SAS SQL #1
SAS Structured Query Language, An Introduction

JC Wang

Western Michigan University
Department of Statistics
Outline

1 SQL—An Introduction
   - What is SQL

2 Introduction to PROC SQL
   - SAS SQL
   - Query a Table
   - Sorting Query Output
What is SQL Language

Structured Query Language (SQL), a standardized, widely used language that retrieves / updates data in relational databases.

Relation = concept similar to concept of a set in mathematics

Relations are represented physically as two-dimensional tables (arranged in rows and columns).
Outline

1. SQL—An Introduction
   - What is SQL

2. Introduction to PROC SQL
   - SAS SQL
   - Query a Table
   - Sorting Query Output
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
What PROC SQL Does

PROC SQL = SAS implementation of Structured Query Language. It can

- generate reports
- generate summary statistics
- retrieve data from tables or views
- combine data from tables or views
- create tables, views, and indexes
- update data values in PROC SQL tables
- update / retrieve data from database management system (DBMS) tables
- modify PROC SQL table by adding, modifying, or dropping columns.
Comparison of Terminology

- **SQL tables** = SAS data file = a file
- **rows** = observations = records
- **columns** = variables = fields
- **DBMS tables** = tables from other vendors which can be accessed through SAS/ACCESS

- Queries retrieve data from a table, view, or DBMS with results consisting of rows and columns from a table.
- **SQL view** contains a stored query (instruction only), i.e., a virtual table
- **SAS PROC SQL null values** = missing values.
Comparison of Terminology

- **SQL tables** = SAS data file = a file
- **rows** = observations = records
- **columns** = variables = fields
- **DBMS tables** = tables from other vendors which can be accessed through SAS/ACCESS
- **Queries** retrieve data from a table, view, or DBMS with results consisting of rows and columns from a table.
  - SQL view contains a stored query (instruction only), i.e., a virtual table
  - SAS PROC SQL **null values** = missing values.
Comparison of Terminology

- SQL **tables** = SAS data file = a file
- **rows** = observations = records
- **columns** = variables = fields
- DBMS **tables** = tables from other vendors which can be accessed through SAS/ACCESS
- **Queries** retrieve data from a table, view, or DBMS with results consisting of rows and columns from a table.
- SQL **view** contains a stored query (instruction only), i.e., a virtual table
- SAS PROC SQL **null values** = missing values.
Comparison of Terminology

- **SQL tables** = SAS data file = a file
- **rows** = observations = records
- **columns** = variables = fields
- **DBMS tables** = tables from other vendors which can be accessed through SAS/ACCESS
- **Queries** retrieve data from a table, view, or DBMS with results consisting of rows and columns from a table.
- **SQL view** contains a stored query (instruction only), i.e., a virtual table
- **SAS PROC SQL null values** = missing values.
Syntax of PROC SQL

basic use

PROC SQL <options>;
query-1;
query-2;
...more queries...
QUIT;

- a query = a run block. Upon submission, SAS executes the query.
- End PROC SQL with a QUIT since it’s interactive.
Syntax of PROC SQL
basic use

PROC SQL <options>;
query-1;
query-2;
...more queries...
QUIT;

- a query = a run block. Upon submission, SAS executes the query.
- End PROC SQL with a QUIT since it’s interactive.
Syntax of a Query
from single table

```
SELECT  input-columns
FROM    table
WHERE   cond-to-subset-input-rows
GROUP BY column
HAVING  cond-to-subset-groups
ORDER BY columns;
```

- It’s a single statement in this order of clauses.
- SELECT and FROM are required.
Syntax of a Query
from single table

```
SELECT input-columns
FROM table
WHERE cond-to-subset-input-rows
GROUP BY column
HAVING cond-to-subset-groups
ORDER BY columns;
```

- It’s a single statement in this order of clauses.
- SELECT and FROM are required.
SELECT and FROM Clauses

**SELECT** input-columns

**FROM** table;

- **input-columns**: a comma-separated ordered list of columns or the wild-card * (= all columns in table)
- **table**: one-word table (SAS data set) name or two-word name `libref.sas-data-set-name` (usually the latter)
- use **SELECT DISTINCT** a-table-column to query only unique values of the table column
SELECT and FROM Clauses

