf{V}^+ = f^+(\triangle \mathbf{R}_1^+,\ldots,\triangle \mathbf{R}_n^+$
$\triangle \mathbf{R}_1^-,\ldots,\triangle \mathbf{R}_n^-, \mathbf{V}^0$
$\mathbf{R}_1^0,\ldots,\mathbf{R}_n^0)$
$\triangle \mathbf{V}^- = f^-(\ldots)$

Snapshot 1

View
$\mathbf{V}^0 = V(\mathbf{R}_1^0, \ldots,\mathbf{R}_n^0)$ → $\mathbf{V}^1 = V(\mathbf{R}_1^1, \ldots,\mathbf{R}_n^1)$

Database tables
$\mathbf{R}_1^0, \ldots,\mathbf{R}_n^0$ → $\mathbf{R}_1^1, \ldots,\mathbf{R}_n^1$

Table Updates
$\triangle \mathbf{R}_1^+, \ldots, \triangle \mathbf{R}_n^+$
$\triangle \mathbf{R}_1^-, \ldots, \triangle \mathbf{R}_n^-$

From logs or intercepted by triggers

IVM: Deferred version

Snapshot 0

Problem: Find efficient view updates
$\triangle \mathbf{V}^+ = f^+(\triangle \mathbf{R}_1^+,\ldots,\triangle \mathbf{R}_n^+$
$\triangle \mathbf{R}_1^-,\ldots,\triangle \mathbf{R}_n^-, \mathbf{V}^0$
$\mathbf{R}_1^1,\ldots,\mathbf{R}_n^1)$
$\triangle \mathbf{V}^- = f^-(\ldots)$

Snapshot 1

View
$\mathbf{V}^0 = V(\mathbf{R}_1^0, \ldots,\mathbf{R}_n^0)$ → $\mathbf{V}^1 = V(\mathbf{R}_1^1, \ldots,\mathbf{R}_n^1)$

Database tables
$\mathbf{R}_1^0, \ldots,\mathbf{R}_n^0$ → $\mathbf{R}_1^1, \ldots,\mathbf{R}_n^1$

Table Updates
$\triangle \mathbf{R}_1^+, \ldots, \triangle \mathbf{R}_n^+$
$\triangle \mathbf{R}_1^-, \ldots, \triangle \mathbf{R}_n^-$

From logs or intercepted by triggers

IVM: Self-maintaining version (not always possible)

Snapshot 0

Problem: Find efficient view updates

$\triangle \mathbf{V}^{+} = f^{+}(\triangle \mathbf{R}_1^{+},\ldots,\triangle \mathbf{R}_n^{+}, \triangle \mathbf{R}_1^{-},\ldots,\triangle \mathbf{R}_n^{-}, \mathbf{V}^0)$

$\triangle \mathbf{V}^{-} = f^{-}(\ldots)$

Snapshot 1

View

$\mathbf{V}^0 = V(\mathbf{R}_1^0, \ldots, \mathbf{R}_n^0) \longrightarrow \mathbf{V}^1 = V(\mathbf{R}_1^1, \ldots, \mathbf{R}_n^1)$

Database tables

$\mathbf{R}_1^0, \ldots, \mathbf{R}_n^0 \longrightarrow \mathbf{R}_1^1, \ldots, \mathbf{R}_n^1$

Table Updates

$\triangle \mathbf{R}_1^{+}, \ldots, \triangle \mathbf{R}_n^{+}$

$\triangle \mathbf{R}_1^{-}, \ldots, \triangle \mathbf{R}_n^{-}$

From logs or intercepted by triggers

Basic IVM Algorithm: Compose operator IVM rules

Example (wlog deferred, i.e., R means R^1 and S means S^1)

- Rule for V= R ⋈ S
  - $\triangle \mathbf{V}^{+} = ((\triangle \mathbf{R}^{+} \bowtie S) \; \mathbf{U} \; (R \bowtie \triangle \mathbf{S}^{+})) - (\triangle \mathbf{R}^{+} \bowtie \triangle \mathbf{S}^{+})$
  - $\triangle \mathbf{V}^{-}$ = ???
- Rule for $V = \sigma_c R$
  - $\triangle \mathbf{V}^{+} = \sigma_c \triangle \mathbf{R}^{+}$
  - $\triangle \mathbf{V}^{-}$ = ???
- Composition of rules leads to solutions for

  $V = T \bowtie \sigma_{A>5} W$

$\triangle \mathbf{V}^{+} = \ldots$

$\triangle \mathbf{V}^{-} = \ldots$

- May rewrite initial expression

14

IVM with Caching

- May associate intermediate views (caches) with subexpressions
- Bottom-up: From updating caches to reaching the materialized view
- Caches will typically needed indices
- Caches may or may not pay off as they incur cost for maintaining them (and their indices)

