VIRTUAL TABLES: CREATING A VIEW

The output of a relational operator such as SELECT is another relation (or table). Suppose that at the end of every day, you would like to get a list of all products to reorder, that is, products with a quantity on hand that is less than or equal to the minimum quantity. Instead of typing the same query at the end of every day, wouldn't it be better to permanently save that query in the database? That's the function of a relational view. A view is a virtual table based on a SELECT query. The query can contain columns, computed columns, aliases, and aggregate functions from one or more tables. The tables on which the view is based are called base tables.

You can create a view by using the CREATE VIEW command:

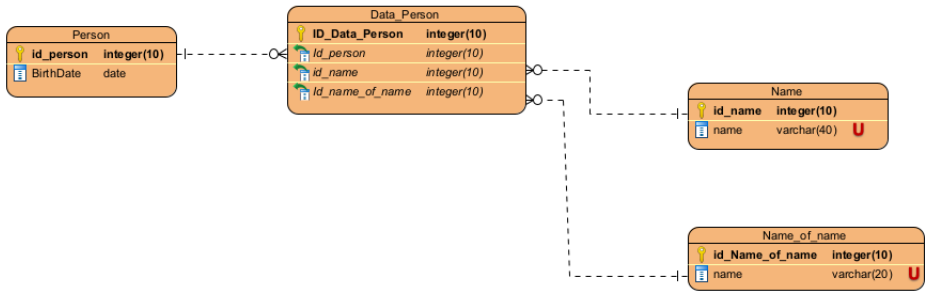
CREATE VIEW view_name AS SELECT query

The CREATE VIEW statement is a data definition command that stores the subquery specification—the SELECT statement used to generate the virtual table—in the data dictionary.

A relational view has several special characteristics:

- You can use the name of a view anywhere a table name is expected in a SQL statement.
- Views are dynamically updated. That is, the view is re-created on demand each time it is invoked. Therefore, if new products are added (or deleted) to meet the criterion P_PRICE > 50.00, those new products will automatically appear (or disappear) in the PRICEGT50 view the next time the view is invoked.
- Views provide a level of security in the database because the view can restrict users to only specified columns and specified rows in a table. For example, if you have a company with hundreds of employees in several departments, you could give the secretary of each department a view of only certain attributes and for the employees that belong only to that secretary's department.
- Views may also be used as the basis for reports. For example, if you need a report that shows a summary of total product cost and quantity-on-hand statistics grouped by vendor, you could create a PROD_STATS view as:

```
CREATE VIEW PROD_STATS AS
SELECT V_CODE, SUM(P_QOH*P_PRICE) AS TOTCOST,
                  MAX(P_QOH) AS MAXQTY, MIN(P_QOH) AS MINQTY,
                  AVG(P_QOH) AS AVGQTY FROM      PRODUCT
GROUP BY V_CODE;
```



An example of a view presentation

JOINING DATABASE TABLES

The ability to combine (join) tables on common attributes is perhaps the most important distinction between a relational database and other databases. A join is performed when data are retrieved from more than one table at a time.

To join tables, you simply list the tables in the FROM clause of the SELECT statement. The DBMS will create the Cartesian product of every table in the FROM clause. However, to get the correct result—that is, a natural join—you must select only the rows in which the common attribute values match. To do this, use the WHERE

clause to indicate the common attributes used to link the tables (this WHERE clause is sometimes referred to as the join condition).

The join condition is generally composed of an equality comparison between the foreign key and the primary key of related tables.

Naturally, if an attribute name occurs in several places, its origin (table) must be specified. If you fail to provide such a specification, SQL will generate an error message to indicate that you have been ambiguous about the attribute's origin.

If you do not specify the WHERE clause, the result will be the Cartesian product.

All of the SQL commands can be used on the joined tables.

When joining three or more tables, you need to specify a join condition for each pair of tables. The number of join conditions will always be N-1, where N represents the number of tables listed in the FROM clause. For example, if you have three tables, you must have two join conditions; if you have five tables, you must have four join conditions; and so on.

Remember, the join condition will match the foreign key of a table to the primary key of the related table.

Finally, be careful not to create circular join conditions. For example, if Table A is related to Table B, Table B is related to Table C, and Table C is also related to Table A, create only two join conditions: join A with B and B with C. Do not join C with A!

