Views

Knowledge objectives

1. Explain the differences between the three levels in the ANSI/SPARC architecture, paying special attention to physical and logical independency
2. Explain the two differences between a view and a table
3. Explain the difference between a view and a materialized view
4. Name four potential uses of views
5. Enumerate and distinguish the four problems associated to views
6. According to the standard, name the two properties a view must fulfill to be updatable
7. Enumerate when and how a materialized view can be refreshed
8. Discuss the benefits of a complete and an incremental view update
9. Enumerate the requirements for a query to be rewritten in terms of a materialized view
10. Identify the two problems in pre-computing queries

Understanding objectives

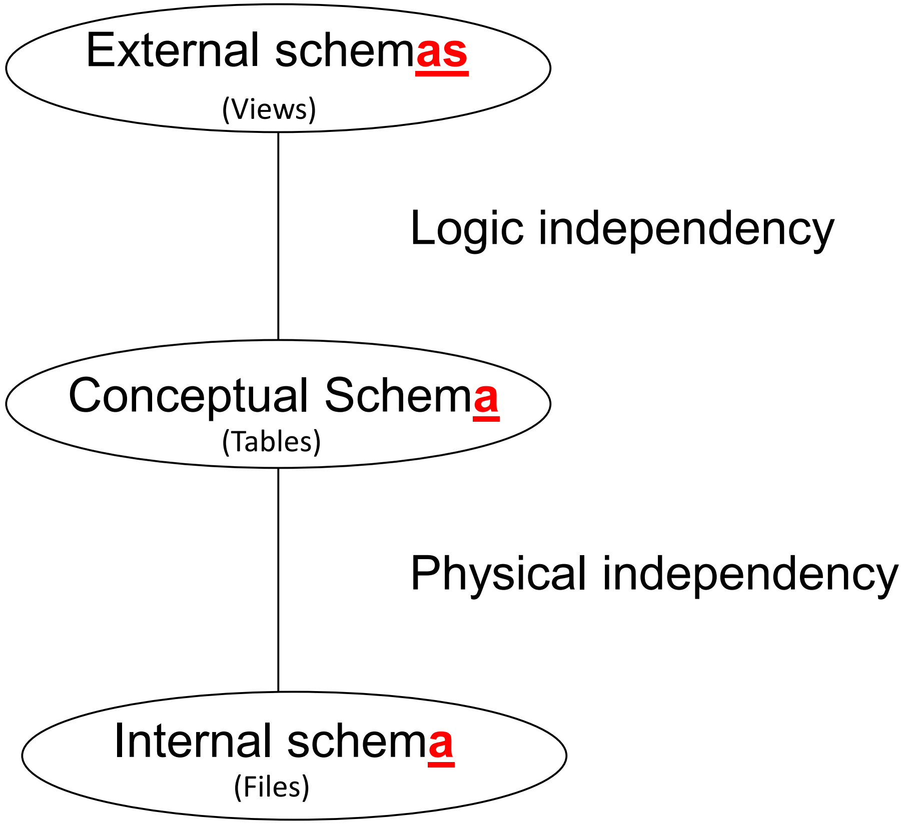
1. Given a set of tables and some views, apply view expansion over a query
2. Given the number of dimensions and levels in each of them, calculate the number of different materialized views in a multidimensional schema, considering only the cases in the group by clause
3. Select a set of views to be materialized using a greedy algorithm in the following scenarios:
   a) Disk space is limited and the system is read-only
      a) Only the given user queries can be materialized
      b) Any query can be materialized
   b) Disk space is not limited and the system is read-write

Application objectives

1. Given an assertion simulate its implementation using the materialized view syntax in Oracle
2. Given a set of source tables (no more than 6) together with their statistics and some views over them (no more than 3), justify if
   a) A given view is updatable
   b) It is worth or not an incremental update of a materialized view in front of a complete one
   c) A specific query over the tables can be rewritten in terms of the materialized views

View definition and difficulties

ANSI/SPARC architecture



Alternatives to implement a relation

- From the structures / data point of view
  - Tables
    - Data in disk (i.e., Materialized)
  - [Non-materialized] views
    - Definition in catalog (SQL statement)
      - Re-executed with every query
  - Materialized views
    - Data in disk (i.e., Materialized) and definition in catalog (SQL statement)
- From data retrieval point of view
  - Tables
    - Querying the materialized data
  - [Non-materialized] views
    - Transforming the query into another one over the underlying tables

$$V = F(R_1, R_2, \dots , R_n)$$
$$Q(V) \rightarrow Q'(R_1, R_2, \dots, R_n)$$

  - Materialized views
    - Querying the pre-computed (i.e., Materialized) result of the query
      - Reduced to a synchronization problem

https://www.postgresql.org/docs/current/rules-materializedviews.html

Example in Data Warehousing

Dimension

STORES	
Id	city
1	Mataró
2	Mataró

Fact

SALES				
storeId	date	time	productId	euros
1	xxx	xxx	1	10
2	xxx	xxx	1	15
1	xxx	xxx	2	20

Dimension

PRODUCTS	
Id	product
1	Rubber
2	Pen

Measure

Materialized view

EUROSALES			
City	Product	sumEuros	salesCounter
Mataró	Rubber	25	2
Mataró	Pen	20	1

