**CHAPTER FIVE**

**Relational Views in SQL**

- A view is a single *virtual table* that is derived from other tables

- The other tables could be base tables or previously defined views

- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views

- There are no limitations on querying a view

- The **CREATE VIEW** command is used to specify a view by specifying a (virtual) table name and a defining query

- The view attribute names can be *inherited from* the attribute names of the tables in the defining query

Examples:

**V1**: CREATE VIEW WORKS\_ON1

|  |  |  |
| --- | --- | --- |
| AS | SELECT | FNAME, LNAME, PNAME, HOURS |
|  | FROM | EMPLOYEE, PROJECT, WORKS\_ON |
|  | WHERE | SSN=ESSN AND PNO=PNUMBER ; |

**V2**: CREATE VIEW DEPT\_INFO

(DEPT\_NAME, NO\_OF\_EMPS, TOTAL\_SAL)

AS SELECT DNAME, COUNT (\*), SUM (SALARY) FROM DEPARTMENT, EMPLOYEE

WHERE DNUMBER=DNO GROUP BY DNAME ;

- In V1 the names of the view attribute names are inherited

- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query

**QUERIES ON VIEWS:**

Example: Retrieve the last name and first name of all employees who work on 'ProjectX'.

|  |  |  |
| --- | --- | --- |
| **QV1:** | SELECT | PNAME, FNAME, LNAME |
|  | FROM | WORKS\_ON1 |
|  | WHERE | PNAME='ProjectX' ; |

- Without the view WORKS\_ON1, this query specification would require two join conditions

- A view can be defined to simplify frequently occurring queries

- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified



- Hence, the view is *not* realized at the time of *view definition*, but rather at the time we specify a query on the view

- A view is removed using the **DROP VIEW** command

Example:

**V1A**: DROP VIEW WORKS\_ON1 ;

**V2A**: DROP VIEW DEPT\_INFO ;

- Views can also be used as a security and authorization mechanism

(see Chapter 20)

**UPDATING OF VIEWS:**

- A view update operation may be mapped in multiple ways to update operations on the defining base relations

- The topic of updating views is still an active research area

Example: Suppose we issue the command in UV1 to update the WORKS\_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

|  |  |  |
| --- | --- | --- |
| **UV1:** | UPDATE | WORKS\_ON1 |
|  | SET | PNAME = 'ProductY' |
|  | WHERE | LNAME='Smith' AND FNAME='John' |

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:

(1) Change the name of the 'ProductX' tuple in the PROJECT relation to

'ProductY'



- It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

|  |  |  |
| --- | --- | --- |
| **(1):** | UPDATE | PROJECT |
|  | SET | PNAME = 'ProductY' |
|  | WHERE | PNAME = 'ProductX' |

(2) Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the

'ProductX' PROJECT tuple

- This is most likely the update the user means

**(2):** UPDATE WORKS\_ON

SET PNO = (SELECT PNUMBER FROM PROJECT WHERE PNAME='ProductY')

WHERE ESSN = (SELECT SSN FROM EMPLOYEE WHERE LNAME='Smith' AND FNAME='John')

WHERE PNAME='ProductX')

- Some view updates may not make much sense; for example, modifying the TOTAL\_SAL attribute of DEPT\_INFO as in UV2

**UV2:** MODIFY DEPT\_INFO SET TOTAL\_SAL=100000

WHERE DNAME='Research' ;

- In general, we cannot guarantee that any view can be updated

- A view update is unambiguous only if *one update* on the base relations can accomplish the desired update effect on the view

- If a view update can be mapped to *more than one update* on the underlying base relations, we must have a certain procedure to choose the desired update

- We can make the following general observations:

• A view with a single defining table is updatable if the view attributes contain the primary key

• Views defined on multiple tables using joins are generally not updatable

Views defined aggregate functions are not updatable.

**Creating Indexes in SQL**

- An SQL base relation generally corresponds to a stored file

- Statements can create and drop indexes on base relations

- These statements have been *removed* from SQL2 because they specify physical access paths - not logical concepts

- One or more *indexing attributes* are specified for each index

- The **CREATE INDEX** statement is used

- Each index is given an *index name*

**I1:** CREATE INDEX LNAME\_INDEX ON EMPLOYEE (LNAME );

**-** The index entries are in ascending (ASC) order of the indexing attributes; DESC specifies descending order

- An index can be created on a combination of attributes

**I2:** CREATE INDEX NAMES\_INDEX

ON EMPLOYEE ( LNAME ASC, FNAME DESC, MINIT );

- Two options on indexes are UNIQUE and CLUSTER

- To specify the *key constraint* on the indexing attribute or combination of attributes, the keyword **UNIQUE** is used

**I3:** CREATE UNIQUE INDEX SSN\_INDEX ON EMPLOYEE (SSN);

**- This is** best done before any tuples are inserted in the relation

- An attempt to create a unique index on an existing base table will fail if the current tuples in the table do not obey the constraint

- A second option on index creation is to specify that the index is a clustering index using the keyword **CLUSTER**

- A base relation can have *at most one* clustering index, but any number of non-clustering indexes

Example:

**I4:** CREATE INDEX DNO\_INDEX

ON EMPLOYEE (DNO) CLUSTER;

- A clustering and unique index in SQL is similar to the primary index

- A clustering but non-unique index in SQL is similar to the clustering index

- A non-clustering index is similar to the secondary index

- Each DBMS will have its own index implementation technique; in most cases, some variation of the B+-tree data structure is used

- To drop an index, we issue the **DROP INDEX** command

- The index name is needed to refer to the index when it is to be dropped

Example:

**I5:** DROP INDEX DNO\_INDEX;

**Joined Relations**

- Can specify a "joined relation" in the FROM-clause

- Looks like any other relation but is the result of a join

- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Examples:

**Q1:** SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME FROM EMPLOYEE E S

WHERE E.SUPERSSN=S.SSN

can be written as:

**Q1A:** SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME

FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES

ON E.SUPERSSN=S.SSN)

|  |  |  |
| --- | --- | --- |
| **Q2**: | SELECT | FNAME, LNAME, ADDRESS |
|  | FROM | EMPLOYEE, DEPARTMENT |
|  | WHERE | DNAME='Research' AND DNUMBER=DNO |

Could be written as:

**Q2A**: SELECT FNAME, LNAME, ADDRESS FROM (EMPLOYEE JOIN DEPARTMENT

ON DNUMBER=DNO)

WHERE DNAME='Research'

Or as:

**Q2B**: SELECT FNAME, LNAME, ADDRESS

FROM (EMPLOYEE NATURAL JOIN DEPARTMENT AS DEPT (DNAME, DNO, MSSN, MSDATE)

WHERE DNAME='Research'