FIB - Disseny de Bases de Dades

Correctness and Materialized Views selection

Correctness

Knowledge Objectives

- Define four logic properties of integrity constraints (i.e. schema satisfiability, liveliness, constraint redundancy and state reachability)
- Exemplify the three necessary conditions for summarizability

Application Objectives

- Find and eventually fix the problems related to the five logical properties of integrity constraints in a given relational schema (with at most 6 tables and views)

Problems in the constraints

- Contradictory constraints generate empty tables/views
    - May even result in empty databases
      ```
      CHECK (a<10 AND a>20)
      ```
- Redundant constraints slow down DBMS performance
  ```
  CHECK (a>10)
  CHECK (a>20)
  ```

Logic properties of constraints

- **Schema-satisfiability**:
    - A **schema** is satisfiable if there is at least **one consistent DB state** containing tuples (i.e. each and every constraint is fulfilled)
- **Liveliness**:
    - A **table/view** is lively if there is at least **one consistent DB state**, so that the table/view contains tuples
- **State-reachability**:
    - A given **set of tuples** is reachable if there is at least **one consistent DB state** containing those tuples (and maybe others)
- Redundancy:
    - A **constraint** is redundant if it is a logic consequence of other constraints

Example of Schema-satisfiability

```
CREATE TABLE employees
(
  id CHAR(9)       PRIMARY KEY,
```

```
  dpt VARCHAR(4)  NOT NULL REFERENCES departments (ID)
);
CREATE TABLE departments
(
  id VARCHAR(4)     PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  basicSalary INT   NOT NULL ,
  CONSTRAINT ckMinSalary CHECK (basicSalary > 2000),
  CONSTRAINT ckMaxSalary CHECK (basicSalary < 1000)
);
```

> `dpt VARCHAR(4) NOT NULL REFERENCES departments (ID)` REFERENCES departments (ID) afegit
>
> `basicSalary INT NOT NULL ,` not null afegit

Example of Liveliness

```
CREATE TABLE departments
(
  id VARCHAR(4)     PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  basicSalary INT   NOT NULL,
  CONSTRAINT ckMinSalary CHECK (basicSalary > 2000)
);
CREATE TABLE employees
(
  id CHAR(9)      PRIMARY KEY,
  dpt VARCHAR(4)  NOT NULL REFERENCES departments (id)
);
CREATE VIEW unassigned AS
(
  SELECT  *
  FROM    employees e
  WHERE   NOT EXISTS
  (
    SELECT  *
    FROM    departments d
    WHERE   d.id = e.dpt
  )
);
```

> `dpt VARCHAR(4) NOT NULL REFERENCES departments (id))` not null afegit

Example of Redundancy

```
CREATE TABLE departments
(
  id VARCHAR(4)     PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  basicSalary INT   NOT NULL,
  CONSTRAINT ckMinSalary  CHECK (basicSalary > 2000),
  CONSTRAINT ckDeptName   CHECK (id <> 'CS')
```

```
);
CREATE TABLE employees
(
  id CHAR(9)      PRIMARY KEY,
  dpt VARCHAR(4)  NOT NULL REFERENCES departments (ID),
  CONSTRAINT ckEmpName CHECK (dpt <> 'CS')
);
```

> `CONSTRAINT ckEmpName CHECK (dpt <> 'CS')` redundant

Example of State-reachability

```
CREATE TABLE departments
(
  id VARCHAR(4)     PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  basicSalary INT   NOT NULL,
  CONSTRAINT ckMinSalary CHECK (basicSalary>2000)
);
CREATE TABLE employees
(
  id CHAR(9)      PRIMARY KEY,
  dpt VARCHAR(4)  NOT NULL REFERENCES departments (ID)
);
```

```
Employees (id, dpt);
           1   CS
           2   MK
```

```
Departaments (id, name,     basicSalary);
              CS  Compu...  10000
              MK  Marke...  2001
```