**SELECT** input-columns
**FROM** table;

- **input-columns**: a comma-separated ordered list of columns or the wild-card * (= all columns in table)
- **table**: one-word table (SAS data set) name or two-word name `libref.sas-data-set-name` (usually the latter)
- use **SELECT DISTINCT** a-table-column to query only unique values of the table column
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be
- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Input Columns in SELECT Clause

an input column can be

- a table column
- a quoted text string
- a calculated column
- a column created by a simple CASE expression
- a column created by a CASE-Operand expression
- a column created by COALESCE function of columns
Dress Up Input Columns with Attributes

\textit{input-column} \textless \textit{attributes}\textgreater

\textbf{where attributes} are blank-separated column attributes of

- \textsc{Format}=
- \textsc{Informat}=
- \textsc{Label}='text-string' (or simply 'text-string')
- \textsc{Length}=

Use of \textsc{Label}='\#' suppresses the column header.

See example sql01.sas.
Dress Up Input Columns with Attributes

\[ \text{input-column } <\text{attributes}> \]
where \text{attributes} are blank-separated column attributes of

- FORMAT=
- INFORMAT=
- LABEL='text-string' (or simply 'text-string')
- LENGTH=

Use of LABEL='#' suppresses the column header.
See example sql01.sas.
Dress Up Input Columns with Attributes

\[
\text{input-column} \ <\text{attributes}>
\]

where attributes are blank-separated column attributes of

- FORMAT=
- INFORMAT=
- LABEL='text-string' (or simply 'text-string')
- LENGTH=

Use of LABEL='#' suppresses the column header.

See example sql01.sas.
Dress Up Input Columns with Attributes

```
input-column <attributes>
where attributes are blank-separated column attributes of

  FORMAT=
  INFORMAT=
  LABEL='text-string' (or simply 'text-string')
  LENGTH=

Use of LABEL=' #' suppresses the column header.
See example sql01.sas.
```
Using Column Alias

```sql
input-column AS column-alias
where
```

- `column-alias` is a valid SAS name used as an alias
- when referencing it in subsequent calculated column(s) in the calculation, use the alias

See example `sql01.sas`.
Using Column Alias

`input-column AS column.alias`

where

- `column.alias` is a valid SAS name used as an alias
- when referencing it in subsequent calculated column(s) in the calculation, use the alias

See example `sql01.sas`.
Using Column Alias

\[ \text{input-column AS column-alias} \]

where

- \text{column-alias} \text{ is a valid SAS name used as an alias}
- when referencing it in subsequent calculated column(s) in the calculation, use the alias

See example sql01.sas.
Adding Text to Output

Codes

data test;
  do r = rank('A') to rank('Z');
    c = byte(r);
    output;
  end;
run;
proc sql;
  select 'Character' format=$9.,
          r label='#' format=2.,
          'is', c '#'
    from test;
quit;

Output

Character  65 is A
Character  66 is B
Character  67 is C
Character  68 is D
more output lines here
Character  90 is Z
Adding Text to Output

Codes

data test;
  do r = rank('A') to rank('Z');
    c = byte(r);
    output;
  end;
run;
proc sql;
  select 'Character' format=$9.,
         r label='#' format=2.,
         'is', c '#'
  from test;
quit;

Output
---------------------
Character 65 is A
Character 66 is B
Character 67 is C
Character 68 is D
Character 90 is Z
### Adding Text to Output

#### Codes

data test;
  do r = rank('A') to rank('Z');
    c = byte(r);
    output;
  end;
run;

proc sql;
  select 'Character' format=$9.,
     r label='#' format=2.,
     'is', c '#'
  from test;
quit;

#### Output

Character 65 is A
Character 66 is B
Character 67 is C
Character 68 is D
more output lines here
Character 90 is Z
Calculated Column

- created by
  \[ \text{arithmetic-expression} \ <\text{AS alias}> \ <\text{attributes}> \]
  where \text{alias} is required if it is referenced in subsequent calculated column(s)

- referencing it by
  \text{CALCULATED alias}

See example sql01.sas.
Calculated Column

- created by
  \[ \text{arithmetic-expression} \ <\text{AS alias}> \ <\text{attributes}> \]
  where \text{alias} is required if it is referenced in subsequent calculated column(s)
- referencing it by
  \text{CALCULATED alias}