Aggregations

Potential uses

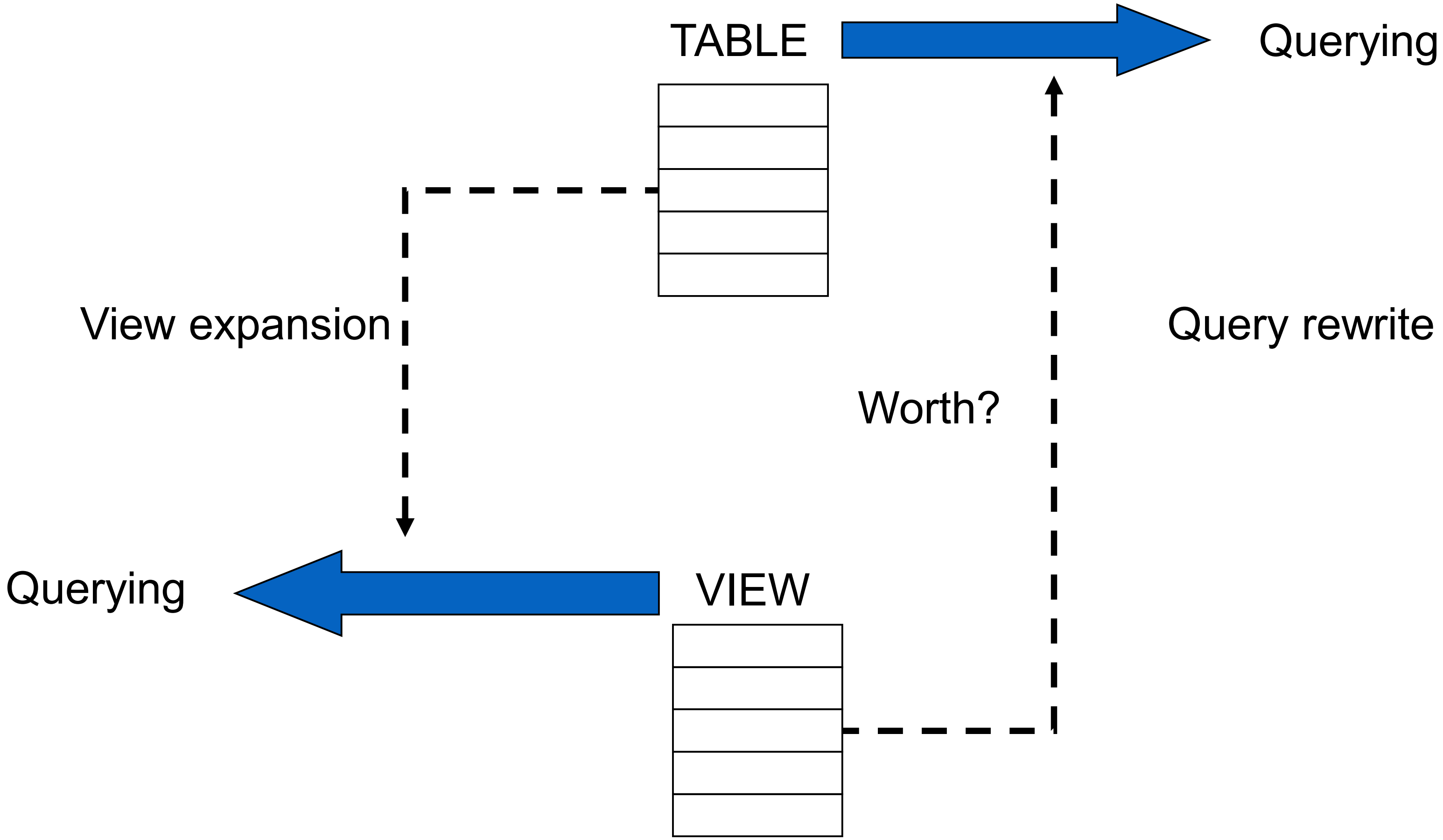
- Simplify complex schemas
  - Simplify queries
- Hide data / implementation details
  - Solve security issues
- Improve performance
  - Only if materialized
- Integrity checking

DBMS difficulties/features associated to views

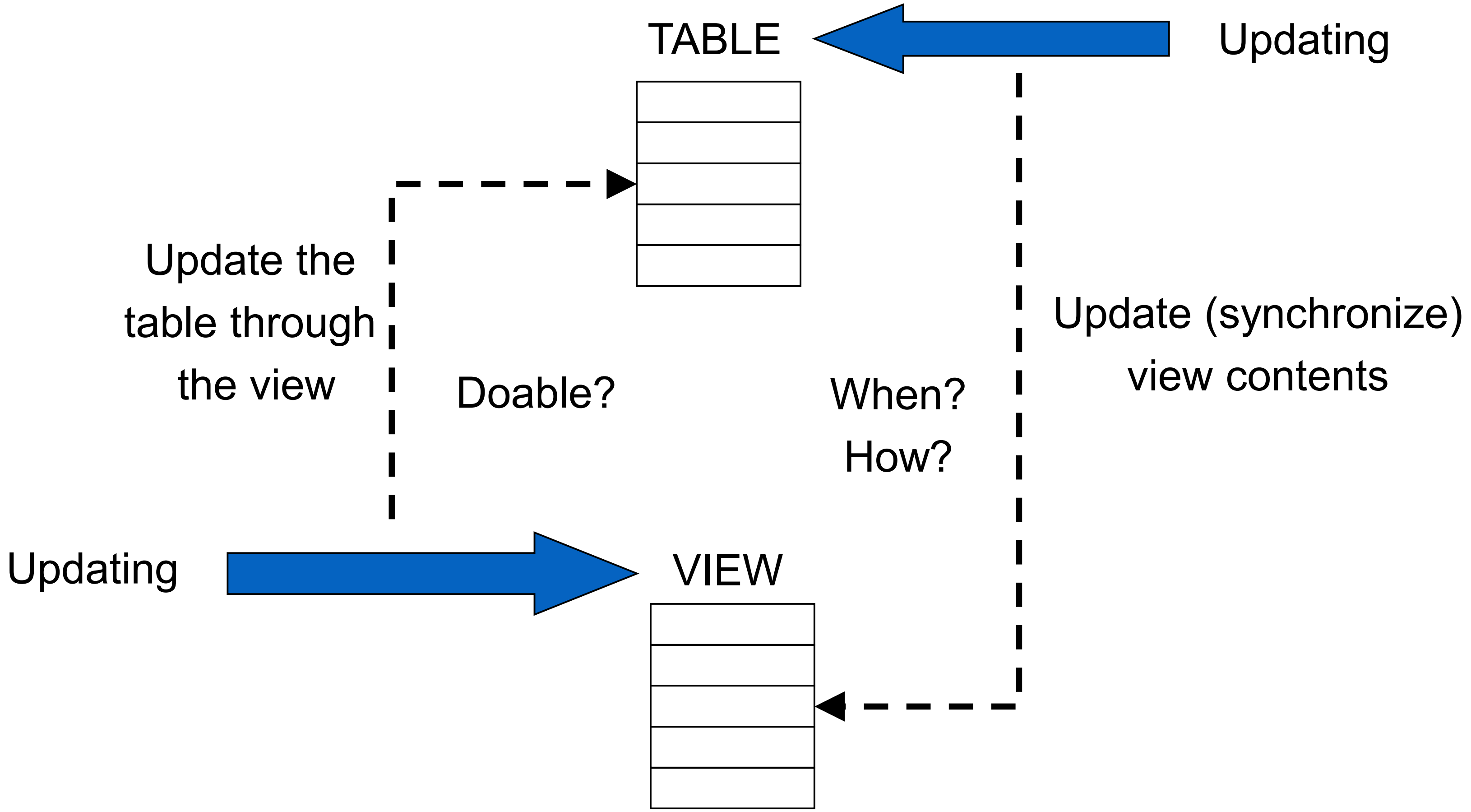
	Queries	Modifications
Over a View → Affect a Table	View expansion	Update through views
Over a Table → Affect a (Materialized) View	Query rewriting	View updating