15

Generalizations

- Multiple views
  - self maintenance may involve a view utilizing the other views in its computation
- Genuine updates
  - Not simulated via insertions/deletions
- Insertions, deletions, updates on tables and views expressed as DML statements

Comparisons

Materialized View

- High query performance
- Queries not visible outside warehouse
- Local processing at sources unaffected
- Can operate when sources unavailable
- Extra information at warehouse
  - Modify, summarize (store aggregates)
  - Add historical information

Virtual View

- No need for yet another database
- More up-to-date data
  - Depending on specifics of IVM
- Query needs can be unknown
- Only query interface needed at sources

=> Lower Total Cost of Ownership

Performance revisited: What if indices are not enough for decent online performance?

- Buy RAM
- Use a column database
  - In analytics queries can give a 10x easily
- Scalable, parallel processing
  - Mostly via no SQL
- **Precompute**
  - **Fast answers!**
  - **Penalty: Cost of maintaining precomputed results**
  - **Applicability depends on schema and queries**
  - **Star schemas and summation are a good (but not the only) target of precomputation**

Precomputation problems

Steps:

1. Choose what data to precompute
2. Use the precomputed data smartly in your queries
3. Update smartly the precomputed data as the database changes (IVM)

Tradeoff:

- Precomputed data accelerate analytics => faster queries
- But need to be updated => cost

Example: Precomputation and its Use

Database has huge table `Sales(product, store, date, amt`

Application issues often this slow query and displays the results

```
SELECT product, SUM(amt) AS sumamt
FROM Sales
GROUP BY product
```

To improve performance we precompute table `ProductSales(product, sumamt)`

and insert in it the precomputed data by

```
INSERT INTO ProductSales (
SELECT product, SUM(amt) AS sumamt
FROM Sales
GROUP BY product )
```

Now the application issues instead this fast query below

```
SELECT *
FROM ProductSales
```

Example (cont'd)

Now we have to keep up to date the `ProductSales(product, sumamt)`

as new sales happen. E.g., if another $10 of product 23 were just s

```
UPDATE ProductSales
SET sumamt = sumamt + 10
WHERE product = 23
```

(in actual code it will use prepared queries)

You do not need the “exact” view

- Consider V1(Product,Customer,Sales) and V2(Product,Customer,Date,Sales) are precomputed
- ProductSales is not precomputed
- You need to answer the query

SELECT product, SUM(amt)
FROM SALES
GROUPBY product

- Write it in an alternate way, using one of the views in the most efficient way

Star Schemas

product

prodID	name	price
p1	bolt	10
p2	nut	5

store

storeID	city
c1	nyc
c2	sfo
c3	la

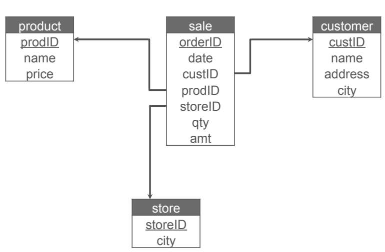
sale

orderID	date	custID	prodID	storeId	qty	amt
o100	1/7/97	53	p1	c1	1	12
o102	2/7/97	53	p2	c1	2	11
105	3/8/97	111	p1	c3	5	50

customer

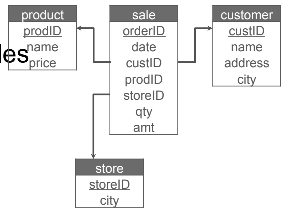
custID	name	address	city
53	joe	10 main	sfo
81	fred	12 main	sfo
111	sally	80 willow	la

Star Schema

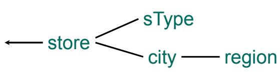


Terms

- Fact table
- Dimension tables
- Measures



Dimension Hierarchies



store	storeId	cityId	tId	mgr
	s5	sfo	t1	joe
	s7	sfo	t2	fred
	s9	la	t1	nancy

sType	tId	size	location
	t1	small	downtown
	t2	large	suburbs

city	cityId	pop	regId
	sfo	1M	north
	la	5M	south

region	regId	name
	north	cold region
	south	warm region

- Snowflake Schema
- Constellations

Cube

Fact table view

sale	prodId	storeId	amt
	p1	c1	12
	p2	c1	11
	p1	c3	50
	p2	c2	8

↔

Multi-dimensional cube

	c1	c2	c3
p1	12		50
p2	11	8	

dimensions = 2

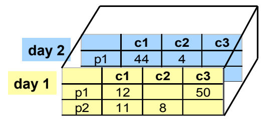
for

3-D Cube

Fact table view

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

Multi-dimensional cube



dimensions = 3

Aggregates on Slices

- Add up amounts for day 1
  - SELECT sum(amt) FROM SALE
  - WHERE date = 1

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

➡ 81

Aggregates

- Add up amounts by day
  - SELECT date, sum(amt) FROM SALE
  - GROUP BY date

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

➡

ans	date	sum
	1	81
	2	48

Another Example

- Add up amounts by day, product
  - SELECT date, sum(amt) FROM SALE
  - GROUP BY date, prodI

