Physical design (views)

Knowledge Objectives

1. Enumerate the main basic tasks in the physical design of a DB
2. Enumerate the main criteria we should use on making a decision about the physical design of a DB
3. Enumerate the main difficulties we would find in the physical design
4. Explain the differences between the three levels in the ANSI/SPARC architecture, paying special attention to physical and logical independency
5. Explain the two differences between a view and a table
6. Explain the difference between a VIEW and a MATERIALIZED VIEW
7. Enumerate and distinguish the four problems associated to views
8. According to the standard, name the two constraints a view must fulfill to be updatable
9. Discuss the benefits of a complete and an incremental view update
10. Enumerate when and how a materialized view can be refreshed

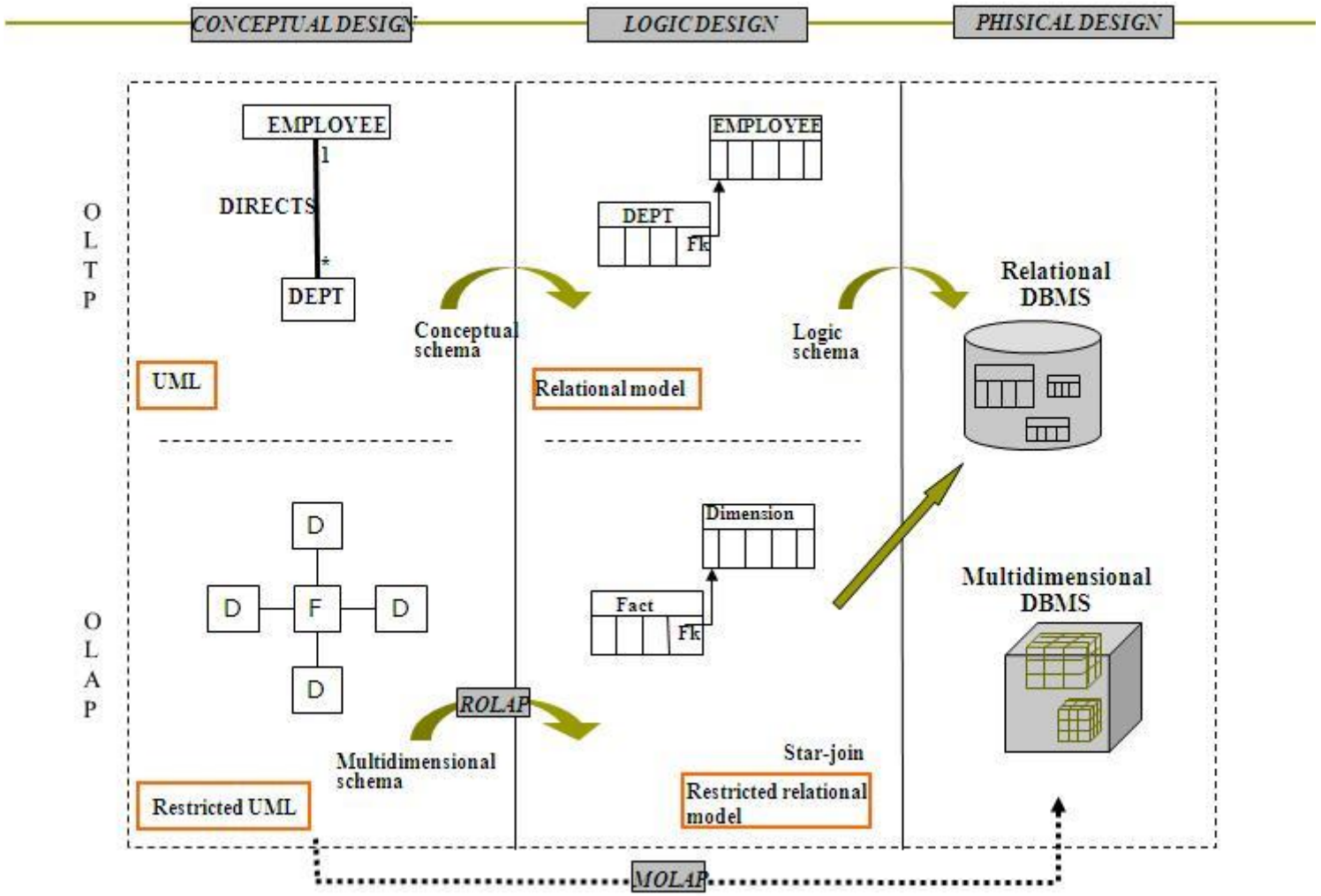
Application Objectives

1. Given a set of source tables (no more than 6) and some views over them (no more than 3), justify if
