SQL language: other definitions

Management of views

Management of views

- Introduction
- Creation and management of views in SQL
- Updating views
- Check option
- Privacy Management

Management of views

Introduction

The concept of view

- A view is a *"virtual"* table
  - the content (tuples) is defined by means of an SQL query on the database
    - the content of the view depends on the content of the other tables present in the database
  - the content is *not* memorized physically in the database
    - it is recalculated every time the view is used by executing the query that defines it
- A view is an object of the database
  - it can be used in queries as if it were a table

DB product suppliers

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Example n.1

- Definition of the view *small suppliers*
  - the suppliers that have fewer than 3 employees are considered "small suppliers"
- The view "small suppliers"
  - contains the code, name, number of employees and city of the suppliers that have fewer than 3 employees.

Example n.1: definition of the view

- Definition of the view "small suppliers"
  - contains the code, name, number of employees and city of suppliers with fewer than 3 employees

```
SELECT SId, SName, #Employees, City
FROM   S
WHERE #Employees <3
```

Query associated with the view

Example n.1: definition of the view

- Definition of the view "small suppliers"
  - it contains the code, name, number of employees and city of suppliers with fewer than 3 employees

Name of the views

```
CREATE VIEW SMALL_SUPPLIERS AS
SELECT SId, SName, #Employees, City
FROM   S
WHERE #Employees<3;
```

Example n.1: query

- View the code, name, employee number and city of "small suppliers" in London
- The query can be answered without using views

```
SELECT *
FROM   S
WHERE #Employees<3 AND
          City='London';
```

Example n.1: query

- View the code, name, employee number and city city of “small suppliers” in London
- The query can be answered using the view defined previously

```
SELECT *
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

- The view SMALL_SUPPLIERS is used like a table

Rewriting the queries

- If the query refers to a view, it has to be reformulated by the DBMS before execution
- The reformulation is carried out automatically
  - the references to the view are substituted by its definition

Example n.1: reformulating the query

- View the code, name, employee number and city city of “small suppliers” in London

```
SELECT *
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

Example n.1: reformulating the query

- View the code, name, employee number and city city of "small suppliers" in London

```
SELECT SId, SName, City, #Employees
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

- Reformulate the SELECT clause
  - the attributes present in the definition of the view are made explicit

Example n.1: reformulating the query

- View the code, name, employee number and city city of "small suppliers" in London

```
SELECT SId, SName, City, #Employees
FROM   S
WHERE  #Employees<3  AND
       City='Torino';
```

- Introduction of the definition of the view
  - In the clause FROM
  - In the clause WHERE

Example n.2

- Definition of the view *number of suppliers per product*
  - The view contains the product code and the number of different suppliers providing it

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - The view contains the product code and the number of different suppliers providing it

```
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId
```


Query associated with the view

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - the view contains the product code and the number of different suppliers providing it

Name of the views

```
CREATE VIEW NUMSUPPLIERS_PER_PRODUCT
(PId, #Suppliers) AS
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId;
```

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - the view contains the product code and the number of different suppliers providing it

```
CREATE VIEW NUMSUPPLIERS_PER_PRODUCT
(PId, #Suppliers) AS
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId;
```

Attributes of the view

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Without using views

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Without using views

```
SELECT PId
FROM   SP
GROUP BY PId
HAVING COUNT(*)=(SELECT MAX(#Suppliers)
                 FROM (SELECT COUNT(*) AS #Suppliers
                       FROM   SP
                       GROUP BY PId));
```

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Using the view NUMSUPPLIERS_PER_PRODUCT

