SAS-seminar
Proc SQL, the pass-through facility

How to make your data processing someone else's problem

Outline

What is SQL and what is a database

Quick introduction to Proc SQL

The pass-through facility

Examples…

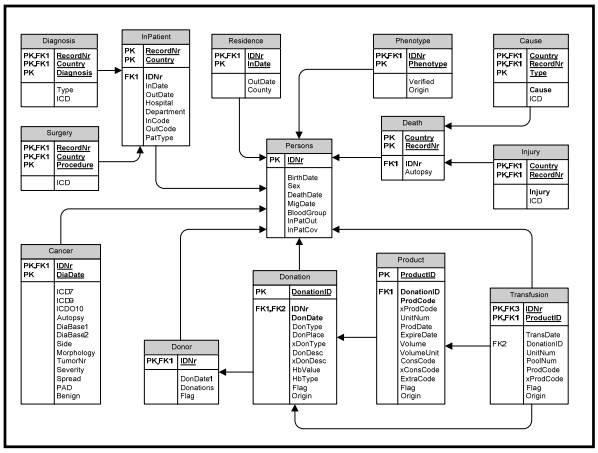
What is SQL?

- SQL = Structured Query Language.
- Developed by IBM in the 1970s for querying, modifying and updating relational databases
- Adopted and standardized by ANSI and ISO during the 1980s
- Used with (almost) all RDBMS (Relational Data Base Management System) e.g. Oracle, DB2, Access, MySQL etc.
- Most RDBMS also offer enhancements to ANSI SQL

What is a database?

"A database is a collection of logically related data and a description of this data, designed to meet the need for information in an organization."

For me, a database is just a collection of tables…



What is Proc SQL?

Proc SQL is a SAS implementation of SQL

It can be conceptualized as a combination of the data step, proc summary and proc sort, all at once

Using Proc SQL you can execute SQL queries on a remote database server

Proc SQL intro – terminology

SAS Data step	Proc SQL
Dataset	Table
Variable	Column
Observation	Row
Merge	Join
Append	Union

Proc SQL syntax

From the name follows that it is extremely structured

It is highly scalable and can be used for a multitude of different tasks – anything from simple counts to complicated joins of multiple tables and sub queries

It typically cannot do anything that ordinary SAS can't…

Proc SQL syntax – selecting

```
proc sql;
  create table tablename as
  select [distinct]
      column1,
      column2,
      [*], …
  from library.table
  where expression
  order by column1;
  quit;
* = all columns
```

data *tablename*;

keep *column1*, *column2*;

set *library.table*;

where *expression*;

proc sort;
by *column1*;
run;

SQL example 1 – selecting

```
proc sql;
  create table women as
  select
     *
  from cblood.persons
  where sex = 2
  order by birthdate;
  quit;
```

* = all variables

SQL example 2 – selecting

```
proc sql;
  create table patients as
  select distinct
     idnr
  from cblood.transfusion;
  quit;
```

distinct ≈ no duplicate values

Proc SQL syntax – modifying

```
proc sql;
  create table tablename as
  select
     function(column1) as new1,
     column2 [+|-|*|/] column3 as new2
  from library.table;
  quit;
```

SQL example 3 – modifying

```
proc sql;
  create table dead as
  select
      idnr,
      (deathdate-birthdate)/365.24 as age
  from cblood.persons
  where not deathdate is null;
  quit;
```


Proc SQL syntax – summarizing

```
proc sql;
  create table tablename as
  select
      summary_function(column1) as new1
  from library.table
  group by byvariable
  having group_conditions;
  quit;
```


SQL example 4 – summarizing

```
proc sql;
  create table cancers as
  select
      idnr,
      count(*) as cancers
  from cblood.cancer
  group by idnr;
  quit;
```

The count() function returns the number of rows as defined by the group statement

SQL example 5 – summarizing

```
proc sql;
  create table unlucky_few as
  select
      idnr,
      count(*) as count
  from cblood.cancer
  group by idnr
  having count(*) > 10;
  quit;
```