   a) A given view is updatable
   b) A given (materialized) view can be incrementally updated

"In theory, there is no difference between theory and practice. In practice, there is."

Jan L. A. Van de Snepscheut

Transactional vs Decisional



Basic tasks on physical design

- Adapting the logic schema to the DBMS
  - Data types
  - Views
  - Integrity constraints
  - Deadlocks
- Revisiting the relational schema
  - Partitioning
- Choose data structures
  - Indexing
- Performance test
  - Concurrency control
  - Recovery
  - Files
  - System parameters

Criteria for physical design

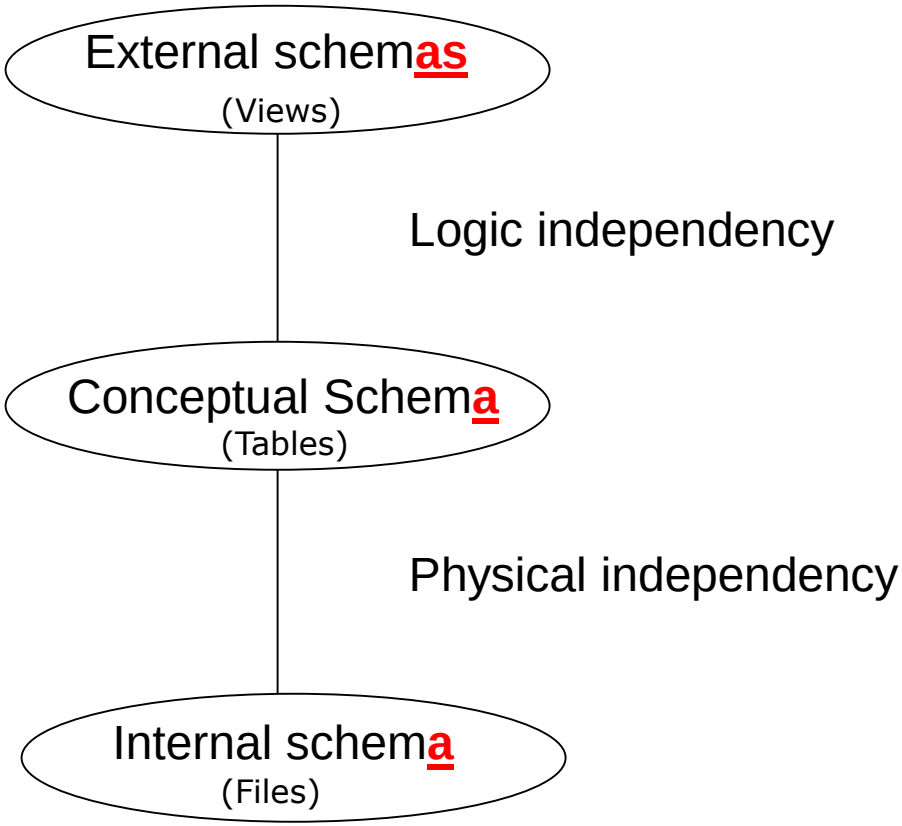
- Performance improvement
  - Memory and disk space
  - CPU time
  - Disk access time
  - Contention
  - Auxiliary processes costs
- Scalability
- Availability
- Integrity
- Administration simplicity

Difficulties in physical design

- Users
- Opposing criteria
- Limited resources
- Imperfections in the DBMS (query optimizer)
- Communications (network)