JOINING TABLES WITH AN ALIAS

An alias may be used to identify the source table from which the data are taken. The aliases P and V are used to label the PRODUCT and VENDOR tables in the next command sequence. Any legal table name may be used as an alias. (Also notice that there are no table name prefixes because the attribute listing contains no duplicate names in the SELECT statement.)

```
SELECT P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE FROM
      PRODUCT P, VENDOR V
WHERE P.V_CODE = V.V_CODE ORDER BY P_PRICE;
```

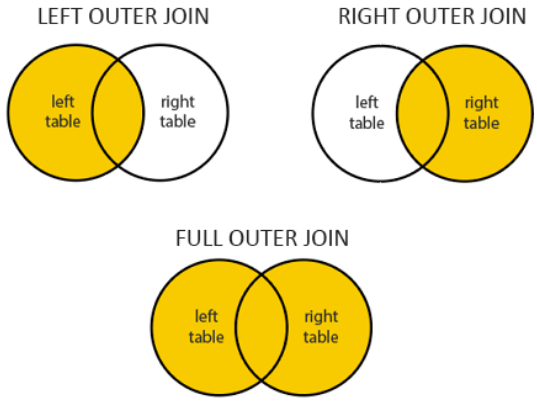
RECURSIVE JOINS

An alias is especially useful when a table must be joined to itself in a recursive query.

Using the data in the EMP table, you can generate a list of all employees with their managers' names by joining the EMP table to itself. In that case, you would also use aliases to differentiate the table from itself. The SQL command sequence would look like this:

```
SELECT E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME FROM EMP E, EMP M
WHERE E.EMP_MGR=M.EMP_NUM ORDER BY E.EMP_MGR;
```

OUTER JOINS



SUMMARY

- The SQL commands can be divided into two overall categories: data definition language (DDL) commands and data manipulation language (DML) commands.

- The ANSI standard data types are supported by all RDBMS vendors in different ways. The basic data types are NUMBER, INTEGER, CHAR, VARCHAR, and DATE.
- The basic data definition commands allow you to create tables, indexes, and views. Many SQL constraints can be used with columns. The commands are CREATE TABLE, CREATE INDEX, CREATE VIEW, ALTER TABLE, DROP TABLE, DROP VIEW, and DROP INDEX.
- DML commands allow you to add, modify, and delete rows from tables. The basic DML commands are SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK.
- The INSERT command is used to add new rows to tables. The UPDATE command is used to modify data values in existing rows of a table. The DELETE command is used to delete rows from tables. The COMMIT and ROLLBACK commands are used to permanently save or roll back changes made to the rows. Once you COMMIT the changes, you cannot undo them with a ROLLBACK command.
- The SELECT statement is the main data retrieval command in SQL. A SELECT statement has the following syntax:

```
SELECT      columnlist
FROM  tablelist
[WHERE      conditionlist ]
[GROUP BY   columnlist ]
[HAVING     conditionlist ]
[ORDER BY   columnlist [ASC | DESC] ] ;
```

- The column list represents one or more column names separated by commas. The column list may also include computed columns, aliases, and aggregate functions. A computed column is represented by an expression or formula (for example, P_PRICE * P_QOH). The FROM clause contains a list of table names or view names.
- The WHERE clause can be used with the SELECT, UPDATE, and DELETE statements to restrict the rows affected by the DDL command. The condition list represents one or more conditional expressions separated by logical operators (AND, OR, and NOT). The conditional expression can contain any comparison operators (=, >, <, >=, <=, and <>) as well as special operators (BETWEEN, IS NULL, LIKE, IN, and EXISTS).
- Aggregate functions (COUNT, MIN, MAX, and AVG) are special functions that perform arithmetic computations over a set of rows. The aggregate functions are usually used in conjunction with the GROUP BY clause to group the output of aggregate computations by one or more attributes. The HAVING clause is used to restrict the output of the GROUP BY clause by selecting only the aggregate rows that match a given condition.
- The ORDER BY clause is used to sort the output of a SELECT statement. The ORDER BY clause can sort by one or more columns and can use either ascending or descending order.
- You can join the output of multiple tables with the SELECT statement. The join operation is performed every time you specify two or more tables in the FROM clause and use a join condition in the WHERE clause to match the foreign key of one table to the primary key of the related table. If you do not specify a join condition, the DBMS will automatically perform a Cartesian product of the tables you specify in the FROM clause.
- The natural join uses the join condition to match only rows with equal values in the specified columns. You could also do a right outer join and left outer join to select the rows that have no matching values in the other related table.