- Non materialized views
  a) View expansion
    - Transform the query over the views into a query over the source tables
- Materialized views
  b) Query rewriting (i.e., answering queries using views)
    - Transform an arbitrary query over the tables into a query over the available views
  c) View updating
    - Changes in the sources are, potentially, propagated to the view
- Both
  d) Update through views
    - Propagate the changes in the view to the sources by means of a translation process

Views and queries



Views and modifications



Materialized views in Oracle (not in SQL:2023)

CREATE MATERIALIZED VIEW <name>
[BUILD {IMMEDIATE|DEFERRED}]
[REFRESH
[{NEVER|FAST|COMPLETE|FORCE}]
[{ON DEMAND|ON COMMIT|ON STATEMENT|NEXT <date>}]]
[FOR UPDATE]
[{DISABLE|ENABLE} QUERY REWRITE]
AS <query>;

Update through views

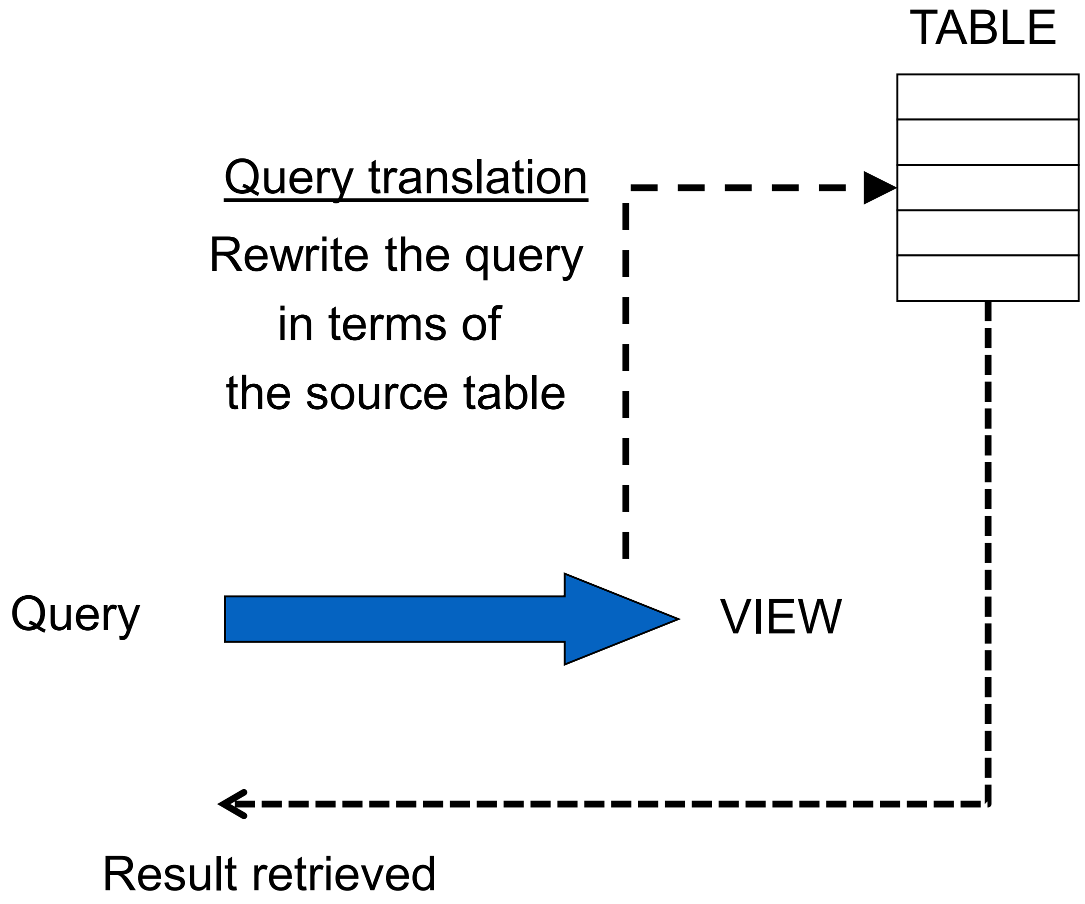
View updating

Answering queries using views

View expansion

View expansion problem

In case the query is over a view, a preliminary step to unfold the view definition is required (!)



View expansion solution

Do

1. Retrieve the view definition(s) from the catalog
2. Replace the view definition(s) in the query

While unresolved views in the query

Example of view expansion

```sql
CREATE VIEW richempl AS
    SELECT *
    FROM employees
    WHERE salary>30000;

SELECT AVG(salary)
FROM richempl;
```

