FIB - Disseny de Bases de Dades

Physical Design (other issues)

Knowledge Objectives

1. Explain how candidate keys are implemented by a RDBMS
2. Explain the difference between immediate and deferred constraint checking
3. Enumerate two alternatives that allow to disable integrity constraints

Understanding Objectives

1. Explain why it is better to use the surrogate mechanisms provided by the DBMS than implementing it ad-hoc

Application Objectives

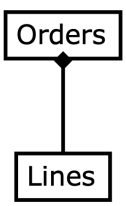
1. In an FK-deadlock situation, choose the best solution, given the data statistics and the involved operations
2. Given a UML class diagram, identify which constraints can be enforced by just the `CREATE TABLE` sentence

Candidate keys

- Primary
  - May not be available in our DBMS (rare)
  - Physically, it generates a B-tree index
- Alternatives
  - They are not part of standard SQL
  - Can be implemented by `NOT NULL + UNIQUE`

Surrogates

- Introduced by E. F. Codd in RM/T
- Substitutes the external key if:
  - There is no such external key
  - Attributes in the external key change often
  - The external demands too much space

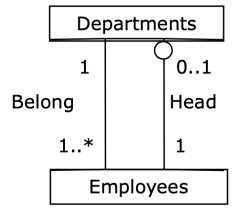


User surrogates in Oracle 11g

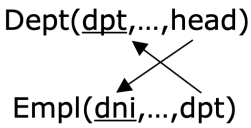
```
CREATE SEQUENCE <seqName>
  INCREMENT BY <int>
  START WITH <int>
  ...;

SELECT <seqName>.CURRVAL FROM DUAL;

INSERT INTO <table> VALUES
  (<seqName>.NEXTVAL, ...);

DROP SEQUENCE <seqName>;
```

- Multiple sequences can be used in the same table
- A sequence can be used in different tables

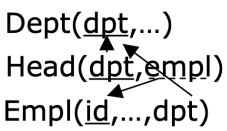
Deadlock in the definition of FK



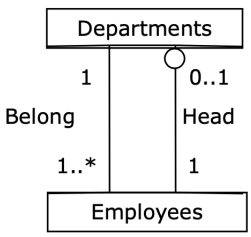
A. Modify the tables
   1) Create "Dept" without FK
   2) Create "Empl" with FK
   3) Alter "Dept" adding the FK



B. Create three tables
   1) Create "Dept" (it doesn't have FK)
   2) Create "Empl" with its FK
   3) Create "Head" with two FK



Deadlock in the load of FK



Dept(	dpt,…,	head)
	CS	1
Empl(	dni,…,	dpt)
	1	CS

A. Deactivate FK
   1) Alter "Dept" to drop FK
   2) Insert the department
   3) Insert all employees
   4) Alter "Dept" to add FK

```
SET CONSTRAINT <name>
    [ENABLE|DISABLE];
```

B. Defer the FK checking

```
SET CONSTRAINT <name>
    [IMMEDIATE|DEFERRED];
```

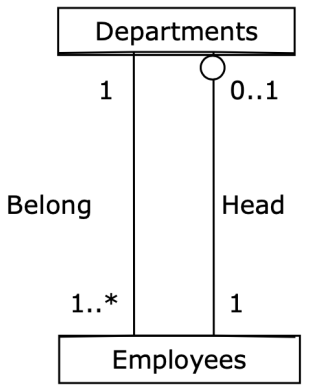
Implementing constraints

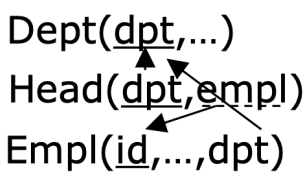
1. In the `CREATE TABLE` sentence
   - Usually available, efficient, automatic and internal
2. Assertions
   - Efficient, automatic and internal
3. Persistent Stored Modules
   1. Triggers: Automatic and internal
   2. Procedures/functions: Internal
4. Call Level Interface (Eg: ODBC, JDBC)
   - Always available

Example of constraint implementation

Constraints reflected in the conceptual schema

- An employee is head of zero or one department
  - Attribute “empl” in table “Head” is UNIQUE
- Every department has exactly one head
  1. Attribute “empl” in table “Head” is FK and NOT NULL
  2. An assertion should be defined to check that each and every department has a head (we may also implement it with two tables)
- An employee belongs to exactly one department
  - Attribute “dpt” in table employee has been defined as FK and NOT NULL
- Each and every department has at least one employee
  - An assertion should be defined





