### Basic Queries

**PROC SQL**

```sql
PROC SQL <options>;
SELECT column-1 <,...column-n>
FROM input-table
<WHERE expression>
<GROUP BY col-name>
<HAVING expression>
<ORDER BY col-name> <DESC> <,...col-name>
```

**SQL Query Order of Execution**:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Retrieve data from a table</td>
</tr>
<tr>
<td>FROM</td>
<td>Choose and join tables</td>
</tr>
<tr>
<td>WHERE</td>
<td>Filter the data</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Aggregate the data</td>
</tr>
<tr>
<td>HAVING</td>
<td>Filter the aggregate data</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Sort the final data</td>
</tr>
</tbody>
</table>

### Modifying Columns

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE</td>
<td>CREATE TABLE table-name</td>
</tr>
<tr>
<td>DESCRIPTABLE</td>
<td>DESCRIPTABLE table-name</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>DROP TABLE table-name</td>
</tr>
</tbody>
</table>

### Managing Tables

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO</td>
<td>INSERT INTO table</td>
</tr>
<tr>
<td>SET</td>
<td>SET column-name=value</td>
</tr>
<tr>
<td>VALUES</td>
<td>VALUES (value,...value)</td>
</tr>
<tr>
<td>SELECT DISTINCT</td>
<td>SELECT DISTINCT col-name&lt;,...col-name</td>
</tr>
</tbody>
</table>
| WHERE        | WHERE col-name IN (value1, value2, ...)
| LIKE         | LIKE "_string%"                       |
| BETWEEN      | BETWEEN value AND value              |
| IS NULL      | IS NULL                                |
| DATE VALUE   | DATE VALUE                            |
| TIME VALUE   | TIME VALUE                            |
| DATETIME VALUE | DATETIME VALUE "<01JAN201914:45:35"|

### Modifying Rows

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO</td>
<td>INSERT INTO table</td>
</tr>
<tr>
<td>SET</td>
<td>SET column-name=value</td>
</tr>
<tr>
<td>VALUES</td>
<td>VALUES (value,...value)</td>
</tr>
<tr>
<td>SELECT</td>
<td>SELECT col-name&lt;,...col-name</td>
</tr>
</tbody>
</table>
| WHERE        | WHERE col-name IN (value1, value2, ...)
| LIKE         | LIKE "_string%"                       |
| BETWEEN      | BETWEEN value AND value              |
| IS NULL      | IS NULL                                |
| DATE VALUE   | DATE VALUE                            |
| TIME VALUE   | TIME VALUE                            |
| DATETIME VALUE | DATETIME VALUE "<01JAN201914:45:35"|

### Managing Views

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE VIEW</td>
<td>CREATE VIEW table-name AS query</td>
</tr>
<tr>
<td>DESCRIPTABLE</td>
<td>DESCRIPTABLE view-name-1</td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>DROP VIEW view-name-1</td>
</tr>
</tbody>
</table>

### Reemerging Summary Statistics

```sql
SELECT col-name, summary function(argument)
FROM input table;
```
**Joins Summary**

- **Inner Join**
  ```sql```
  ```
  SELECT <list>
  FROM table-A INNER JOIN table-B
  ON A.Key = B.Key;
  ```
  ```
  ```

- **Full Join**
  ```sql```
  ```
  SELECT <list>
  FROM table-A FULL JOIN table-B
  ON A.Key = B.Key;
  ```
  ```
  ```

- **Right Join**
  ```sql```
  ```
  SELECT <list>
  FROM table-A RIGHT JOIN table-B
  ON A.Key = B.Key;
  ```
  ```
  ```

- **Left Join**
  ```sql```
  ```
  SELECT <list>
  FROM table-A LEFT JOIN table-B
  ON A.Key = B.Key;
  ```
  ```
  ```

**Creating Macro Variables**

Storing a value in a macro variable using SQL:
```sql```
```
SELECT col-name-1, ..., col-name-n
  INTO: macrvar_1, ... macrvar-n
  FROM input-table;
```
```
Storing a list of values in a macro variable using SQL:
```sql```
```
SELECT col-name-1, ..., col-name-n
  INTO: macrvar_1 SEPARATED BY 'delimiter'
  FROM input-table;
```
```
Viewing the value of the macro variable in the SAS Log:
```sql```
```
%PUT &macrvar;
```
```

**Set Operators**

- **INTERSECT**
  ```sql```
  ```
  SELECT <list>
  FROM table-A INTERSECT
  SELECT <list>
  FROM table-B;
  ```
  ```

The INTERSECT operator selects unique rows that are common to both tables.