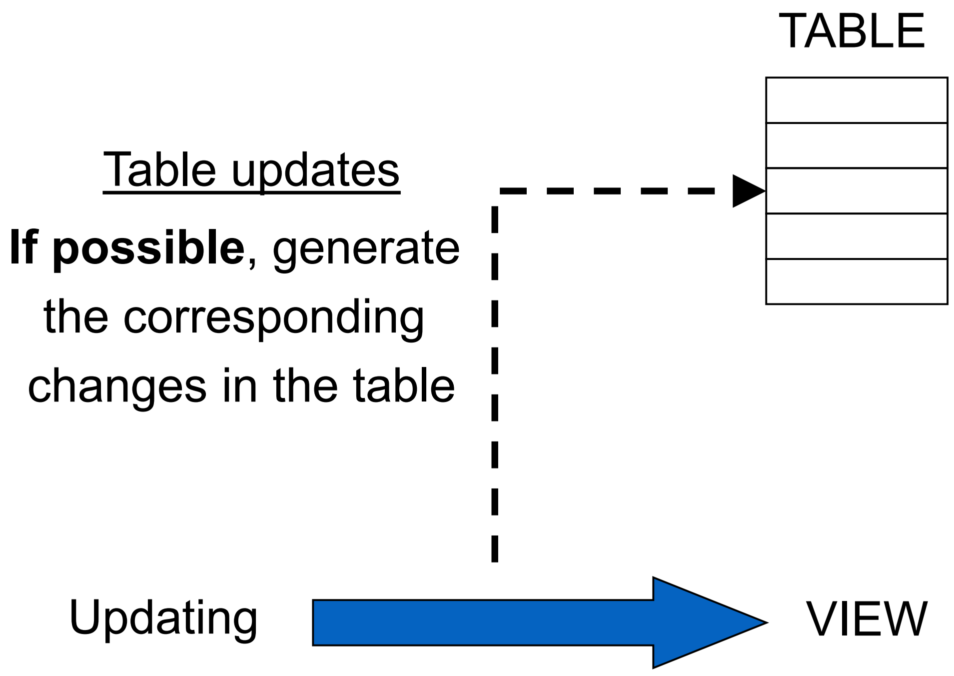
```sql
CREATE TABLE employees (
    dni CHAR(8),
    name CHAR(8),
    salary INTEGER,
    PRIMARY KEY (dni)
    );
```

```sql
SELECT AVG(salary)
FROM (
    SELECT *
    FROM employees
    WHERE salary>30000
    );
```

Update through views

Update through views problem

In case the update is over a view, it is still posible to modify the table to produce the desired effect, under some conditions (!)



Update through views solution

- Views, in general, are non-updatable
  - Only when the update can be translated **unambiguously** over the relational table
- According to the standard, views can be updated only if
  a) It is an algebraic selection on a single relation or updatable view
  b) It may also contain an algebraic projection iff the following are projected:
     - The primary key
     - All not-null atributes

Subqueries, joins or aggregate functions are not allowed!!!

Example of update through views with joins

```
CREATE VIEW sup-part_sup  AS
SELECT s.name, p.nsupp, p.part, p.qt
FROM supplier s, part_supplied  p
WHERE s.nsupp = p.supp;
```

supplier (nsupp, name)

nsupp	name
100	Joan

part_supplied (supp, part, qt)

supp	part	qt
100	p1	10
100	p2	20

sup-part_sup (name, supp, part, qt)

name	supp	part	qt
Joan	100	p1	10
Joan	100	p2	20

```
INSERT INTO sup-part_sup VALUES ('Pere',200,p1,20);
```

```
INSERT INTO supplier VALUES (200,'Pere');
INSERT INTO part_supplied VALUES (200,'p1',20);
```

```
UPDATE sup-part_sup SET name='Joana'
WHERE supp=100 AND part='p1';
```

```
DELETE FROM sup-part_sup WHERE supp=100 AND part='p1';
```

Example of update through views with aggregates

part_supplied (supp, part, qt)

supp	part	qt
sp1	p1	100
sp2	p1	200
sp2	p2	200

```
CREATE VIEW total_qt (part, total) AS
SELECT part, sum(qt)
FROM part_supplied
GROUP BY part;
```

total_qt (part, qt)

part	qt
p1	300
p2	200

```
INSERT INTO total_qt VALUES ('p3',400);
```

```
UPDATE total_qt SET qt=301 WHERE part='p1';
```

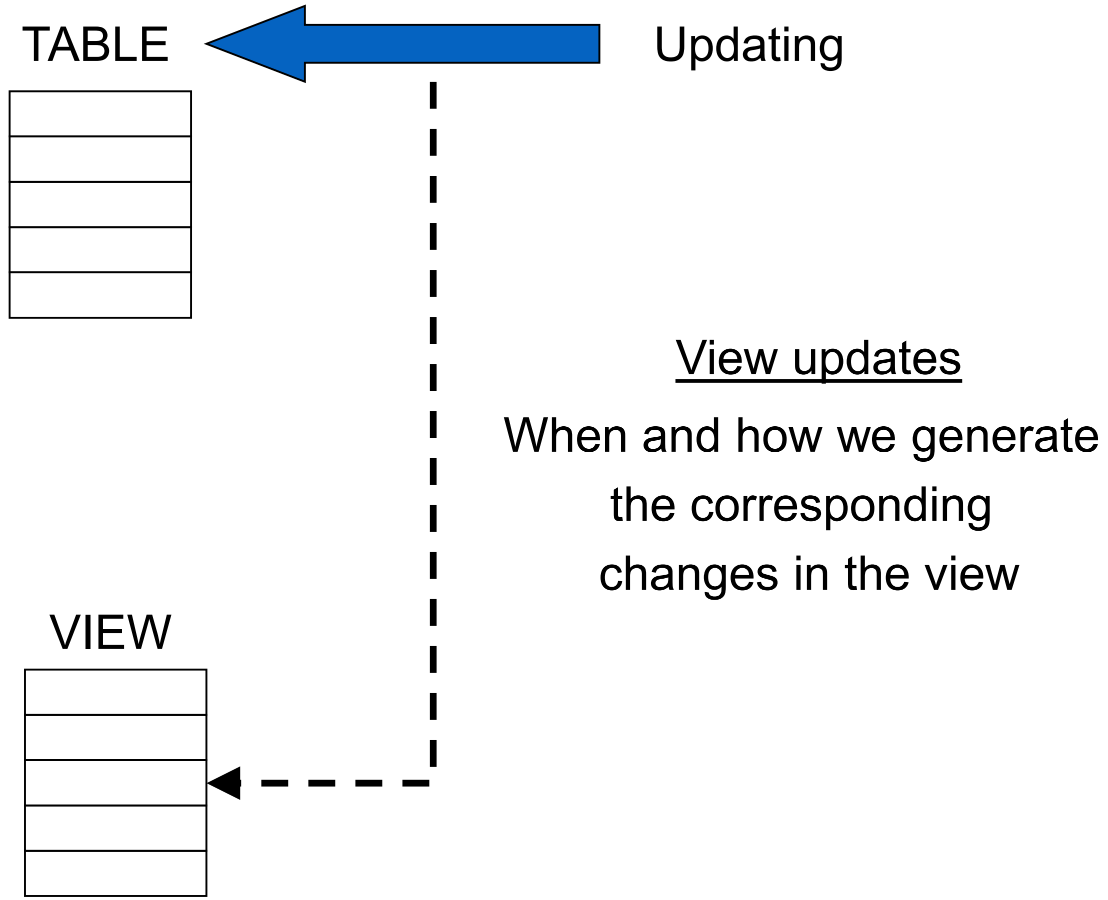
```
DELETE FROM total_qt WHERE part='p1';
```

```
DELETE FROM part_supplied
WHERE part='p1';
```

View updating

View updating problem

In case the update is over a table, we should eventually sinchronize the content of the view to avoid inconsistences (!)