Summary

- Surrogates
- Deadlock
  - In the definition of FK
  - In the load of FK

- Integrity constraints

Peixos

Suposa que volem mantenir informació dels diferents tipus de peixos d'aquari. Concretament, volem mantenir el nom tècnic de l'espècie (en llatí), la família a la qual pertanyen, el color que presenten, el mecanisme de reproducció i la seva raresa. Tingues en compte que dins d'una família de peixos n'hi ha que són més o menys rars, totes les espècies de la mateixa família tenen el mateix mecanisme de reproducció, una espècie pertany només a una família i que tots els peixos d'una espècie tenen el mateix color. També sabem que no pot haver dues espècies dins de la mateixa família que tinguin el mateix color.

```
Peixos(especie, familia, color, reproduccio, raresa)
```

- Família -> Reproducció
- Espècie -> Família
- Espècie -> Color
- Família, Color -> Espècie
- Espècie -> Raresa

Physical Design (views)

Knowledge Objectives

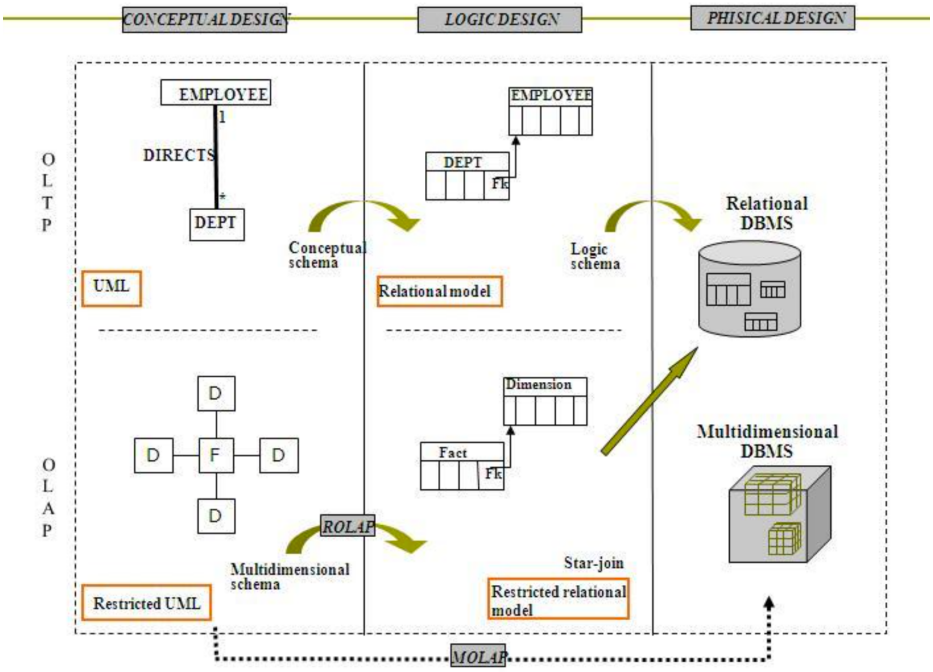
1. Enumerate the main basic tasks in the physical design of a DB
2. Enumerate the main criteria we should use on making a decision about the physical design of a DB
3. Enumerate the main difficulties we would find in the physical design
4. Explain the differences between the three levels in the ANSI/SPARC 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

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

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

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

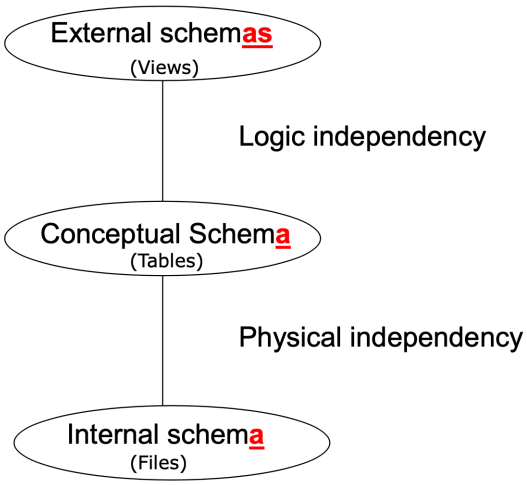
- Performance improvement
  - Memory and disk space
  - CPU time
  - Disk access time
  - Contention
  - Auxiliary processes costs
- Scalability
- Availability
- Integrity
- Administration simplicity