VIEWS
(AND MATERIALIZED VIEWS)

ANSI/SPARC architecture



Alternatives to implement a relation

- From the structures / data point of view
  - Tables
    - Data in disk (Materialized)
  - Non-materialized views
    - Definition in catalog (SQL statement)
      - Re-executed with every query
  - Materialized views
    - Data in disk (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 materialized result of the query
      - Reduced to a synchronization problem

Materialized views in Oracle 11g

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

Materialized views in Oracle 11g

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

Materialized views in Oracle 11g

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

Materialized views in Oracle 11g

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

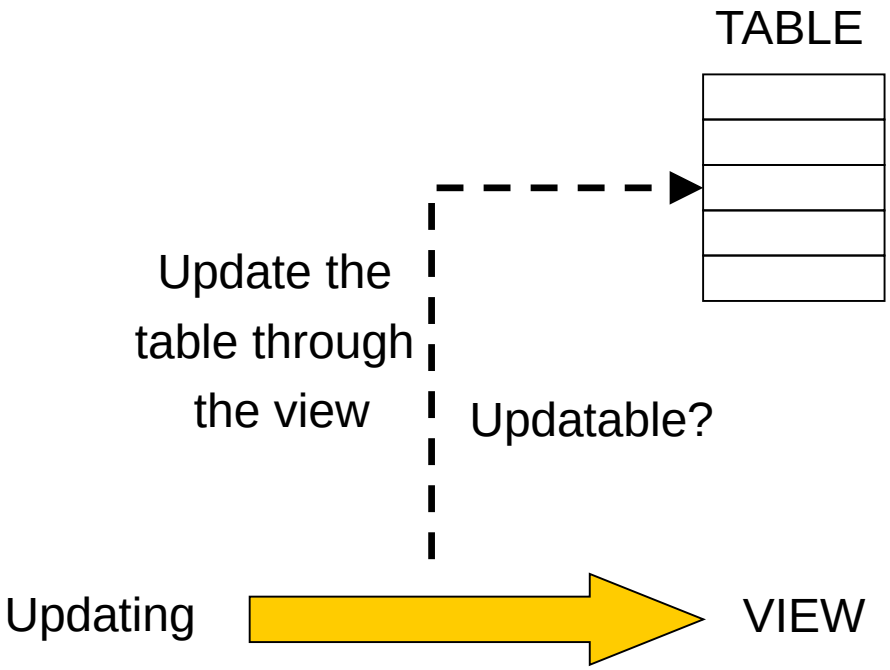
Problems associated to views

- Non materialized views
  a) View expansion
     - Transform the query over the views into a query over the source tables
- Materialized views
  b) 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 to the sources by means of a translation process