```
SELECT PId
FROM NUMSUPPLIERS_PER_PRODUCT
WHERE #Suppliers=(SELECT MAX(#Suppliers)
                  FROM
                       NUMSUPPLIERS_PER_PRODUCT);
```

Considerations on the examples

- The use of views simplifies the formulation of the queries
- The view SMALL_SUPPLIERS conceals the definition of the concept of "small suppliers"
  - it is possible to redefine the concept of "small suppliers" just by changing the definition of the view
    - it is not necessary to modify the queries that use it
- The view NUMSUPPLIERS_PER_PRODUCT enables us to avoid using the table function

Advantages of views

- Simplification of the queries
  - very complex expressions can be defined in a simpler way by using views
    - by breaking down a complex query into subqueries associated with the views
      - useful in the presence of repeated (complex) subqueries

Advantages of views

- Extension of the SQL language's power of expression
  - in the absence of a table function, some typologies of queries can only be defined by using views
    - as an alternative to using the procedural code

Advantages of views

- Security management
  - it is possible to introduce different privacy protection mechanisms for each user or group
    - access authorization is associated with the view
    - each user, or group, accesses the database only via views that are appropriate for the operation they are authorized to carry out

Advantages of views

- Evolution of databases
  - If a database is restored, it is possible to define views that correspond to the eliminated tables
    - the view substitutes the eliminated table which was present in the database prior to restoration
      - it is not necessary to re-formulate the queries written before the restoration and present in the applications that have already been developed

Management of views

Creation and management of views in SQL

Creating a view

CREATE VIEW *ViewName* [*(AttributieList)*]
AS *SQLquery*;

Creating a view

- If the names of the attributes of a view are not specified
  - use those present in the SQL query selection
- The names of the attributes have to be specified if
  - they represent the result of an internal function
  - they represent the result of an expression
  - they are constant
  - two columns (from different tables) have the same name

Cancelling a view

DROP VIEW *ViewName;*

Effect of cancelling tables

- Cancelling a table that a view refers to can have various effects
  - automatic elimination of the associated views
  - automatic invalidation of the associated views
  - prohibition to execute the operation of cancelling the table
- the effect depends on the DBMS utilized

Modifying the definition of a view

ALTER VIEW *ViewName* [*(AttributieList)*]
AS *SQLquery*;

Management of views

Updating views

Updating views

- It is possible to update the data in a view *only* for some typologies of views
- Standard SQL-92
  - views in which a single row of each table corresponds to a single row of the view can be updated
    - univocal correspondence between the tuple of the view and the tuple of the table on which it is defined
    - it is possibile to propagate without ambiguity the changes made to the view to each table on which it is defined

Updating views

- *It is not possible to update* a view which in the farthest block of its defining query
  - lacks the primary key of the table on which it is defined
  - contains joins that represent correspondences to one-to-many or many-to-many
  - contains aggregate functions
  - contains DISTINCT

Example n.1

- View SUPPLIER_CITY

```
CREATE VIEW SUPPLIER_CITY AS
SELECT SId, City
FROM   S;
```

Example n.1: insertion

- Insertion in SUPPLIER_CITY of
  ('S10', 'Rome')
  - corresponds to the insertion in S of
    ('S10',NULL,NULL,'Rome')
  - the attributes SName, #Employees have to admit the value NULL

Example n.1: cancellation

- Cancellation of SUPPLIER_CITY of
  ('S1', 'London')
  - cancellation from S of
    ('S1', 'Smith',20,'London')
  - identification of the tuple to cancel is permitted by the primary key

Example n.1: change

- change to SUPPLIER_CITY of
  ('S1', 'London') in ('S1', 'Milan')

  - change in S of
    ('S1', 'Smith',20,'London') in ('S1', 'Smith',20,'Milan')
  - identification of the tuple to change is permitted by the primary key

Example n.1: updating

- The view SUPPLIER_CITY *can be updated*
  - each tuple of the view corresponds to a single tuple of table S
  - the changes carried out on the view can be propagated to the table on which it is defined

Example n.2

- View NUMEMPLOYEE_CITY

```
CREATE VIEW NUMEMPLOYEE_CITY AS
SELECT DISTINCT #Employees, City
FROM   S;
```

Example n.2: insertion

- Insertion in NUMEMPLOYEE_CITY of
  (40, 'Rome')

  - it is impossible to insert in S
    (NULL,NULL,40,'Rome')
    - the value of the primary key is missing

Example n.2: cancellation

- Cancellation from NUMEMPLOYEE_CITY of (20, 'London')
  - several tuples are associated with the pair (20, 'London')
    - Which tuple has to be cancelled from S?

Example n.2: change

- Change in NUMEMPLOYEE_CITY of
  (20, 'London') in (30, 'Rome')

  - Several tuples are associated with the pair (20, 'London')
    - Which tuple has to be changed in S?

Example n.2: updating

- The view NUMEMPLOYEE_CITY *cannot be updated*
  - the primary key of table S is not present in the view
    - the insertion of new tuples in the view cannot be propagated to S
  - some tuples of the view correspond to several tuples in the table S
    - the association between the tuples in the view and the tuples in the table is ambiguous
    - it is not possible to propagate the changes carried out on the tuples of the view to the tuples of the table on which it is defined

Updating the views

- Some non-updatable views become updatable by changing the SQL expression associated with the view
  - it may be necessary to change the information content of the view

Example n.3: non-updatable view

