7.4 SQL Views

SQL views are part of the external data model. A **view** is defined by means of an SQL query and its content is generated upon invocation of the view by an application or by another query. In this way, it can be considered as a virtual table without physical tuples. Views offer several advantages. By hiding complex queries, such as join queries or correlated queries from their users, views facilitate ease of use. They can also provide data protection by hiding columns or tuples from unauthorized users. Views allow for logical data independence, which makes them a key component in the three-layer database architecture (see Chapter 1).

Views can be defined using the CREATE VIEW statement. Here you can see three examples:

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50

CREATE VIEW TOPSUPPLIERS_SF
AS SELECT * FROM TOPSUPPLIERS
WHERE SUPCITY = 'San Francisco'

CREATE VIEW ORDEROVERVIEW(PRODNR, PRODNAME,
TOTQUANTITY)
AS SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY)
FROM PRODUCT AS P LEFT OUTER JOIN PO_LINE AS POL
ON (P.PRODNR = POL.PRODNR)
GROUP BY P.PRODNR
```

The first view is called TOPSUPPLIERS and offers the supplier number and supplier name of all suppliers whose status is greater than 50. The second view refines the first by adding an additional constraint: SUPCITY = “San Francisco”. The third view is called ORDEROVERVIEW. It contains the product number, product name, and total ordered quantity of all products. Note that the left outer join in the view definition ensures that products with no outstanding orders are included in the result.

Once the views have been defined, they can be used in applications or other queries as follows:

```
SELECT * FROM TOPSUPPLIERS_SF

SELECT * FROM ORDEROVERVIEW
WHERE PRODNAME LIKE '%CHARD%'
```

The first query selects all tuples from the view TOPSUPPLIERS_SF, which we defined earlier. The second query retrieves all tuples from the ORDEROVERVIEW view where the product name contains the string “CHARD”.

The RDBMS automatically modifies queries that query views into queries on the underlying base tables. Suppose we have our view TOPSUPPLIERS and a query which uses it as follows:

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50

SELECT * FROM TOPSUPPLIERS
WHERE SUPCITY= 'Chicago'
```

Both view and query can be modified into the following query, which works directly on the SUPPLIER table:

```
SELECT SUPNR, SUPNAME
FROM SUPPLIER
WHERE SUPSTATUS > 50 AND SUPCITY='Chicago'
```

This is often referred to as query modification. View materialization is an alternative strategy for DBMSs to perform queries on views, in which a physical table is created when the view is first queried. To keep the materialized view table up-to-date, the DBMS must implement a synchronization strategy whenever the underlying base tables are updated. Synchronization can be performed as soon as the underlying base tables are updated (immediate view maintenance) or it can be postponed until just before data are retrieved from the view (deferred view maintenance).

Some views can be updated. In this case, the view serves as a window through which updates are propagated to the underlying base table(s). Updatable views require that INSERT, UPDATE, and DELETE statements on the view can be unambiguously mapped to INSERTs, UPDATEs, and DELETEs on the underlying base tables. If this property does not hold, the view is read only.

Let’s reconsider our view ORDEROVERVIEW, which we defined earlier:

```
CREATE VIEW ORDEROVERVIEW(PRODNR, PRODNAME,
TOTQUANTITY)
AS SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY)
FROM PRODUCT AS P LEFT OUTER JOIN PO_LINE AS POL
ON (P.PRODNR = POL.PRODNR)
GROUP BY P.PRODNR
```

The following UPDATE statement tries to set the total ordered quantity to 10 for product number 0154.

```
UPDATE VIEW ORDEROVERVIEW
SET TOTQUANTITY = 10
WHERE PRODNR = '0154'
```

Since there are multiple ways to accomplish this update, the RDBMS will generate an error. A view update is feasible when only one possible update on the base table(s) can accomplish the desired update effect on the view. In other words, this is an example of a view which is not updatable.

Various requirements can be listed for views to be updatable. They typically depend upon the RDBMS vendor. Common examples are: no DISTINCT option in the SELECT component; no aggregate functions in the SELECT component; only one table name in the FROM component; no correlated subquery in the WHERE component; no GROUP BY in the WHERE component; and no UNION, INTERSECT, or EXCEPT in the WHERE component.

Another issue may arise in the case that an update on a view can be successfully performed. More specifically, in the case that rows are inserted or updated through an updatable view, there is the chance that these rows no longer satisfy the view definition. Hence, the rows cannot be retrieved through the view anymore. The WITH CHECK option allows us to avoid such undesired effects by checking the UPDATE and INSERT statements for conformity with the view definition. To illustrate this, consider the following two examples of UPDATE statements on our TOPSUPPLIERS view, which has now been augmented with a WITH CHECK option.

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
```

```
WHERE SUPSTATUS > 50
WITH CHECK OPTION

UPDATE TOPSUPPLIERS
SET STATUS = 20
WHERE SUPNR = '32'

UPDATE TOPSUPPLIERS
SET STATUS = 80
WHERE SUPNR = '32'
```

The first UPDATE will be rejected by the RDBMS because the supplier status of supplier 32 is updated to 20, whereas the view requires the supplier status to be greater than 50. The second update statement will be accepted by the RDBMS because the new supplier status is bigger than 50.

Connections

We will return to views in Chapter 20, where we will use them as a mechanism to safeguard privacy and security in a Big Data and analytics setting.

Retention Questions

- What are SQL views and what can they be used for?
- What is a view WITH CHECK option? Illustrate with an example.