Database Programming

KRYSTIAN WOJTKIEWICZ

VIEWS are coming...

Views

VIEW

a selection of data and expressions values constructed on the basis of data taken from one or several relations or other views

is visible by the user as table

are stored in the form of their definitions (this does not apply to so called materialized views)

- every reference to view creates its structure

Views are defined for the following reasons

to limit access to some data in a relation,

to help the user retrieve the results of complex queries using simple queries based on views,

to free the user from analyzing the data structure,

to provide information that is seen differently by different users.

Simple views

SELECT doesn’t contain

- joins
- group or analytical functions,
- ordering,
- DISTINCT qualifier,
- GROUP BY clause
- correlated subqueries
- subqueries at all ☺,
- CONNECT BY and START WITH.

Complex view and modifiable view

do not contain joins, except for joins for which joined relations keep key (key-preserved tables) in view. If the view contains a join for which the linked relations keep the key, then the DML operation on the view must apply to only one relations. Additionally:

- to perform the INSERT operation, the view must select primary key attributes and all mandatory attributes of the key-preserved relation,
- to perform the UPDATE operation, all modified attributes must come from a key-preserved relation,
- for the DELETE operation, the join operation can only apply to one key-preserved relation.

do not contain group or analytical functions, ordering, DISTINCT qualifier, GROUP BY clause,

do not contain correlated subqueries and subqueries in the SELECT clause

do not contains CONNECT BY and START WITH clauses,

do not contain expressions or pseudo-columns (INSERT and UPDATE commands based on the view cannot apply to them).

CREATE VIEW syntax

CREATE VIEW View_name [({view_attribute [, ...]})]

AS SELECT_command

[**WITH CHECK OPTION** [**CONSTRAINT** constraint_name]] |

[**WITH READ ONLY**]

The optional WITH CHECK OPTION clause (only for simple views) allows updating through the view only if changed or new records would still appear in the view (they will meet the condition after the WHERE clause of the SELECT command).

After the WITH CHECK OPTION clause, one can additionally place the CONSTRAINT constraint_name clause to name the constraint created.

The optional WITH READ ONLY clause (for simple views only) prevents data updates through the view.

Example

Task 52. *Define a simple view providing part of the data (nickname, date of entry to the herd, ration of mice) of cats that belong to band No. 3.*

```
CREATE VIEW Band3
AS SELECT nickname,in_herd_since,mice_ration
   FROM Cats
   WHERE band_no=3;
 view BAND3 created.
```

Lets check it:

```
UPDATE Band3
SET mice_ration=55
WHERE nickname='ZERO';
1 rows updated.
```

```
SELECT nickname,mice_ration
FROM Cats
WHERE nickname='ZERO';

NICKNAME    MICE_RATION
----------- --------------
ZERO        55
ROLLBACK;
rollback complete.
```

Example

Task 53. *Define a complex view which provides the names of the bands and the minimum, maximum and average mice ration in each band.*

```
CREATE VIEW Mice_in_bands
(name,minm,maxm,average)

AS SELECT
B.name,MIN(mice_ration),MAX(mice_ration),

          AVG(mice_ration)

   FROM Cats JOIN Bands B USING(band_no)

   GROUP BY B.name;

 view MICE_IN_BANDS

created.
```

```
SELECT * FROM Mice_in_bands;
```

NAME	MINM	MAXM	AVERAGE
BLACK KNIGHTS	24	72	56,8
WHITE HUNTERS	20	75	49,75
SUPERIORS	22	1	50
PINTO HUNTERS	40	65	49,4

Example

Task 54. *Define a view that provides part of the data of cats from band No. 4, that enables DML operations through the view only for this band.*

```
CREATE VIEW Band4

AS SELECT
nickname,name,function,mice_ration,band_no

   FROM Cats WHERE band_no=4

   WITH CHECK OPTION;

 view BAND4 created.
```

```
INSERT INTO Band4 VALUES ('LALA','KOBOL','CAT',30,3);


Error starting at line 1 in command:

INSERT INTO Band4 VALUES ('LALA','KOBOL','CAT',30,3)
Error report:

SQL Error: ORA-01402: naruszenie klauzuli WHERE dla
perspektywy z WITH CHECK OPTION

01402. 00000 -  "view WITH CHECK OPTION where-clause
violation"

*Cause:

*Action:
```