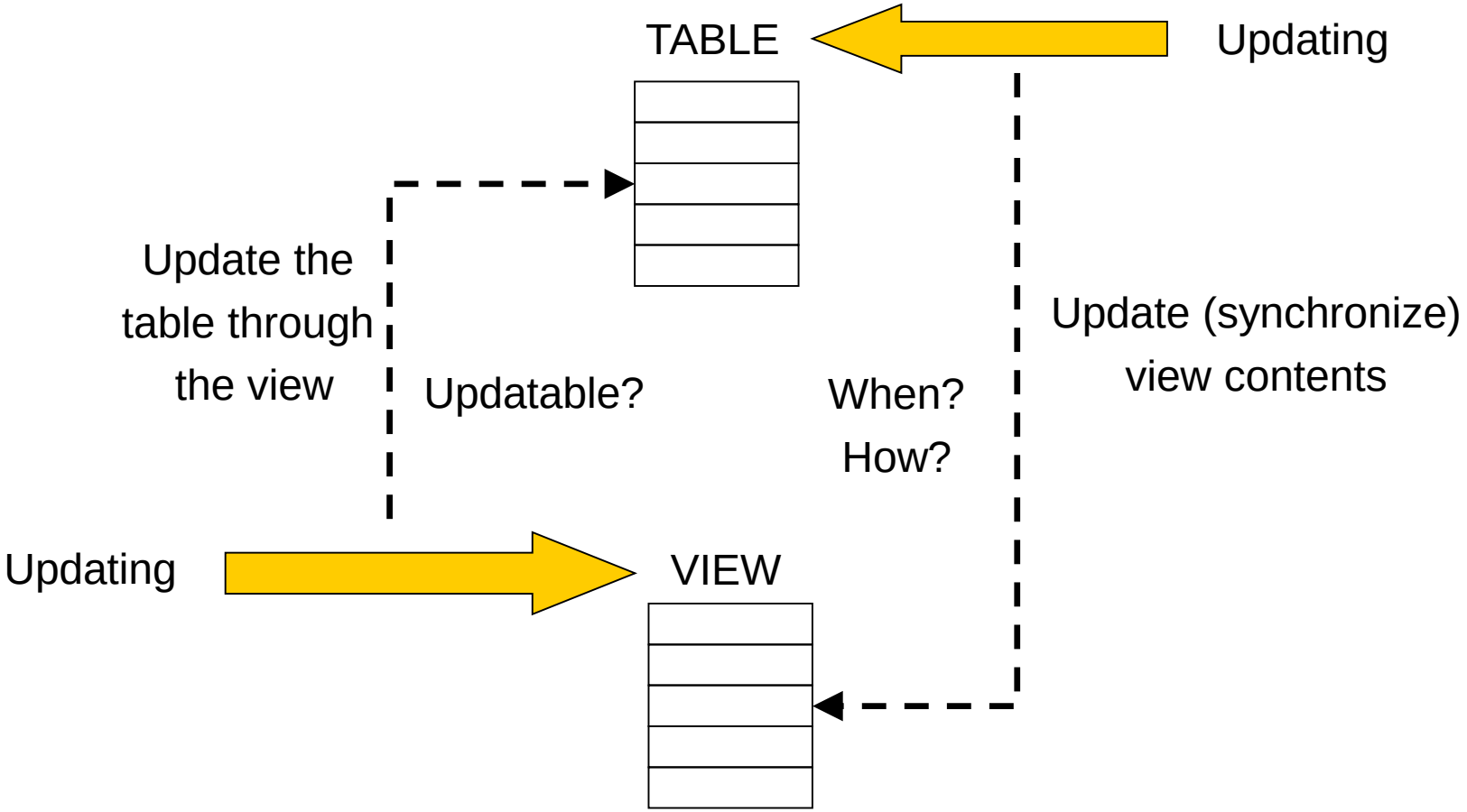
Problems associated to views

- Materialized views
  - b) 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 to the sources by means of a translation process

Views and modifications



Views and modifications



Update through views

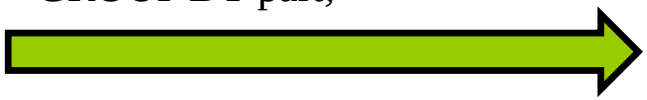
- Views, in general, are non-updatable
  - Only when the update can be unambiguously translated over the relational table
- Updatable views according to the standard:
  - Relational selection on a single table or updatable view
    - Subqueries, joins or aggregate functions are not considered
  - It may also contain a relational projection iff the following attributes are projected
    - The primary key
    - Not-null attributes

Update through views with aggregates

part_supplied (<u>supp, part</u>, 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 (<u>part</u>, qt)

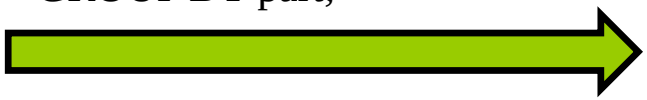
p1	300
p2	200

Update through views with aggregates

part_supplied (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)

p1	300
p2	200

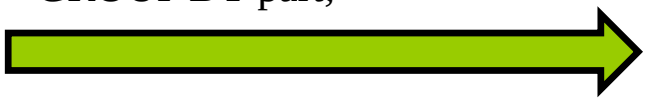
```
DELETE FROM total_qt WHERE part='p1';
```