View updating (when)

- On statement
  - Avoids the need of the associated log, but is very inefficient, in general
    a) All DML operations are slower, as they include the update of the MV
       - The same MV row can be updated several times in the same transaction

    b) The transaction cancellation is more expensive, as it includes undoing changes in the MV
- On commit
  - May still be too expensive
- On demand
  - Generates temporal inconsistences
- Next <date>
  - Generates temporal inconsistences

https://www.postgresql.org/docs/current/sql-refreshmaterializedview.html

View updating (how)

- Complete update
  - All instances are regenerated

    REFRESH MATERIALIZED VIEW <name>;

    - Clearly inefficient ?
    - Always possible
- Incremental update (called “fast” in Oracle)
  - Only instances that changed are regenerated
    - Much more efficient ?
    - Not always possible
      - Depends on the information available

https://www.postgresql.org/docs/current/sql-refreshmaterializedview.html

Incremental materialized view refresh (in Oracle)

- Incremental (called fast) updates allow on commit refreshment (otherwise not allowed)
- A log must be defined for every source table
  - Only one log per table is allowed!
  - Stores rows describing changes from last refresh
  - Tuples should be univocally identified (ROWID or PK needed)

```
CREATE MATERIALIZED VIEW LOG ON table
  [WITH [PRIMARY KEY,] [ROWID,] [SEQUENCE] (list_of_attr)]
  [{INCLUDING | EXCLUDING} NEW values]
```

- Both the log and the view definition query (Q') must fulfill a set of constraints depending on the kind of query
  - Basic queries (without groupings nor joins)
  - Join queries
  - Grouping queries

Oracle DataWarehousing Guide, chapter 9 “Basic Materialized Views”

Assertions

- A constraint that may involve several tuples or tables
  - Introduced in SQL:1992
  - They are, generically, not yet implementable in most RDBMS (!)
- Can be simulated using materialized views in some DBMS (e.g., Oracle)
  - An ON COMMIT refresh will be required, most of the times
    - ON DEMAND or NEXT may be enough in some cases
  - There are two alternative implementations:
    - a) Using an empty materialized view
      1) Define an MV with the negation of the desired assertion definition
      2) Define a dummy check, which should never be satisfied (i.e., an inconsistent IC)
    - b) Using a non-empty materialized view
      1) Define an MV with the desired assertion definition without the where/having condition
      2) Define a check corresponding to the negation of the where/having of the assertion

Example of assertion simulation (empty MV)

- Assertion (standard syntax):

```
CREATE ASSERTION IC_debt (NOT EXISTS
   (SELECT c.#customer
   FROM customers c, orders o
   WHERE c.#customer = o.#customer
     AND c.type = 'regular' AND o.payment = 'pending'
   GROUP BY c.#customer
   HAVING SUM(o.quantity) >= 10000));
```

- MV simulating the assertion (Oracle syntax):

```
CREATE MATERIALIZED VIEW mv BUILD IMMEDIATE REFRESH FAST ON COMMIT AS
   SELECT 'x' AS X
   FROM customers c, orders o
   WHERE c.#customer = o.#customer
     AND c.type = 'regular' AND o.payment = 'pending'
   GROUP BY c.#customer
   HAVING SUM(o.quantity) >= 10000))

ALTER TABLE mv ADD CONSTRAINT mv_check CHECK (X IS NULL);
```

Example of assertion simulation (non-empty MV)

- Assertion (standard syntax):

```
CREATE ASSERTION IC_debt (NOT EXISTS
  (SELECT c.#customer
  FROM customers c, orders o
  WHERE c.#customer = o.#customer
    AND c.type = 'regular' AND o.payment = 'pending'
  GROUP BY c.#customer
  HAVING SUM(o.quantity) >= 10000));
```

- MV simulating the assertion (Oracle syntax):

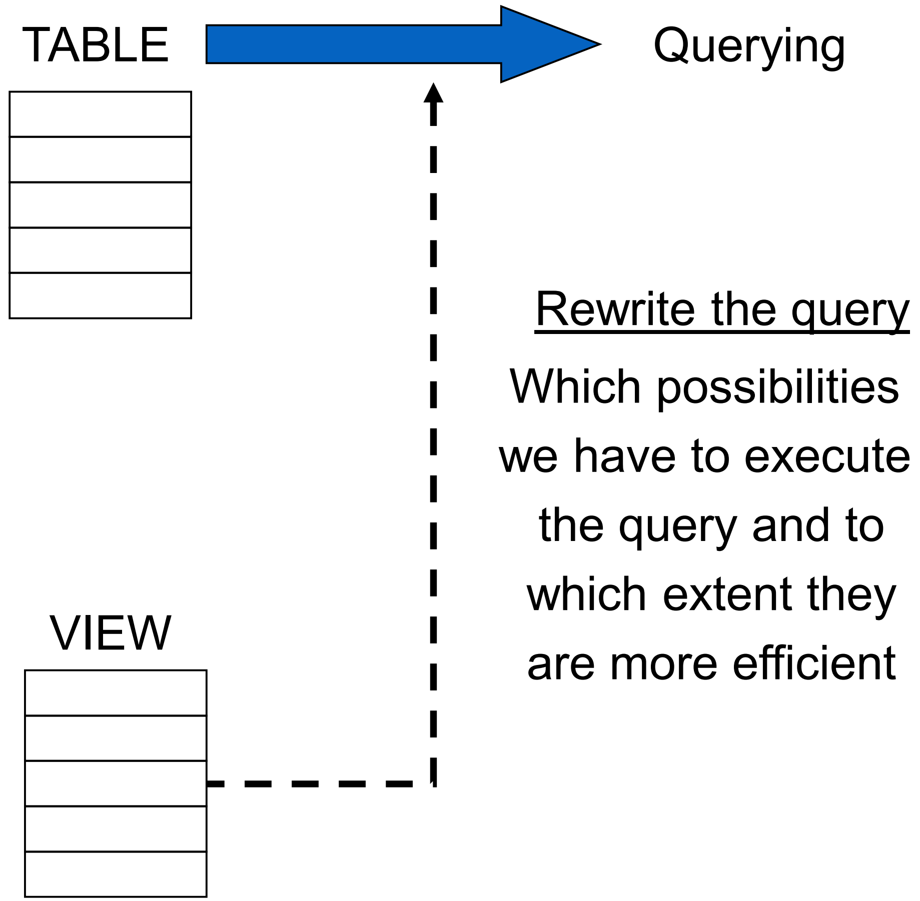
```
CREATE MATERIALIZED VIEW mv BUILD IMMEDIATE REFRESH FAST ON COMMIT AS
  SELECT c.#customer AS id, SUM(o.quantity) as debt
  FROM customers c, orders o
  WHERE c.#customer = o.#customer
    AND c.type = 'regular' AND o.payment = 'pending'
  GROUP BY c.#customer;

ALTER TABLE mv ADD CONSTRAINT mv_check CHECK (debt < 10000);
```