Example

Task 54. *After a long reflection, the Tiger concluded that the situation in the herd is as perfect as he is. Therefore, he ordered an IT specialist to define a "guarding" view so that the existing status quo could not be violated.*

The view have to ensure that:

- *no new band could be created,*
- *no cat outside the chiefs' elite has usurped the right to be a chief,*
- *cats could only perform existing functions,*
- *the ration of the mice was at least equal to the value of cats' social minimum level (6 mice) and that could not exceed the ration of the mice of Tiger.*

Define a view that meets these requirements.

```
CREATE OR REPLACE VIEW Status_quo_of_cats
AS SELECT nickname,name,chief,function,mice_ration,band_no
   FROM Cats
   WHERE (band_no IN (SELECT band_no FROM Cats)
                     OR band_no=5 OR band_no IS NULL)
          AND (chief IN (SELECT chief FROM Cats)
               OR chief IS NULL)
          AND (function IN (SELECT function FROM Cats)
              OR function='HONORARY' OR function IS NULL)
          AND (mice_ration BETWEEN 6
                           AND(SELECT mice_ration
                               FROM Cats
                               WHERE nickname='TIGER')
              OR mice_ration IS NULL)
          AND nickname<>'TIGER'
   WITH CHECK OPTION CONSTRAINT nothing_more;
 view STATUS_QUO_OF_CATS created.
```

Indexes

INDEX

Indexes do not belong to the ANSI SQL standard although they are implemented by most DBMS, including Oracle.

Their creation and removal are other elements of the DDL component of SQL language.

A data structure for accelerating the search for data from relation

A data structure for enforcing unique attribute values.

Usually are created using the so-called balanced B-trees (B*-tree indexes).

CREATE INDEX syntax

CREATE [UNIQUE] INDEX Index_name

ON Table_name({attribute_name [DESC | ASC] [, ...]})

UNIQUE forces the uniqueness of the set of attribute values listed after the relation's name.

DESC specifies the descending direction of building of the index column (by default, the index column is built in ascending order - ASC) via the index attribute.

Index attribute may be also argument of function. Such an index is called a **functional index**.

There are two types of indexes:

- **simple** index: based on one attribute,
- **complex** index: based on more than one attribute.

INDEX

The following conditions must be met for the index to be used:

- the indexed attribute must appear in the WHERE clause,
- the attribute in the WHERE clause cannot be a function argument or part of an expression.

INDEX

The selection and definition of indexes is part of the physical database design. When choosing them, the following principles should apply:

- setting up an index is beneficial for relations with more rows (over 200 in Oracle). With small relations, the index reading time may be greater than the gain of the command execution time,
- indexation is recommended for attributes whose values rather do not repeat,
- indexation is advisable for the attributes often used in the WHERE clause (also in connecting conditions),
- if two or more attributes often appear together in a WHERE clause, we should consider creation of the complex indexes,
- avoid more than three indexes for one relation (overloading DML operations). This rule does not apply if SELECT is the most frequently used command,
- with batch modification of the relation, it is advisable to temporarily delete the indexes superimposed on it, because each batch command causes an independent refresh of the index, which takes time. After batch modification, one ought to restore the indexes.