See example sql01.sas.
Calculated Column

- created by
  \[\text{arithmetic-expression} \ AS \ \text{alias} \ \text{attributes}\]
  where \text{alias} is required if it is referenced in subsequent calculated column(s)

- referencing it by
  \text{CALCULATED} \ \text{alias}

See example \text{sq101.sas}.
Column Created By CASE Expression

created by

CASE
  WHEN cond-1 THEN expr-1
  more WHEN-THEN clauses as needed
  ELSE expr-2
END <AS alias> <attributes>

referencing it by

CALCULATED alias

See example sql01.sas.
Column Created By CASE Expression

- created by
  ```
  CASE
    WHEN cond-1 THEN expr-1
    more WHEN-THEN clauses as needed
    ELSE expr-2
  END <AS alias> <attributes>
  ```

- referencing it by
  ```
  CALCULATED alias
  ```

See example sql01.sas.
Column Created By CASE Expression

- created by
  CASE
    WHEN cond-1 THEN expr-1
    more WHEN-THEN clauses as needed
    ELSE expr-2
  END <AS alias> <attributes>

- referencing it by
  CALCULATED alias

See example sql01.sas.
Column Created By CASE-Operand Expression

created by

```
CASE  operand
    WHEN  value-1  THEN  expr-1
    more  WHEN-THEN  clauses  as  needed
    ELSE  expr-2
END  <AS  alias>  <attributes>
```

referencing it by

```
CALCULATED  alias
```

See example sql01.sas.
Column Created By CASE-Operand Expression

- created by
  **CASE**  *operand*
    **WHEN**  *value-1*  **THEN**  *expr-1*
    more **WHEN-THEN** clauses as needed
  **ELSE**  *expr-2*
  **END**  
  *<AS alias>  <attributes>*

- referencing it by
  **CALCULATED**  *alias*

See example *sql01.sas*. 
Column Created By CASE-Operand Expression

- created by
  
  ```
  CASE  operand
    WHEN  value-1  THEN  expr-1
    more  WHEN-THEN  clauses  as  needed
    ELSE  expr-2
  END  <AS  alias>  <attributes>
  ```

- referencing it by
  
  ```
  CALCULATED  alias
  ```

See example `sql01.sas`. 
Column Created By COALESCE Function

Usage:

```
COALESCE(col-1, col-2, ..., <AS alias> <attributes>)
```

returns first non-missing value of the columns; returns missing value if all missing. For example,

```
select Name,
    coalesce(LowPoint, '*Not Available*') as LowPoint
from mylib.continents;

select Name,
    case
        when LowPoint is missing then '*Not Available*_'
        else LowPoint
    end as LowPoint
from mylib.continents;
```
Subset Input Table Using WHERE Clause

```
SELECT input-columns
FROM table
WHERE cond;
```

where `cond` is expression that makes use of:

- table columns and/or input columns and with comparison operators (symbols or mnemonics), logical operators (symbols or mnemonics)
- special operators including BETWEEN-AND, CONTAINS, IN, IS NULL/IS MISSING, LIKE, and `=`
- truncated string comparison operators (3-letter mnemonics formed by appending 2-letter comparison operators)
- special operators linking subquery: ANY `(subquery)`, ALL `(subquery)`, and EXISTS `(subquery)`
Subset Input Table Using WHERE Clause

```sql
SELECT input-columns
FROM table
WHERE cond;
```

where `cond` is expression that makes use of:

- table columns and/or input columns and with comparison operators (symbols or mnemonics), logical operators (symbols or mnemonics)
- special operators including BETWEEN-AND, CONTAINS, IN, IS NULL/IS MISSING, LIKE, and =*
- truncated string comparison operators (3-letter mnemonics formed by appending 2-letter comparison operators)
- special operators linking subquery: ANY (subquery), ALL (subquery), and EXISTS (subquery)
Subset Input Table Using WHERE Clause

```sql
SELECT input-columns
FROM table
WHERE cond;
```

where `cond` is expression that makes use of

- table columns and/or input columns and with comparison operators (symbols or mnemonics), logical operators (symbols or mnemonics)
- special operators including BETWEEN-AND, CONTAINS, IN, IS NULL/IS MISSING, LIKE, and =*
- truncated string comparison operators (3-letter mnemonics formed by appending 2-letter comparison operators)
- special operators linking subquery: `ANY (subquery)`, `ALL (subquery)`, and `EXISTS (subquery)`
Subset Input Table Using WHERE Clause