> `MK Marke... 2001` afegit

Aggregation problems

(I)

- Number of students per department and year, assuming the students follow a two-year program

	1994	1995	1996	All
Informatics	15	17	13	28
Statistics	10	15	11	21
All	25	32	24	49

disjunció

- Number of students per department and year, assuming the students follow a two-year program where there are inter-department courses

	1994	1995	1996	All	
Informatics	15	17	13	28	
Statistics	10	15	11	21	
All	23	30	24	47	

(II)

- Number of car accidents per province chief town and year

	1994	1995	1996	All
Barcelona	5	6	3	14
Tarragona	1	0	1	2
Lleida	0	2	1	3
Girona	3	5	6	14
Catalunya	20	23	22	65

completesa

(III)

	interval Cumulative	**instant** State	**e.g. preu** Value per unit
min	No problem	No problem	No problem
max	No problem	No problem	No problem
sum	No problem	Non-temporal **nombre d'habitants**	Never **no té sentit**
avg	No problem	No problem	No problem

compatibilitat

Summary

- Logic properties of constraints
  - Schema satisfiability
  - Lifeliness
  - Redundancy
  - State-reachability
- Aggregation problems
  - Summarizability necessary conditions

Materialized Views selection

Understanding Objectives

- Select a set of views to be materialized in the following scenarios:

- Disk space is limited and the system is read-only
    - Only the given user queries can be materialized
    - Any query can be materialized
- Disk space is not limited and the system is read-write

Application Objectives

- Given a set of source tables (no more than 6) and some views over them (no more than 3), justify if a specific query over the tables can be rewritten in terms of the (materialized) views.

Answering queries using views

- Principles:
    - Query predicate ⇒ View predicate
    - Query tuples $\subsetneq$ View tuples
    - Aggregation level must be higher or equal in the query
        - Functional dependencies can be used to check it
    - Query Aggregate must be computable from the view one.
- The problem of deciding whether it is possible to rewrite a query in terms of existing views or not is computationally complex
    - DBMSs restrict the search space to common cases by using rules

Example of query rewriting (I)

> **HAVE**

```
CREATE MATERIALIZED VIEW euroSales ENABLE QUERY REWRITE AS
(
  SELECT
  (
    d1.city,
    d2.product,
    SUM(f.euros) AS sumEuros,
    COUNT(*) AS salesCounter
  )
  FROM      sales f, stores d1, products d2
  WHERE     f.storeId = d1.Id AND f.productId = d2.Id
  GROUP BY  d1.city, d2.product
);
```

> **WANT**

```
SELECT    d1.city, AVG(f.euros) AS avgSales
FROM      sales f, stores d1
WHERE     f.storeId = d1.Id
GROUP BY  d1.city;
```

> **GET**

```
SELECT    city, SUM(sumEuros)/SUM(salesCounter) AS avgSales
FROM      euroSales
GROUP BY  city;
```

Example of query rewriting (II)

> **HAVE**

```
CREATE MATERIALIZED VIEW euroSales ENABLE QUERY REWRITE AS
(
  SELECT
  (
    d1.city,
    d2.product,
    SUM(f.euros) AS sumEuros,
    COUNT(*) AS salesCounter
  )
  FROM      sales f, stores d1, products d2
  WHERE     f.storeId = d1.Id AND f.productId = d2.Id
  GROUP BY  d1.city, d2.product
);
```

> **WANT**

```
SELECT    d1.city, p2.productId, c.id, SUM(f.euros) AS sales
FROM      sales f, stores d1, product p2, customer c
WHERE
(
  f.storeId = d1.Id   AND
  f.productId = p2.id AND
  f.customerId = c.id
)
GROUP BY  d1.city, p2.productId, c.id
;
```

> **GET**

```
??????????????????
```

no es pot —> group by de vista restrictiu

Example of query rewriting (III)