Update through views with aggregates

part_supplied (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)

p1	300
p2	200

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

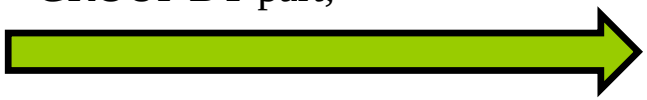
```
DELETE FROM part_supplied
WHERE part='p1';
```

Update through views with aggregates

part_supplied (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)

p1	300
p2	200

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

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

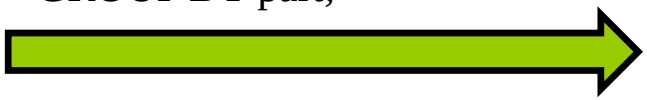
```
UPDATE total_qt SET qt=301 WHERE part='p1';
```

Update through views with aggregates

part_supplied (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)

p1	300
p2	200

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

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

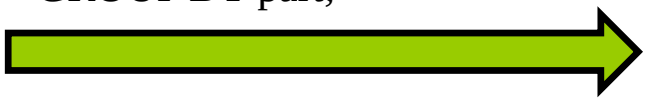
```
UPDATE total_qt SET qt = qt - 100 WHERE part='p1';
```

Update through views with aggregates

part_supplied (<u>supp, part</u>, 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 (<u>part</u>, qt)

p1	300
p2	200

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

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

```
UPDATE total_qt SET qt=qt+100 WHERE part='p1';
```

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

Update through views with aggregates

part_supplied (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)

p1	300
p2	200

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

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

```
UPDATE total_qt SET ... WHERE part='p1';
INSERT INTO total_qt VALUES ('p2',400);
```

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)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

sup-part_sup (name, supp, part, qt)
Joan 100 p1 10
Joan 100 p2 20

Update through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (name, supp, part, qt)
Joan 100 p1 10
Joan 100 p2 20

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

Update through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (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 through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (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);
```

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

Update through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (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);

DELETE FROM s[illegible] AND part='p1';

Update through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (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);

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

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

Update through views with joins

supplier (nsupp, name)
100 Joan

part_supplied (supp, part, qt)
100 p1 10
100 p2 20

```
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;
```

sup-part_sup (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);
```

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

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

View updating

- Complete update
  - All instances are regenerated
    - Clearly inefficient?
    - Always possible
- Incremental update (called “fast” in Oracle)
  - Only instances that changed are regenerated
    - Much more efficient?
    - Not always possible

Fast materialized views (Oracle 11g)

- On commit refresh is only possible for views allowing incremental (fast) updates
- 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
  - Basic queries (without groupings nor joins)
  - Join queries
  - Grouping queries
- Oracle explanation for *fast* update
  - BEGIN DBMS_MVIEW.EXPLAIN_MVIEW('materialized_view_name'); END;
  - The MV_CAPABILITIES_TABLE table is needed to store the explanations produced

Assertions

- They are predicates expressing a constraint. May involve several tuples/tables
- Not yet provided by most RDBMS
- Simulation with materialized views (Oracle)

Empty M.V.
- Define a materialized view with the negation of the assertion
- Define a check, which should never be satisfied

M.V. grows
- Define a materialized view with a check equivalent to the assertion
- Most times, an ON COMMIT refresh will be required
  - ON DEMAND or NEXT may be enough

Example of simulating assertions

- Assertion:

```
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));
```

- Materialized view simulating the assertion:

```
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 simulating assertions

- Assertion:

```
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));
```

- Materialized view simulating the assertion:

```
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);
```

Summary

- Design tasks and criteria
- ANSI/SPARC architecture
- Problems when dealing with views
  - [View expansion]
  - Answering queries using views
  - Update through views
  - View updating
    - Assertions

Bibliography

- Jaume Sistac, et al. *Disseny de bases de dades*. Editorial UOC, 2002. Col·lecció Manuals, número 43
- George Gardarin and Patrick Valduriez. Relational databases and knowledge bases. Addison Wesley Publishing Company, 1989