SQL – Creating Tables and Views

Dr. Villanes

Creating a New Table

Creating a New Table

There are three ways to create new tables in SQL:

Method 1	Copy columns and rows from existing table.
Method 2	Copy columns but no rows from an existing table.
Method 3	Define only the columns in the SQL code.

```
+--------+-------------+------------+--------+---------+
| emp_id | emp_name    | hire_date  | salary | dept_id |
+--------+-------------+------------+--------+---------+
|      1 | Ethan Hunt  | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana| 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor| 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard| 2007-01-03 |   7200 |       3 |
|      5 | Martin Blank| 2008-06-24 |   5600 |    NULL |
+--------+-------------+------------+--------+---------+
```

Method 1: Copy columns and rows from existing table(s)

CREATE TABLE *table-name* **AS SELECT …;**

```
proc sql;
create table birthmonths as
select  Employee_Name as Name format=$25.,
        City format=$25.,
        month(Birth_Date) as BirthMonth
          'Birth Month' format=3.
   from jupiter.employee_payroll as p,
        jupiter.employee_addresses as a
   where p.Employee_ID=a.Employee_ID
         and Employee_Term_Date is missing
   order by BirthMonth,City,Name;
quit;
```

```
create table practice.temp as
select movies.movie_name as name
from practice.movies
where movies.movie_name like 'A%'
UNION
select genres.genre as name
from practice.genres
where genres.genre like 'B%';
```

Method 2: Copy columns but no rows from an existing table

CREATE TABLE table2
LIKE table1;

```
proc sql;
create table work.new_sales_staff
   like jupiter.sales;
quit;
```

SAS Log:

```
proc sql;
create table work.new_sales_staff
   like jupiter.sales;
NOTE: Table WORK.NEW_SALES_STAFF created,
with 0 rows and 9 columns.
quit;
```

CREATE TABLE table2 AS
SELECT * FROM mytable1
WHERE 0;

```
CREATE TABLE exercise.copied AS
SELECT * FROM exercise.records where 0;
```

Method 3: Define only the columns in the SQL code.

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

```
proc sql;
CREATE TABLE discounts (
    Product_ID num format=z12.,
    Start_Date date,
    End_Date date,
    Discount num format=percent.
);
quit;
```

```
CREATE TABLE contacts (
 contact_id integer PRIMARY KEY,
 first_name text NOT NULL,
 last_name text NOT NULL,
 email text NOT NULL UNIQUE,
 phone text NOT NULL UNIQUE
);
```

Loading Data

Adding Data to a Table

The INSERT statement can be used to **add data to an empty table, or to append data** to a table that already contains data.

Method	Description	Syntax
1	One clause per row using positional values	**INSERT INTO** *table-name* <*(column list)*> **VALUES** (*value,value,...*)**;**
2	A query returning multiple rows based on positional values	**INSERT INTO** *table-name* <(*column list*)> **SELECT** *columns* **FROM** *table-name*;

Method 1: One clause per row using positional values

INSERT INTO table-name <(column list)>
VALUES (value,value,...);

```
insert into discounts
(Start_Date,End_Date, Product_ID, Discount)
   values ('01MAR2013'd,'15MAR2013'd,
           230100300006,.33)
   values ('16MAR2013'd,'31MAR2013'd,
           230100600018,.15);
```

```
INSERT INTO practice.people
(id, name)
VALUES
(3000,'Jack Smith')
```

The order of the columns in the column list is independent of the order of the columns in the table.

Method 2: A query returning multiple rows based on positional values

INSERT INTO *table-name* <(*column list*)>

SELECT *columns* **FROM** *table-name*;

```
proc sql;
insert into discounts
 (Product_ID,Discount,Start_Date,End_Date)
   select distinct Product_ID,.35,
          '01MAR2013'd,'31MAR2013'd
      from jupiter.Product_Dim
      where Supplier_Name contains
           'Pro Sportswear Inc';
quit;
```

```
INSERT INTO exercise.countries2 SELECT
id,name FROM exercise.countries;
```

Deleting

DELETE Statement

DELETE FROM *table-name*
WHERE *condition*;

If you omit a WHERE clause, then the DELETE statement deletes all the rows from the specified table

Views

What is a View?

- **Virtual table** based on the result-set of an SQL statement: **stored query**
- Contains rows and columns, just **like a real table**
- Contains **no actual data**
- **Extracts underlying data** each time it is used and accesses the most current data
- Can be **referenced** in queries in the same way as a data table

CREATE VIEW *view-name*
AS SELECT …;

Creating a View

CREATE VIEW *view-name*
AS SELECT …;

```
create view jupiter.Tom_Zhou as
   select Employee_Name as Name format=$25.0,
          Job_Title as Title format=$15.0,
          Salary "Annual Salary"
format=comma10.2,
          int((today()-
Employee_Hire_Date)/365.25)
             as YOS 'Years of Service'
      from jupiter.employee_addresses as a,
           jupiter.employee_payroll as p,
           jupiter.employee_organization as o
      where a.Employee_ID=p.Employee_ID and
            o.Employee_ID=p.Employee_ID and
            Manager_ID=120102;
```

```
CREATE TEMP VIEW v_movies
AS
select p.name, count(*) as count_movies
from people p, people_movies pm
where p.id=pm.person_id
group by 1;
```

Views: **Advantages**

You can use views to do the following:

- **avoid** storing copies of large tables.
- **avoid** a frequent refresh of table copies. When the underlying data changes, a view surfaces the most current data.
- **pull** together data from multiple database tables and multiple libraries or databases.
- **simplify** complex queries.
- **prevent** other users from inadvertently altering the query code.

Views: Disadvantages

- Because views access the most current data in changing tables, the **results might be different each time you access the view**.
- Views can require significant resources each time that they execute. With a view, you **save disk storage space at the cost of extra CPU and memory usage**.
- When accessing the same data several times in a program, use a table instead of a view. This ensures consistent results from one step to the next and can significantly reduce the resources that are required.