> **HAVE**

```
CREATE MATERIALIZED VIEW euroSales ENABLE QUERY REWRITE AS
(
  SELECT
  (
    d1.city,
    d2.product,
    SUM(f.euros) AS sumEuros,
    COUNT(*) AS salesCounter
```

```
  )
  FROM      sales f, stores d1, products d2
  WHERE     f.storeId = d1.Id AND f.productId = d2.Id
  GROUP BY  d1.city, d2.product
);
```

WANT

```
SELECT    d1.city, MAX(f.euros) As mx
FROM      sales f, stores d1, product p2
WHERE
(
  f.storeId = d1.Id   AND
  f.productId = p2.Id AND
  p2.id = "1"
)
GROUP BY  d1.city;
```

GET

```
??????????????????
```

no es pot -> max, només tinc sum

To improve query performance ...

- Build access structures (Indexes)
- Pre-calculate as much as possible
  - Redundant tables
    - Less attributes
    - Less tuples
      - Only those fulfilling the query predicate
      - Only one per combination of values of attributes in the `GROUP BY`
        - The sparser the basic tuples, (proportionally) the more space aggregates will use (e.g. Twelve days per year may generate twelve months per year)
    - Less space than the table
    - Less I/O to be accessed

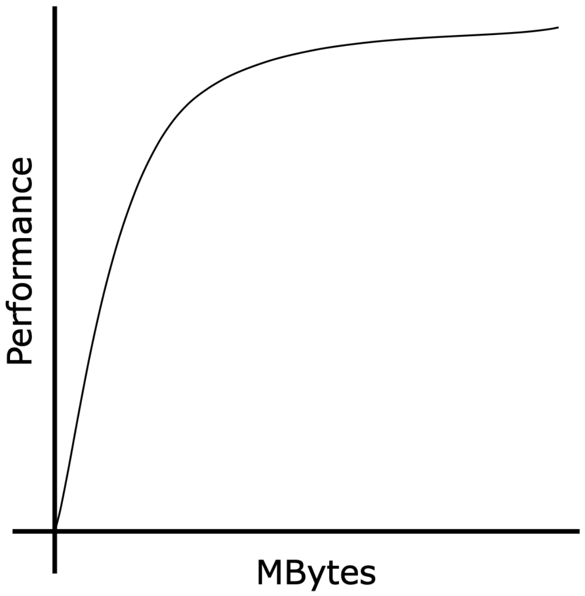
no es pot —> group by de vista restrictiu

Problems in pre-calculating

- Cost
  - Space
  - Time
    - Query vs Modification frequency
- Consistency and rewriting control
  - Using materialized views
  - Using triggers
    - Advantages
      - Flexible
      - Allows rewriting of any query
      - Maybe efficient

- Disadvantages
  - Difficulties management (table administration and load)
  - Ad-hoc rewriting must be implemented for each query
    - Users are bound to our rewriting tools

Materialization trade-off



> Is our Update Window enough?

Combinatorial aggregation explosion

- Choosing the best combination of views to be materialized is NP-complex
  - A fact table with m dimension tables with n aggregation levels (including the “atomic” and “All” levels) for each one, would generate nm possible materialized views

Candidate views to be materialized

- Given a workload $W=\{q_1, q_2, q_3, ...\}$, and identifying queries by their `GROUP BY` clause, candidate views v_i are those that:
  - $GB(v_i) = GB(q_j)$
  - $GB(v_i) = \bigcup\limits_{q_j \in Q} GB(q_j)$ where $Q \subseteq W$
- Provided predicates allow rewriting
- Adding (**aggregations** and/or **dimensions**) to the `SELECT` if needed

Algorithm to choose among candidates

- Greedy algorithm (guarantees 63% minimum improvement):

```
do
(
  Consider those candidate views that fit
  in the available space and time

  Sort views based on the performance
```

```
  improvement they induce

  Materialize first view in the list,
  if it improves performance
)
while (performance improved and available space and time)
```