**SELECT**  input-columns
**FROM**  table
**WHERE**  cond;

where  cond  is expression that makes use of

- table columns and/or input columns and with comparison operators (symbols or mnemonics), logical operators (symbols or mnemonics)
- special operators including BETWEEN-AND, CONTAINS, IN, IS NULL/IS MISSING, LIKE, and =*
- truncated string comparison operators (3-letter mnemonics formed by appending 2-letter comparison operators)
- special operators linking subquery:  ANY (subquery), ALL (subquery), and EXISTS (subquery)
Subset Input Table Using WHERE Clause

```sql
SELECT input-columns
FROM table
WHERE cond;
```

where `cond` is expression that makes use of

- table columns and/or input columns and with comparison operators (symbols or mnemonics), logical operators (symbols or mnemonics)
- special operators including BETWEEN-AND, CONTAINS, IN, IS NULL/IS MISSING, LIKE, and =*
- truncated string comparison operators (3-letter mnemonics formed by appending 2-letter comparison operators)
- special operators linking subquery: ANY (subquery), ALL (subquery), and EXISTS (subquery)
Summarizing Data
using aggregate functions

Aggregate functions (or summary functions) are used to combine columns. If single column is specified then the column statistic is produced; otherwise, row statistics are produced for the columns. Aggregate functions include

1. AVG/MEAN, MIN, MAX, SUM, SUMWGT
2. RANGE, STD, STDERR, CV
3. USS, CSS, VAR
4. COUNT, FREQ, N, NMISS
5. T, PRT
Summarizing Data
using aggregate functions

Aggregate functions (or summary functions) are used to combine columns. If single column is specified then the column statistic is produced; otherwise, row statistics are produced for the columns. Aggregate functions include

1. AVG/MEAN, MIN, MAX, SUM SUMWGT
2. RANGE, STD, STDERR, CV
3. USS, CSS, VAR
4. COUNT, FREQ, N, NMISS
5. T, PRT
Sorting Query Results

SELECT input-column
    FROM table
    ORDER BY sort-columns;

where columns is a comma-separated list of columns and each sort-column

• can be an input-column either by name or by its position (a number) in the input-column list
• can be a table column but not an input column
• if it’s a calculated column, do NOT use CALCULATED keyword
• use sort-column DESC for a sort-column to be in descending order
Sorting Query Results

\[ \text{SELECT } \text{input-column} \]
\[ \text{FROM } \text{table} \]
\[ \text{ORDER BY } \text{sort-columns}; \]

where \text{columns} is a comma-separated list of columns and each sort-column

- can be an input-column either by name or by its position (a number) in the input-column list
- can be a table column but not an input column
- if it’s a calculated column, do NOT use \text{CALCULATED} keyword
- use \text{sort-column DESC} for a sort-column to be in descending order
Sorting Query Results

```
SELECT input-column
    FROM table
    ORDER BY sort-columns;
```

where columns is a comma-separated list of columns and each sort-column

- can be an input-column either by name or by its position (a number) in the input-column list
- can be a table column but not an input column
- if it’s a calculated column, do NOT use CALCULATED keyword
- use sort-column DESC for a sort-column to be in descending order
Sorting Query Results

```
SELECT  input-column
    FROM  table
    ORDER BY  sort-columns;
```

where `columns` is a comma-separated list of columns and each `sort-column`

- can be an input-column either by name or by its position (a number) in the input-column list
- can be a table column but not an input column
- if it’s a calculated column, do NOT use `CALCULATED` keyword

- use `sort-column  DESC` for a sort-column to be in descending order
Sorting Query Results

```
SELECT input-column
    FROM table
    ORDER BY sort-columns;
```

where `columns` is a comma-separated list of columns and each sort-column:

- can be an input-column either by name or by its position (a number) in the input-column list
- can be a table column but not an input column
- if it’s a calculated column, do NOT use `CALCULATED` keyword
- use `sort-column DESC` for a sort-column to be in descending order