Proc SQL in SAS

By: Meisam Hejazinia
Class of Data Analysis of Professor B.P.S. Murthi @ UTD
Winter 2013

Normalization rules

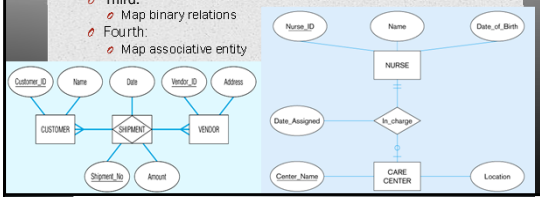
- **First Normal Form (1NF):** sets the very basic rules for an organized database
  - Eliminate duplicative columns from the same table
  - Create separate tables for each group of related data and identify each row with a unique column (the primary key)
- **Second Normal Form (2NF)**
  - Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
  - Create relationships between these new tables and their predecessors through the use of foreign keys.
- **Third Normal Form (3NF)**
  - Remove columns that are not dependent upon the primary key.

Relational Databases

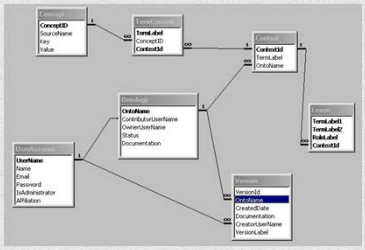
- Data is represented in the form of tables, and the model has 3 components
  - Data structure:
    - data are organised in the form of tables with rows and columns
  - Data manipulation:
    - powerful operations (using the SQL language) are used to manipulate data stored in the relations
  - Data integrity:
    - facilities are included to specify business rules that maintain the integrity of data when they are manipulated

Entity Relationship Diagram

- Steps:
  - First: map regular entities: transform to relation
  - Second: Map weak entities
    - Can not be identified by its attribute
    - Needs foreign key in conjunction with primary key as a primary key
  - Third:
    - Map binary relations
  - Fourth:
    - Map associative entity



Example



SQL

- SQL: Structured Query Language
- Attempts to allow humans to ask questions of data sources using natural language
- retrieves and updates data in tables and views based on those tables
- The SAS System's SQL procedure enables you to
  - retrieve and manipulate data that are stored in tables or views.
  - create tables, views, and indexes on columns in tables.
  - create SAS macro variables that contain values from rows in a query's result.
  - add or modify the data values in a table's columns or insert and delete rows. You can also modify the table itself by adding, modifying, or dropping columns.
  - send DBMS-specific SQL statements to a database management system (DBMS) and to retrieve DBMS data.

Format of SQL command

- **SELECT**
  - **FROM**
    - **WHERE**
      - **GROUP BY**
    - **HAVING**

Example : Select & Aggregate Functions

```
/*Step4: Calculate B stat(degree, duration, amount)*/
PROC SQL;
CREATE TABLE TLCM.NetStatB AS
SELECT Bnumber,COUNT(*) as degree,
      SUM(duration) as duration,
       SUM(amount) as Bamount,month AS Month
       FROM TLCM.CalldetailJTB
   GROUP BY Bnumber,month;
QUIT;
```

Example: Simple selection, create table

```
PROC IMPORT OUT= TLCM.STATUS9104
        DATAFILE= "C:\..\9104status.csv"
        DBMS=CSV REPLACE;
    GETNAMES=YES;
    DATAROW=2;
RUN;
PROC SQL;
CREATE TABLE TLCM.CalldetailAAge AS
SELECT Anumber,Bnumber, APChurncall,BPChurncall,AGE AS AAGE
            FROM TLCM.Calldetailjtb A, TLCM.Age B
                    WHERE A.Anumber=B.MSISDN
           /*ORDER BY calldate */
;
QUIT;
```

Another Aggregate Query

```
/*
Add on group sizes to data sets...
*/

PROC SQL PRINT;

SELECT
    sqldemo.*,
    Count(id) AS Frequency
FROM sqldemo

GROUP BY ID   /* Does the COUNT
function BY GROUP */
;

QUIT;
```

id	status	visit	Frequency
Pat1	A	6	10
Pat1	B	10	10
Pat1	A	7	10
Pat1	B	3	10
Pat1	B	9	10
Pat1	B	4	10
Pat1	A	5	10
Pat1	A	1	10
Pat1	B	2	10
Pat1	B	8	10
Pat2	A	4	8
Pat2	B	5	8
Pat2	A	3	8
Pat2	A	2	8
Pat2	B	1	8
Pat2	B	8	8
Pat2	B	7	8
Pat2	A	6	8

Example: Query

```
/**
Find those records with 2 or more 'B'
status and print the counts
******/
PROC SQL PRINT;
SELECT
    id,
    status,
    count(*) as freq
FROM sqldemo
WHERE status = 'B'

GROUP BY id,status

HAVING (Count(*))>= 2 ;
QUIT;
```

id	status	freq
Pat1	B	6
Pat2	B	4

Multiple Table Examples

```
/* Find out the number of times
something changed status
between concsecutive visits
STEP 1    */

PROC SQL FEEDBACK STIMER PRINT  ;

SELECT

c1.id as c1id,
c2.status as newstatus,
c1.status as oldstatus,
c2.visit  as newvisit,
c1.visit  as oldvisit,
count(c1.id) as idcount

FROM sqldemo AS c1,
     sqldemo AS c2

WHERE c1.id = c2.id AND
      /*c1.status ne c2.status AND
      c2.visit - c1.visit = 1 */
      c2.visit > c1.visit
GROUP BY
      c1.id
       ;
quit;    * end of PROC SQL ;
```