```
CREATE VIEW SUPPLIER_LONDON AS
SELECT *
FROM S
WHERE City='London';
```

- The view is non-updatable
  - it does not explicitly select the primary key of table S
- It is sufficient to replace the symbol "*" with the name of the attributes

Example n.3: changed view

```
CREATE VIEW SUPPLIER_LONDON AS
SELECT SId, SName, #Employees, City
FROM S
WHERE City='London';
```

- The view is updatable

Example n.4: non-updatable view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT DISTINCT SId, SName
FROM   S, SP
WHERE S.SId=SP.SId AND
          Qty>100;
```

- The view is non-updatable
  - a join is present
  - the keyword DISTINCT is present

Example n.4: changed view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT SId, SName
FROM   S
WHERE SId IN (SELECT SId
                    FROM   SP
                    WHERE Qty>100);
```

- The view is updatable
  - the join was realised using IN
  - the keyword DISTINCT is no longer necessary

Example n.5: non-updatable view

```
CREATE VIEW TOP_SUPPLIER (SId, SName, TotQty) AS
SELECT SId, SName, SUM(Qty)
FROM   S, SP
WHERE S.SId=SP.SId
GROUP BY SId, SName
HAVING SUM(Qty)>500;
```

- The view is non-updatable
  - an aggregate function is present
  - a join is present

Example n.5: changed view

```
CREATE VIEW TOP_SUPPLIER (SId, SName) AS
SELECT SId, SName
FROM   S
WHERE SId IN (SELECT SId FROM SP
                    GROUP BY SId
                    HAVING SUM(Qty)>500);
```

- The view is updatable
  - The "group by" has been moved into the nested query
- The *information content has changed*

Management of views

Check option

CHECK OPTION clause

- For the updatable views use the clause WITH CHECK OPTION
  - this limits the possible updates

CREATE VIEW *ViewName* [*(AttributeList)*]
AS *SQLQuery*
[WITH [LOCAL|CASCADED] CHECK OPTION];

CHECK OPTION clause

- After an update the tuples have to still belong to the view
  - otherwise the operation is prohibited
- A new tuple can be inserted in the view if and only if the tuple satisfies the constraints present in the definition of the view
  - otherwise the operation is prohibited

Example n.1

```
CREATE VIEW PRODUCT_SIZE_SMALL_OR_LARGE (PId,
PName, Size) AS
SELECT PId, PName, Size
FROM P
WHERE Size>=42
WITH CHECK OPTION;
```

- The view is updatable
  - it is not possible to update the tuples present in the view if their size is less than 42

Example n.1

- Content of the view PRODUCT_SIZE_SMALL_OR_LARGE

PId	PName	Size
P2	Jeans	48
P3	Blouse	48
P4	Blouse	44
P6	Shorts	42

46
46
42
40

- Updating operation

```
UPDATE PRODUCT_SIZE_SMALL_OR_LARGE
SET Size=Size-2;
```

Example n.1

- Content of the view PRODUCT_SIZE_SMALL_OR_LARGE

PId	PName	Size
P2	Jeans	48
P3	Blouse	48
P4	Blouse	44
P6	Shorts	42

46
46
42
40

Outside the definition of the view

- Updating operation

```
UPDATE PRODUCT_SIZE_SMALL_OR_LARGE
SET Size=Size-2;
```

Update prohibited

CHECK OPTION clause

CREATE VIEW *ViewName* [*(AttributeList)*]
AS *SQLQuery*
[WITH [LOCAL|CASCADED] CHECK OPTION];

- When a view is defined in terms of other views
  - if LOCAL is specified
    - the update is correct only on the most external view
  - if CASCADED is specified
    - the update is correct on all the views involved
    - default options

Example n.2

```
CREATE VIEW PRODUCT_SIZE_MEDIUM(PId, PName, Size) AS
SELECT PId, PName, Size
FROM   PRODUCT_SIZE_SMALL_OR_LARGE
WHERE Size<=46
WITH CASCADED CHECK OPTION;
```

- I can update the content of the view PRODUCT_SIZE_MEDIUM using only sizes between 42 and 46
- Default behaviour

Example n.2

- Content of the view PRODUCT_SIZE_MEDIUM

PId	PName	Size
P4	Blouse	44
P6	Shorts	42

→ 42
→ 40

- Updating operation