- Modify the set of materialized views as user needs evolve

Example of materialized view selection

- Taula CentMilResp(ref, pobl, edat, cand, val)
- `D = 1seg; C=0`
- `BCentMilResp=10000; | CentMilResp | =100000`
- `Ndist(pobl)= 200; Ndist(edat)=100; Ndist(cand)=10`
- La informació de control ocupa el mateix que un atribut
- Tots els atributs ocupen el mateix
- Freqüència de les consultes:
  - 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`
- Tenim `10140` blocs de disc

> 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

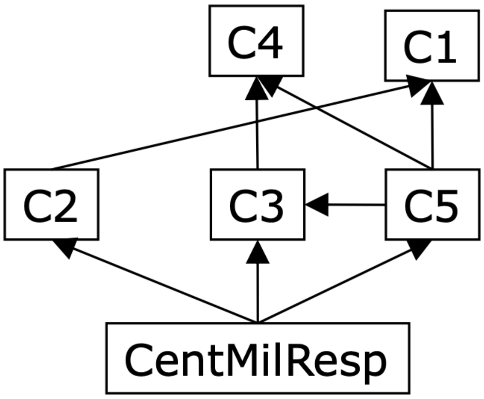
```
SELECT    pobl, MAX(val)
FROM      CentMilResp
GROUP BY  cand, pobl
```

> C4/Q4

```
SELECT    pobl, MAX(val)
FROM      CentMilResp
GROUP BY  pobl
```

> C5

```
SELECT    cand, pobl, MAX(val)
FROM      CentMilResp
GROUP BY  cand, pobl
```



FILES AGREGACIÓ

mín(Nfiles_0, Ndist(a_1) * ... * Ndist(a_n))mı´n(N f iles0, N dist(a1) ∗ ... ∗ N dist(an))

- C2: mín(100000, 10 * 100) = 1000mı´n(100000, 10 ∗ 100) = 1000
- C3: mín(100000, 10 * 200) = 2000mı´n(100000, 10 ∗ 200) = 2000

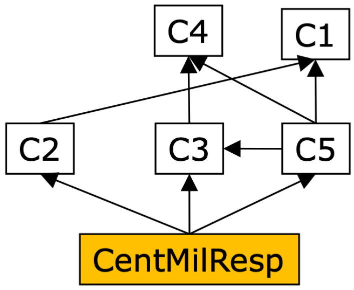
ESPAI AGREGACIÓ

E_0 * \frac{Natr}{Natr_0} * \frac{Nfiles}{Nfiles_0}E0 ∗ N atr0N atr ∗ N files0N files

- C2: 10000 * \frac{6}{6} * \frac{1000}{100000} = 10010000 ∗ 66 ∗ 1000001000 = 100
- C3: 10000 * \frac{3}{6} * \frac{2000}{100000} = 10010000 ∗ 63 ∗ 1000002000 = 100

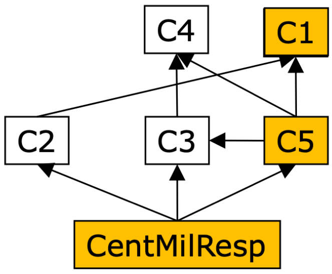
C1	1
C2	100
C3	100
C4	10
C5	134

Cost if there is no materialized view:
- Time: 10000 sec/query
- Space: 10000 blocks



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

Cost if C5 is materialized:
- Time: 2107,2 sec/query
- Space: 10134 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~~

Cost if C1 and C5 are materialized:
- Time: 2060,7 sec/query
- Space: 10135 blocks

Summary

- Pre-aggregation
- Materialized view selection

Materialized view example

STORES	
Id	city
1	Mataró
2	Mataró

PRODUCTS	
Id	product
1	rubber
2	pen

Dimension 1

Fact

Dimension 2

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

Measures

EUROSALES			
City	Product	sumEuros	salesCounter
Mataró	rubber	25	2
Mataró	pen	20	1

Aggregations

Materialized view