c1id	newstatus	oldstatus	newvisit	oldvisit	idcount
Pat1	B	B	10	3	45
Pat1	B	A	8	7	45
Pat1	A	A	7	1	45
Pat1	B	B	8	4	45
Pat1	A	A	7	6	45
Pat1	A	B	6	3	45
Pat1	B	B	10	8	45
Pat1	B	B	10	4	45
Pat1	A	B	6	4	45
Pat1	A	B	7	2	45
Pat1	B	A	4	1	45
Pat1	B	A	9	5	45
Pat1	B	A	10	1	45
Pat2	B	A	8	3	28
Pat2	A	B	3	1	28
Pat2	A	B	2	1	28
Pat2	B	B	8	7	28
Pat2	B	A	8	6	28
Pat2	B	A	7	6	28
Pat2	B	B	8	5	28
Pat2	B	B	7	5	28
Pat2	A	B	6	5	28
Pat2	B	A	8	4	28
Pat2	B	A	7	4	28
Pat2	A	A	6	4	28
Pat2	B	A	5	4	28

Example2 – Nested and set relation of queries

```
PROC SQL;
CREATE TABLE TLCM.CalldetailJT AS
Select * FROM(
  SELECT *,0 AS APChurnCall
    FROM TLCM.DetailJalali A
      WHERE A.Anumber NOT IN
   (SELECT A.Anumber
     FROM TLCM.DetailJalali A, TLCM.Status9107 B
         WHERE A.Anumber=B.MSISDN AND B.churn=1 AND B.first=0 AND
           B.second=0 AND B.third=0 AND NOT A.month>4)
    OUTER UNION CORR
          SELECT A.*, 1 as APChurnCall
            FROM TLCM.DetailJalali A, TLCM.Status9107 B
               WHERE A.Anumber=B.MSISDN AND B.churn=1 AND B.first=0
                  AND  B.second=0 AND B.third=0 AND NOT A.month>4)
           /*ORDER BY calldate */ ;
QUIT;
```

Outer Join of tables options

Example of aggregate Query

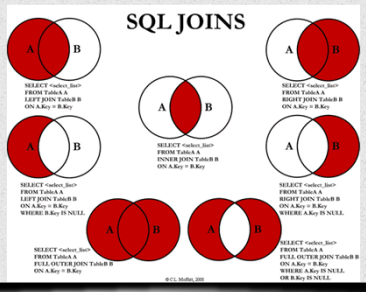
```
PROC SQL;
CREATE TABLE HIGHBPP2 AS
SELECT PATIENT, COUNT(PATIENT) AS N,
DATE FORMAT=DATE7., MAX(BPD) AS BPDHIGH
FROM VITALS
WHERE PATIENT IN (101 102 103)
GROUP BY PATIENT
HAVING BPD = CALCULATED BPDHIGH
ORDER BY CALCULATED BPDHIGH;
QUIT;
```

Join Examples

```
PROC SQL;
CREATE TABLE BOTH AS
SELECT A.PATIENT,
A.DATE FORMAT=DATE7. AS DATE,
A.PULSE,
B.MED, B.DOSES, B.AMT FORMAT=4.1
FROM VITALS A INNER JOIN DOSING B
ON (A.PATIENT = B.PATIENT) AND
(A.DATE = B.DATE)
ORDER BY PATIENT, DATE;
QUIT;
```

```
PROC SQL;
CREATE TABLE LEFT AS
SELECT A.PATIENT,
A.DATE FORMAT=DATE7. AS DATE,
A.PULSE,
B.MED, B.DOSES, B.AMT FORMAT=4.1
FROM VITALS A LEFT JOIN DOSING B
ON (A.PATIENT = B.PATIENT) AND
(A.DATE = B.DATE)
ORDER BY PATIENT, DATE;
QUIT;
```

Type of Joins



Joint examples

```
PROC SQL;
CREATE TABLE FULL AS
SELECT A.PATIENT,
A.DATE FORMAT=DATE7. AS DATE,
A.PULSE,
B.MED, B.DOSES, B.AMT FORMAT=4.1
FROM VITALS A FULL JOIN DOSING B
ON (A.PATIENT = B.PATIENT) AND
(A.DATE = B.DATE)
ORDER BY PATIENT, DATE;
QUIT;
```

Set operators

set-operator is one of the following:

INTERSECT <CORRESPONDING> <ALL>

OUTER UNION <CORRESPONDING>

UNION <CORRESPONDING> <ALL>

EXCEPT <CORRESPONDING> <ALL>

```
proc sql;
title 'ME1 and ME2: OUTER UNION';
select * from me1
outer union
select * from me2;
```

```
proc sql;
title 'ME1 and ME2: OUTER
UNION CORRESPONDING';
select * from me1
outer union corr
select * from me2;
```

```
proc sql;
title 'Flights from IN_USA and OUT_USA';
select * from in_usa
intersect
select * from out_usa;
```

ALL: includes duplicate

Union: produces all unique rows from both queries.

Except: produces rows that are part of the first query only.

Outer Union: contains all the rows produced by the first table-expression followed by all the rows produced by the second table-expression

Set operator cont.

- CORR causes PROC SQL to match the columns in table-expressions **by name** and not by ordinal position
- For example, when performing a set operation on two table-expressions, PROC SQL matches the first specified column-name (listed in the SELECT clause) from one table-expression with the first specified column-name from the other.
- If CORR is omitted, PROC SQL matches the columns by ordinal position.

Conclusion

- SQL is:
  - Good and useful
  - less complex than data step
  - More efficient both in term of coding and system optimization
- Let's use it