```
UPDATE PRODUCT_SIZE_MEDIUM
SET SIZE=Size-2;
```

Example n.2

- Content of the view PRODUCT_SIZE_MEDIUM

PId	PName	Size
P4	Blouse	44
P6	Shorts	42

42
40

Outside definition of the view PRODUCT_SIZE_SMALL_OR_LARGE

- Updating operation

```
UPDATE PRODUCT_SIZE_MEDIUM
SET SIZE=Size-2;

CREATE VIEW PRODUCT_SIZE_SMALL_OR_LARGE (PId,
PName, Size) AS
SELECT PId, PName, Size
FROM   P
WHERE Size>=42
WITH CHECK OPTION;
```

Example n.2

- Content of the view PRODUCT_SIZE_MEDIUM

PId	PName	Size
P4	Blouse	44
P6	Shorts	42

42
40

Outside definition of the view PRODUCT_SIZE_SMALL_OR_LARGE

- Updating operation

```
UPDATE PRODUCT_SIZE_MEDIUM
SET SIZE=Size-2;
```

- With CASCADED CHECK OPTION
  - Update prohibited because of PRODUCT_SIZE_SMALL_OR_LARGE

Example n.3

```
CREATE VIEW PRODUCT_SIZE_MEDIUM(PId, Pname, Size) AS
SELECT PId, PName, Size
FROM   PRODUCT_SIZE_SMALL_OR_LARGE
WHERE Size<=46
WITH LOCAL CHECK OPTION;
```

- Control is carried out *only* on the view PRODUCT_SIZE_MEDIUM
  - this is updatable with sizes below or equal to 46

Example n.2

- Content of the view PRODUCT_SIZE_MEDIUM

PId	PName	Size
P4	Blouse	44
P6	Shorts	42

→ 42
→ 40

- Updating operation

```
UPDATE PRODUCT_SIZE_MEDIUM
SET SIZE=Size-2;

CREATE VIEW PRODUCT_SIZE_MEDIUM(PId, Pname, Size) AS
SELECT PId, PName, Size
FROM   PRODUCT_SIZE_SMALL_OR_LARGE
WHERE Size<=46
WITH LOCAL CHECK OPTION;
```

Example n.2

- Content of the view PRODUCT_SIZE_MEDIUM

PId	PName	Size
P4	Blouse	44
P6	Shorts	42

→ 42
→ 40

- Updating operation

```
UPDATE PRODUCT_SIZE_MEDIUM
SET SIZE=Size-2;
```

- With LOCAL CHECK OPTION
  - Updating allowed

Management of views

Privacy management

Views and privacy management

- Views enable the identification of data subsets
  - Identified by a SELECT expression
- Assigning a user access to specific views means limiting
  - its visibility on existing tables
  - the operations it can execute

Example n.1

```
CREATE VIEW SUPPLIER_LONDON (SId, SName, #Employees)
AS
SELECT SId, SName, #Employees
FROM   S
WHERE City='London'
WITH CHECK OPTION;
```

Example n.1

```
CREATE VIEW SUPPLIER_LONDON (SId, SName, #Employees)
AS
SELECT SId, SName, #Employees
FROM   S
WHERE City='London'
WITH CHECK OPTION;
```

- The view SUPPLIER_LONDON selects only data on suppliers in London
- A user has access *only* to this view
  - it cannot access table S
    - it cannot operate on suppliers whose offices are not in London

Example n.2

```
CREATE VIEW SUPPLIER_CODE_NAME (SId, SName) AS
SELECT SId, SName
FROM   S;
```

- The view SUPPLIER_CODE_NAME selects only the code and the name of the suppliers
- A user that has access *only* to this view
  - Cannot access table S
    - Cannot operate on the attributes #Employees and City

Data dictionary

- The data dictionary contains the metadata of a relational database
  - metadata is information (data) on the data
  - it describes the objects of the database (tables, views,…)
- In the data dictionary views are defined which limit the visibility of the individual users on the metadata of the dictionary
  - each user can only see the information regarding objects in the database defined by itself

Example: Oracle

- The Oracle DBMS makes numerous views available which describe the data created by a user
  - USER_TABLES contains metadata regarding the user's tables
  - USER_TAB_STATISTICS contains the statistics calculated on the user's tables
  - USER_TAB_COL_STATISTICS contains the statistics calculated on the columns of the user's tables