Returns 156 unlucky souls…

… with 10 or more cancers

Proc SQL syntax – joining tables

```
proc sql;
  create table tablename as
  select
      alias1.column1,
      alias2.column1
  from library.table1 as alias1
      [inner|left|right|full] join
      library.table2 as alias2
  on join clause
  quit;
```

Specifies how the two tables are joined

Sort of like the by statement in a SAS merge

Cartesian products…

```
proc sql;
  create table tablename as
  select
      alias1.column1,
      alias2.column1
  from library.table1 as alias1,
      library.table2 as alias2
  quit;
```

Failing to specify a join clause will create a cartesian product.

i.e. every row in table1 will be matched to every row in table2…

SQL example 6 – joining

```
proc sql;
  create table unlucky_donors as
  select
      a.idnr
  from cblood.donor as a
      inner join cblood.cancer as b
  on a.idnr=b.idnr
  group by a.idnr
  having count(*) > 10;
  quit;
```

Returns only 4 unlucky donors


```
proc sql;
Create view table1 as
  select
      coalesce(a.idnr,b.idnr) as idnr
  from cblood.donor as a
      full outer join cblood.cancer as b
  on a.idnr=b.idnr
  group by a.idnr
  having count(*) > 10;
  quit;
```


SQL example – select controls

```
proc sql outobs=10;
create table controls as
    select
        a.casenr label='',
        a.exit as indexdate,
        b.recipient label='',
        ranuni(2) as randomnr
    from cases a left join available_controls b
    on a.entry-180 le b.entry le a.entry+180
        and a.country=b.country
        and a.county=b.county
        and a.bloodgroup=b.bloodgroup
        and a.age=b.age
        and b.entry < a.exit < b.exit
    where a.county ne .
        and b.county ne .
    order by a.casenr, calculated randomnr;
    quit;
```


What is the pass-through facility?

The Proc SQL pass-through facility is a SAS tool for directly with a database server/RDBMS

The Pass-through facility can:

- Efficiently run complicated queries
- Use native RDBMS functions
- Execute RDBMS-specific commands
- You can use RDBMS indices

When do we pass through?

When Proc SQL doesn't get it…

```
proc sql;
  create table yearly_counts
  as
  select
      year(dondate) as year,
      count(*) as count
  from cblood.donation
  group by year(dondate);
  quit;
```

Not a standard SQL function, so SAS downloads the whole dataset and performs the query locally

~12 minutes

So, how do we "Pass-through"?

```
proc sql;
  connect to DBMS-name (connection statements);
  select
      column list
  from connection to DBMS-name
      (
      DBMS-query
      )
  disconnect from DBMS-name;
  quit;
```

DBMS specific query

SQL Example 7 – Pass-through

```
proc sql;
connect to oracle (user=? path=? password=?);
select
    *
from connection to oracle
    (
    select
          extract(year from dondate) as year,
          count(*) as count
    from cblood2.donation
    group by extract(year from dondate)
    );
disconnect from oracle;
quit;
```

The Oracle version of the year() function

~30 seconds

Pass-through galore!

```
proc sql;
connect to oracle (user=? path=? password=?);
select
    *
from connection to oracle
    (
    select
          a.country,
          b.sex,
          trunc(months_between(a.transdate,b.birthdate)/12) as age,
          count(*) as count
    from cblood2.transfusion a inner join cblood2.persons b
    on a.idnr=b.idnr and b.birthdate <= a.transdate
    where extract(year from a.transdate) between 1968 and 2002
          and instr(coalesce(b.flag,' '),'ID')=0
    group by
          a.country,
          b.sex,
          trunc(months_between(a.transdate,b.birthdate)/12)
    );
quit;
```

Returns age in exact years

Only individuals with a valid ID…

Considerations for Pass-through

- The SAS implementation of Proc SQL is anything but standard, so what works with SAS may not work with Oracle…
- Oracle has a very odd way of handling missing values

Conclusions…

Proc SQL in itself is a very powerful tool

…coupled with a powerful database server, Proc SQL can save you a lot of time