Query rewriting

Query rewriting problem

In case a regular query over a table, which has materialized views already defined on it, we could find execution strategies that use the materialized view(s) instead of the table (!)



Query rewriting solution

- Deciding whether it is possible to rewrite a query in terms of existing materialized views or not is computationally complex
  - Also known as answering queries using views
  - DBMSs restrict the search space to common cases by using rules
- Requirements:
  a) Query predicate must be subsumed by that of the view

     $$\text{Query tuples} \subseteq \text{View tuples}$$

  b) Aggregation level in the query must be higher or equal to that in the MV
     - Functional dependencies can be used to check it

  c) Aggregates must coincide with (or be computable from) the MV

Example of predicate subsumption requirement

Table

A	B	C	D
1			
2			
3			
4			
5			

Odd numbers

MView

A	B	C	D
1			
3			
5			

A IN (2,5)

A IN (2,5)

A IN (3,5)

Query 1

A	B	C	D
2			
5			

Query 2

A	B	C	D
3			
5			

Example of aggregation level requirement

Table

A	B	C	D
1	a	x	
1	a	y	
1	b	x	
2	a	x	
2	a	y	

GROUP BY A,B

MView

A	B	f(D)
1	a	
1	b	
2	a	

GROUP BY C

GROUP BY C

GROUP BY A

Query 1

C	f(D)
x	
y	

Query 2

A	f(D)
1	
2	

Example of aggregation function requirement

Table

A	B	C	D
1	a	x	
1	a	y	
1	b	x	
2	a	x	
2	a	y	

SUM

MView

A	B	SUM(D)
1	a	
1	b	
2	a	

COUNT(*)

COUNT(*)

SUM(D)

Query 1

A	COUNT(*)
1	
2	

Query 2

A	SUM(D)
1	
2	

Materialized view selection

To improve query performance …

… Pre-compute as much as possible

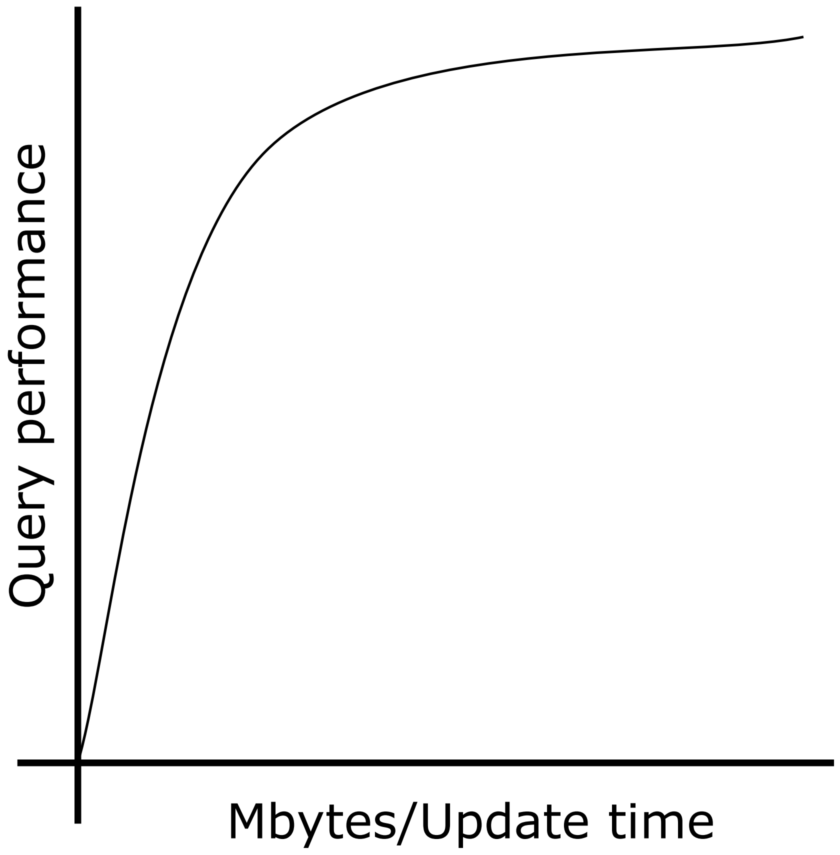
- Redundant “tables” (a.k.a. materialized views)
  - Less attributes
  - Less tuples
    - Only those fulfilling the query predicate
    - Only one per combination of values of attributes in the GROUP BY
  - Less space than the table
    - Less I/O to be accessed

