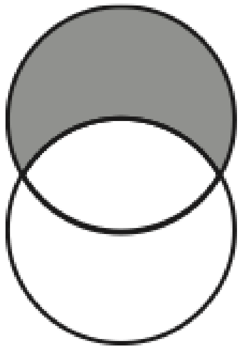
Biostats 597E

Week 4 - Introduction to SQL

Vertical Combining: EXCEPT

Similar to **UNION**, **EXCEPT** also combines two tables vertically. It produces rows that are in only the first query result.

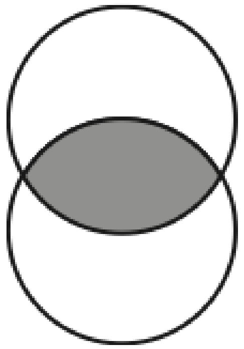


- Find countries in **oilprod** table but not in **oilrsrvs** table

```
select country from oilprod
except
select country from oilrsrvs
```

Vertical Combining: INTERSECT

INTERSECT produces rows that belong to both query results



- Find countries in both **oilprod** table and **oilrsrvs** table

```
select country from oilprod
intersect
select country from oilrsrvs
```

Save Query Results to Table

We can save results returned from a query to a table in the database. We use **CREATE TABLE <TABLE NAME> AS**

Example: For each continent, find the country with most population and save as a new table call toppopulation

```
create table toppopulation as
select t1.continent, t1.name
  from countries t1 inner join
    (select continent, max(population) as max_population
    from countries group by continent ) t2
    on t1.continent = t2.continent
  where t1.population = t2.max_population
```

Create A New Table

We can create new table from scratch with syntax

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

Example

```
CREATE TABLE people
(
ID INT,
First CHAR(50),
Last CHAR(50),
```

Insert/Delete Data

```
INSERT INTO table_name
VALUES (value1,value2,value3,...);
```

```
DELETE FROM table_name
WHERE some_column=some_value;
```

Example

```
INSERT INTO people
VALUEs (1, "John", "Smith", 30), (2, "Joe", "Doe", 60)
```

```
DELETE FROM people
where first = "John"
```

Delete Table

We can delete a table using **DROP**

Example

```
DROP table people;
```

SQL Views

- SQL View is a virtual table
- A view contains rows and columns just like real table. The fields in a view are fields from one or more real tables in the database
- The SQL view can be just used as a real table
- A view always shows up-to-date data
- A view is created just like a new table

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Compare View and Table

First we create two source tables

```
create table one (X INT, Y INT);
insert into one values (1, 2), (2, 3);
create table two (X INT, Y INT);
insert into two values (2, 5), (3, 6), (4, 9)
```

- Create a table joining one and two

  ```
  create table three_table as
  select one.*, two.Y as Y2 from one, two where one.X = two.X
  ```

- Create a view joining one and two

  ```
  create view three_view as
  select one.*, two.Y as Y2 from one, two where one.X = two.X
  ```

Compare View and Table

three_table and three_view are same.

Now let's make some change to the source tables

```
insert into one values (3, 9)
```

Are three_table and three_view still same?

SQL in SAS

We can use **PROC SQL** in SAS to execute sql state

```
PROC SQL;
  <SQL STATEMENT>;
  <SQL STATEMENT>;
  ...
QUIT;
```

- We can specify SAS table, such as **WORK.CARS** as input for **FROM**
- We can specify multiple SQL statementes in one **PROC SQL**
- In general, we will save query results to a SAS table for further processing by other data steps or procedures

Examples

We use freely downloaded SAS Unversity edition.

- Use **AIR** data, find mean, min, max, sd of air for each year.

```
proc sql;
  create table year_sum as
    select year(date)as Year,
            mean(air) as mean_air,
            min(air) as min_air,
            max(air) as max_air,
            std(air) as sd_air
    from sashelp.air
    group by Year;
quit;
```

- This is actually a cleaner and more flexible approach to obtain the statistics than using **PROC MEANS**

Create Macro Variables

We can create macro varaibls using **PROC SQL**. The macro vaiables can be further used by data steps and procedures.

We use **into** in select clause.

Example: create macro variable for number of rows

```
PROC SQL noprint; select count(*) into :n from sashelp.air; quit;
%put there are &n rows in the data;
```

If there are multiple values, then we use **SEPARATED BY** to aggregate the values or select into multiple macro variables.

```
PROC SQL noprint;
  select distinct year(date) into :years separated by ","
  from sashelp.air; quit;
%put The years in data are &years;
```

Dictionary Tables and Views

We can dynamically retrieve meta information about the SAS tables such as what tables currently in the system, what variables are in each table.

- **Dictionary.Columns**: information about columns in tables
- **Dictionary.Tables**: information about tables
- For more tables and views, refer to http://www2.sas.com/proceedings/sugi30/070-30.pdf

Example: Find variable information for table sashelp.cars?

```
proc sql;
select *
from dictionary.columns
where upcase(libname) = "SASHELP" and upcase(memname) = "CARS";
quit;
```

Example

We can play some trick. **Task**: For each numeric variable in cars, multiply each of them by 10.

```
proc sql;
    select name || "= " || name || "*10;" into :code separated by ""
    from dictionary.columns
    where upcase(libname) = "SASHELP"
        and upcase(memname) = "CARS"
        and type = "num";
quit;

data newdata;
    set sashelp.cars;
    &code;
run;
```