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

sale	prodId	date	amt
	p1	1	62
	p2	1	19
	p1	2	48

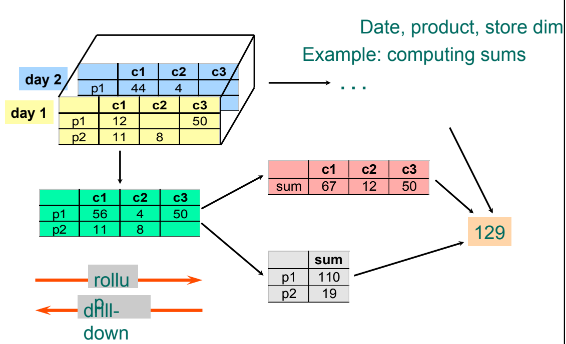
rollup →

← drill-down

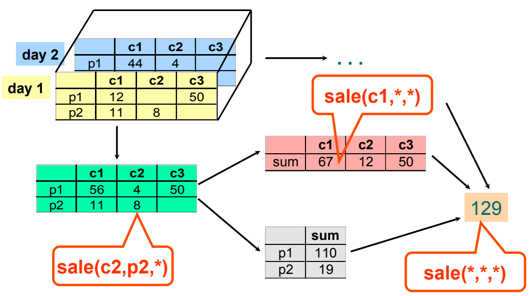
Aggregates

- Operators: sum, count, max, min, median, avg
- "Having" clause
- Using dimension hierarchy
  - average by region (within store)
  - maximum by month (within date)

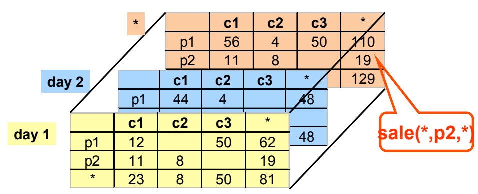
Cube Aggregation



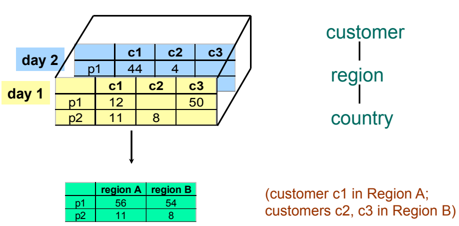
Cube Operators



Extended Cube

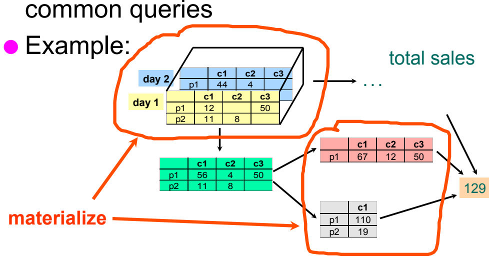


Aggregation Using Hierarchies



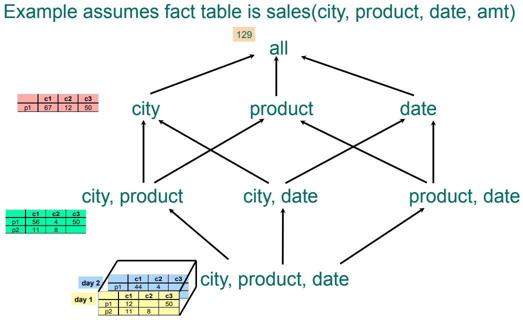
What to Materialize?

- Store in warehouse results useful for common queries
- Example:



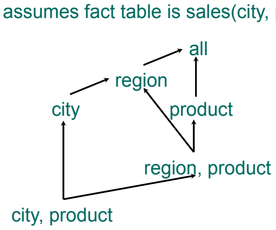
Cube Aggregates Lattice

Example assumes fact table is sales(city, product, date, amt)



Cube Aggregates Lattice

Example assumes fact table is sales(city, product, amt) and cities c



Should one precompute joins?

- Notice that we have featured foreign keys, not printable values. Why?
- Why (city product) and not (city region product)?
- Minor penalty to find the cities of a particular region
- Probably larger penalty by having a larger table
  - Think space in storage and time to scan it