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

$$V = F(R_1, R_2, ..., R_n)$$

$$Q(V) \rightarrow Q'(R_1, R_2, ..., R_n)$$

- 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>;
```

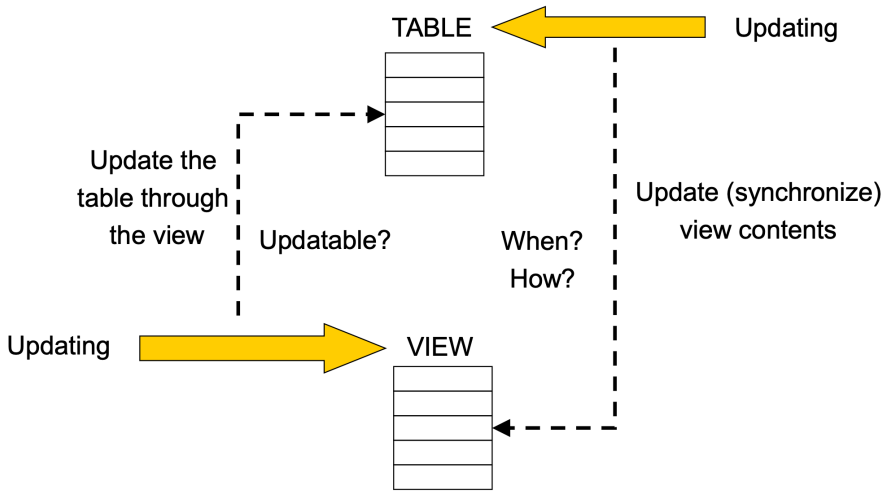
Problems associated to views

- Non materialized views
  - View expansion
    - Transform the query over the views into a query over the source tables
- Materialized views
  - Answering queries using views
    - Transform an arbitrary query over the tables into a query over the available views
  - View updating
    - Changes in the sources are, potentially, propagated to the view
- Both
  - Update through views
    - Propagate the changes to the sources by means of a translation process

> ATFER (?):

- ~~Non materialized views~~
  - ~~View expansion~~
    - ~~Transform the query over the views into a query over the source tables~~
- Materialized views
  - Answering queries using views
    - Transform an arbitrary query over the tables into a query over the available views
  - View updating
    - Changes in the sources are, potentially, propagated to the view
- Both
  - Update through views
    - Propagate the changes to the sources by means of a translation process

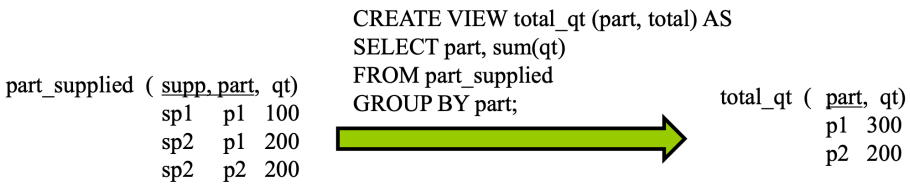
Views and modifications



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



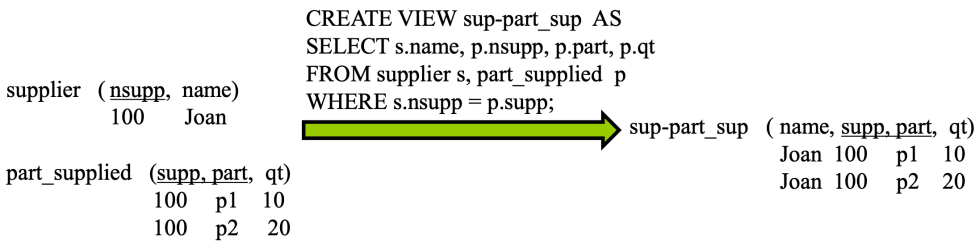
DELETE FROM total_qt WHERE part='p1';

DELETE FROM part_supplied
WHERE part='p1';

~~UPDATE total_qt SET qt=301 WHERE part="p1";~~

~~INSERT INTO total_qt VALUES ("p3",400);~~

Update through views with joins



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
    (
      o.#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)
;
```

```
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