SEQUENCES

a database object for generating unique values, usually for primary keys

```
CREATE SEQUENCE sequence_name
[START WITH begining_value]
[INCREMENT BY step]
[MAXVALUE maximum_value | NOMAXVALUE]
[MINVALUE minimum_value | NOMINVALUE] [CYCLE | NOCYCLE];

ALTER SEQUENCE sequence_name
[INCREMENT BY step]
[MAXVALUE maximum_value | NOMAXVALUE]
[MINVALUE minimum_value | NOMINVALUE] [CYCLE | NOCYCLE];

DROP SEQUENCE sequence_name;
```

SEQUENCES

START WITH specifies the first number to be returned by the sequence,

INCREMENT BY determines about how much are to be increased following numbers (1 by default),

MAXVALUE and **MINVALUE** specify the upper and lower limits of the sequence value (default values: **NOMAXVALUE** and **NOMINVALUE** respectively),

CYCLE determines whether the sequence should be created cyclically (**NOCYCLE** by default) from **MINVALUE** to **MAXVALUE** (reverse for a descending sequence).

The **CURRVAL** pseudo attribute is used to read the current sequence value.

```
SELECT
sequence_name.CURRVAL
FROM Dual;
```

```
SELECT
sequence_name.NEXTVAL
FROM Dual;
```

Example

Task 55. *Create a sequence that provides the opportunity to enter new bands into Bands relation with consecutive their numbers, starting from 6.*

```
CREATE SEQUENCE Band_numbers
START WITH 6;
 sequence BAND_NUMBERS created.

INSERT INTO Bands
VALUES (Band_numbers.NEXTVAL,'NEW','FOREST',NULL);

1 rows inserted.
```

```
SELECT * FROM Bands;

BAND_NO  NAME            SITE         BAND_CHIEF
-------- --------------- ------------ -----------
1        SUPERIORS        WHOLE AREA   TIGER
2        BLACK KNIGHTS    FIELD        BALD
3        WHITE HUNTERS    ORCHARD      ZOMBIES
4        PINTO HUNTERS    HILLOCK      REEF
5        ROCKERSI         FARM
6        NEW              FOREST

6        rows selected

ROLLBACK;  rollback complete.

DROP SEQUENCE Band_numbers;
 sequence BAND_NUMBERS dropped.
```

SEQUENCES

The CURRVAL and NEXTVAL pseudo attributes can be used:

− in the SELECT clause of the SELECT statement,

− in the list of values in INSERT command,

− in the SET clause of the UPDATE command

− only in the main (most external) query.

The CURRVAL and NEXTVAL pseudo attributes cannot be used:

− in the SELECT clause defining the view,

− with the DISTINCT qualifier,

− if in the command appears ORDER BY, GROUP BY, CONNECT BY, HAVING clauses,

− with the operators UNION, INTERSECT, MINUS − in subqueries.

TRANSACTIONS

A successful or unsuccessful operation consisting of a series of changes in one or more relations of a database.

The two basic commands for the DCL component of SQL language are the **COMMIT** command applied to explicitly commit the transaction and the **ROLLBACK** command to explicitly roll back the transaction

There are two types of transactions:

DDL transactions - equivalent to single DDL operation,

DML transactions - consisting of any number of DML operations.

Transaction start

the first executable DML or DDL instruction

Transaction end

COMMIT (explicit transaction confirmation) or ROLLBACK (explicit transaction rolling back) command,

DDL command,

some type of error (e.g. dead-lock),

end of the program session,

computer failure.

TRANSACTIONS

A successful or unsuccessful operation consisting of a series of changes in one or more relations of a database.

The transaction is **committed** implicitly:

- before DDL command,
- after the DDL command,
- after normal disconnection from the base.

The transaction is **rolled back** implicitly:

- after a system error.

Save points allow one to separation part of the transaction for the rolling back only that part. The save point is explicitly created

```
SAVEPOINT savepoint_name;
```

Rolling back part of the transaction to the save point (without closing) executes the command:

```
ROLLBACK TO SAVEPOINT
savepoint_name;
```

DML commands can be implicitly committed during the transaction. To allow this, one ought to use the SET AUTOCOMMIT command with the syntax:

```
SET AUTOCOMMIT {ON | OFF |
number_of_command}
```

After executing the above command, the DML transaction commands will be always implicitly committed when their number equals the number specified in the syntax above.
If an ON element occurs, an implicit commit will follow each DML command (number_commands = 1).