Problems in pre-computing

- Cost
  - Space
  - Time
    - Query vs Modification frequency

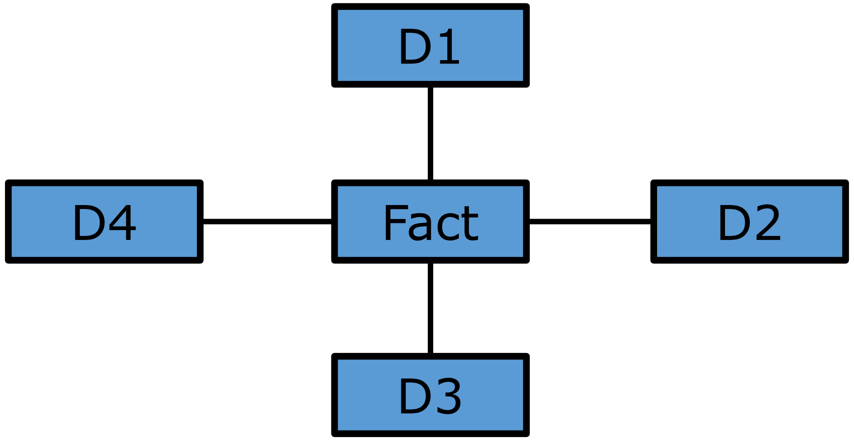
- Consistency and rewriting must be controlled
  - Using triggers
    - Advantages
      - Flexible
      - Allows rewriting of any query
      - Maybe efficient
    - Disadvantages
      - Complicates the management of the DBMS
      - Ad-hoc rewriting must be implemented for each query
        - Users are bound to our rewriting tools
  - Using MVs

Materialization trade-off



Aggretion levels combinatorial explosion

- Choosing the best combination of views to be materialized is NP-complex
  - A fact table with *D* dimension tables with *L* aggregation levels (excluding the “All” level) for each one, would generate $(L+1)^D$ possible MVs



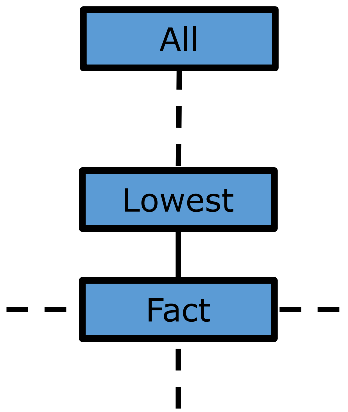
GROUP BY D1, D2, D3, D4;
GROUP BY D1, D2, D3;
GROUP BY D1, D2, D4;
GROUP BY D1, D2;
GROUP BY D1, D3, D4;
GROUP BY D1, D3;
…
GROUP BY ();

D1	D2	D3	D4
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

$(L+1)^D$

Solutions to the aggregation levels explosion

- The sparser the basic table/cube, (proportionally) the more space aggregates will use
  - Twelve days per year may generate twelve months per year
- Heuristics:
  - Materialize lower aggregation levels
    - They highly reduce the size (in absolute value) and solve many queries
  - Materialize higher aggregation levels
    - They are queried very often
  - Materialize a view if it solves a critical query or many queries
  - Do not materialize a view if it is a close successor of an already materialized view
    - Each tuple comes from the aggregation of at least 10



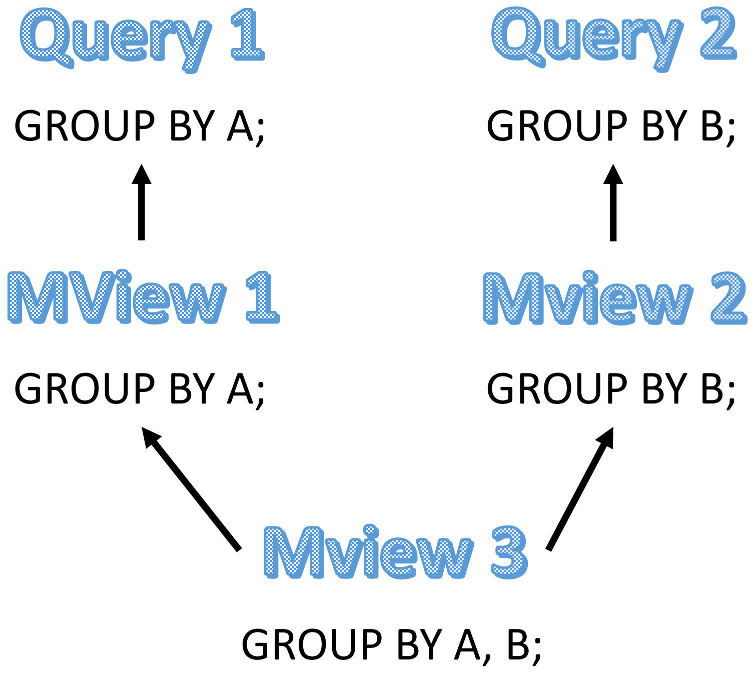
- Modify the set of MVs as user needs evolve

Candidate views to be materialized

Given a workload $W=\{q_1, q_2, q_3, \ldots\}$, and identifying queries by their GROUP BY clause, candidate views v_i are those that:

a) $GB(v_i) = GB(q_j)$

b) $GB(v_i) = \bigcup_{q_j \in Q} GB(q_j)$, where $Q \subseteq W$



Provided that:
1) Predicates also allow rewriting
2) Aggregations in the select clause also allow rewriting

Algorithm to choose among candidates

Greedy algorithm (guarantees 63% minimum improvement):

Do

1) Consider those candidate views that fit in the available space and update time
2) Sort views based on the performance improvement they induce
3) Materialize first view in the list, if it improves performance

While performance improved and there is available space and update time

- Modify the set of MVs as user needs evolve

Example of materialized view selection

Greedy

Example of materialized view selection (I)

- Table CentMilResp(ref, pobl, edat, cand, val)
- D=1sec; C=0
- $B_{CentMilResp}$=10.000; |CentMilResp|=100.000
- Ndist(pobl)=200; Ndist(edat)=100; Ndist(cand)=10
- All attributes require the same space
  - The control information (a.k.a. metadata) of the row requires as much space as another attribute
- Query frequencies:

35%: SELECT cand, MAX(val) FROM CentMilResp GROUP BY cand;

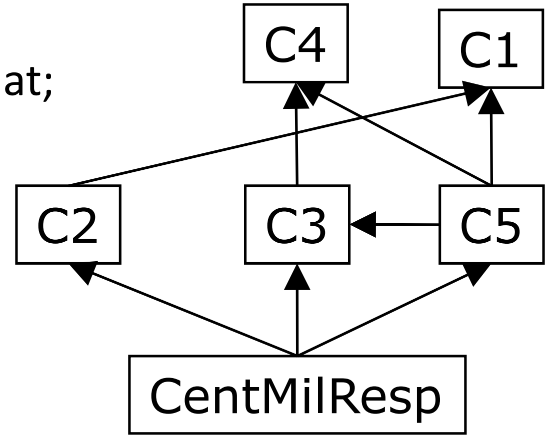
20%: SELECT cand, edat, AVG(val), MAX(val), MIN(val) FROM CentMilResp GROUP BY cand, edat;

20%: SELECT pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;

25%: SELECT pobl, MAX(val) FROM CentMilResp GROUP BY pobl;

- We have 10.140 disk blocks available

Example of materialized view selection (II)



C1/Q1 - SELECT cand, MAX(val) FROM CentMilResp GROUP BY cand;
C2/Q2 - SELECT cand, edat, AVG(val), MAX(val), MIN(val) FROM CentMilResp GROUP BY cand, edat;
C3/Q3 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;
C4/Q4 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY pobl;

C5 - SELECT cand, pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;

Aggregation rows estimation:

$$|C_i| = \min(|T|, Ndist(a_1) * \ldots * Ndist(a_n))$$

$$|C_2| = \min(100000,\ 10 * 100) = 1000$$
$$|C_3| = |C_5| = \min(100000,\ 10 * 200) = 2000$$

Materialized view space estimation:

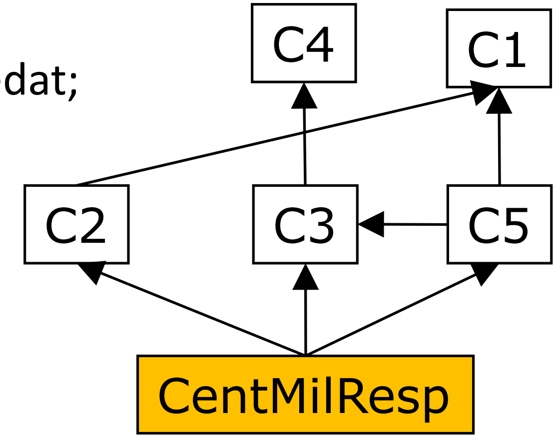
$$B_{Ci} = \lceil B_T * (Arity(C_i)/Arity(T)) * (|C_i|/|T|) \rceil$$

$$B_{C2} = \lceil 10000 * (6/6) * (1000/100000) \rceil = 100$$
$$B_{C3} = \lceil 10000 * (3/6) * (2000/100000) \rceil = 100$$
$$B_{C5} = \lceil 10000 * (4/6) * (2000/100000) \rceil = 134$$

$B_{CentMilResp}$	10000
B_{C1}	1
B_{C2}	100
B_{C3}	100
B_{C4}	10
B_{C5}	134

Example of materialized view selection (II)

C1/Q1 - SELECT cand, MAX(val) FROM CentMilResp GROUP BY cand;
C2/Q2 - SELECT cand, edat, AVG(val), MAX(val), MIN(val) FROM CentMilResp GROUP BY cand, edat;
C3/Q3 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;
C4/Q4 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY pobl;
C5 - SELECT cand, pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;



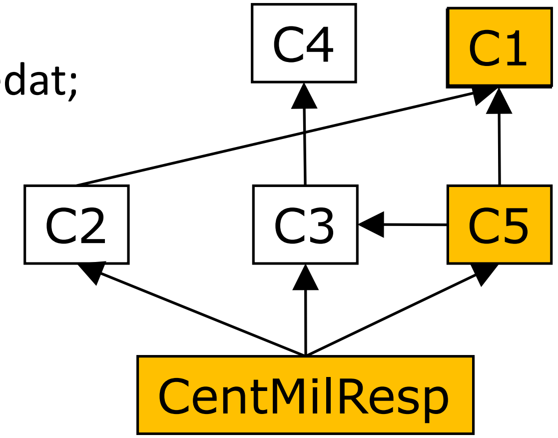
Cost if there is no materialized view:

- Time: 10.000 sec/query
- Space: 10.000 blocks

	Q1 (35%)	Q2 (20%)	Q3 (20%)	Q4 (25%)	Avg
C1	1	10000	10000	10000	6500,4
C2	100	100	10000	10000	4555,0
C3	10000	10000	100	100	5545,0
C4	10000	10000	10000	10	7502,5
C5	134	10000	134	134	2107,2

$B_{CentMilResp}$	10000
B_{C1}	1
B_{C2}	100
B_{C3}	100
B_{C4}	10
B_{C5}	134

Example of materialized view selection (III)



C1/Q1 - SELECT cand, MAX(val) FROM CentMilResp GROUP BY cand;
C2/Q2 - SELECT cand, edat, AVG(val), MAX(val), MIN(val) FROM CentMilResp GROUP BY cand, edat;
C3/Q3 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;
C4/Q4 - SELECT pobl, MAX(val) FROM CentMilResp GROUP BY pobl;
C5 - SELECT cand, pobl, MAX(val) FROM CentMilResp GROUP BY cand, pobl;

Cost if C5 is materialized:
- Time: 2.107,2 sec/query
- Space: 10.134 blocks

	Q1 (35%)	Q2 (20%)	Q3 (20%)	Q4 (25%)	Avg
C1	1	10000	134	134	2060,7
~~C2~~	~~100~~	~~100~~	~~134~~	~~134~~	~~115,3~~
~~C3~~	~~134~~	~~10000~~	~~100~~	~~100~~	~~2091,9~~
~~C4~~	~~134~~	~~10000~~	~~134~~	~~10~~	~~2076,2~~

$B_{CentMilResp}$	10000
B_{C1}	1
B_{C2}	100
B_{C3}	100
B_{C4}	10
B_{C5}	134

Cost if C1 and C5 are materialized:
- Time: 2.060,7 sec/query
- Space: 10.135 blocks

Closing

Summary

- ANSI/SPARC architecture
- Difficulties/Features when dealing with views
  - View expansion
  - Update through views
  - View updating
    - Assertions
  - Answering queries using views
    - Materialized view selection

Bibliography

- J. Sistac et al. *Disseny de bases de dades*. Editorial UOC, 2002. Col·lecció Manuals, número 43
- G. Gardarin and P. Valduriez. *Relational databases and knowledge bases*. Addison Wesley Publishing Company, 1989
- T. Teorey et al. *Database modeling and design*, 4th edition. Morgan Kaufmann Publishers, 2006
- M. Golfarelli and S. Rizzi. *Data Warehouse Design*. McGrau-Hill, 2009