- **EXCEPT**
  ```sql```
  ```
  SELECT <list>
  FROM table-A EXCEPT
  SELECT <list>
  FROM table-B;
  ```
  ```

The EXCEPT operator selects unique rows from table A that are not found in table B.

- **UNION**
  ```sql```
  ```
  SELECT <list>
  FROM table-A UNION
  SELECT <list>
  FROM table-B;
  ```
  ```

The UNION operator selects unique rows from both tables.

- **OUTER UNION**
  ```sql```
  ```
  SELECT <list>
  FROM table-A OUTER UNION
  SELECT <list>
  FROM table-B;
  ```
  ```

The OUTER UNION operator selects all rows from both tables.

**Accessing DBMS Data**

The SQL pass-through facility enables you to code in the native DBMS SQL syntax and pass the query to the database.

```sql```
```
PROC SQL;
CONNECT TO DBMS-name <AS alias>
  (DBMS-connection-options);
SELECT col-name
  FROM CONNECTION TO DBMS-name|alias (dbms-query);
DISCONNECT FROM DBMS-name|alias;
QUIT;
```
Creating Macro Variables

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%GLOBAL macro-variable-1 ...macro-variable-n;</td>
<td>Creates a macro variable that is available during the execution of the entire SAS session.</td>
</tr>
<tr>
<td>%LET variable=value;</td>
<td>Creates a macro variable and assigns it a value.</td>
</tr>
<tr>
<td>%LOCAL macro-variable-1 ...macro-variable-n;</td>
<td>Creates a macro variable that is available only during the execution of the macro where it is defined.</td>
</tr>
</tbody>
</table>

Defining a Macro

%MACRO macro-name <(parameter-list)>;
    macro-text
%MEND <macro-name>;

The parameter-list can be:
<positional-parameter-1, ...positional-parameter-n> or
<keyword-1=value-1, ...keyword-n=value-n>

Calling a Macro

%macro-name
%macro-name(positional-parameter-1, ...positional-parameter-n)
%macro-name(keyword-1=value-1, ...keyword-n=value-n)

Referencing a Macro Variable

Use the name of the macro variable with an ampersand.
&macro-variable;

Macro Character Functions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%INDEX (source, string)</td>
<td>Determines the position of the first character of a string within another string.</td>
</tr>
<tr>
<td>%SCAN (argument, n, &lt;charlist &lt;,modifiers&gt;&gt;)</td>
<td>Searches the argument and returns the nth word.</td>
</tr>
<tr>
<td>%SUBSTR (argument, position, &lt;,length&gt;)</td>
<td>Produces a substring of character string (argument) by extracting the specified number of characters (length) beginning at the specified starting position.</td>
</tr>
<tr>
<td>%UPCASE (character-string text-expression)</td>
<td>Converts lowercase characters in the argument to uppercase.</td>
</tr>
</tbody>
</table>

SAS Functions with Macro Variables

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%EVAL (arithmetic or logical expression)</td>
<td>Evaluates arithmetic and logical expressions using integer arithmetic.</td>
</tr>
<tr>
<td>%SYSEVALF (expression &lt;,conversion-type&gt;)</td>
<td>Evaluates arithmetic and logical expressions using floating-point arithmetic.</td>
</tr>
<tr>
<td>%SYSFUNC (function (argument-1 &lt;...argument-n&gt;) &lt;,format&gt;)</td>
<td>Executes SAS functions or user-written functions in the macro facility.</td>
</tr>
</tbody>
</table>

Troubleshooting Macro Variable References

Enables you to write your own messages to the SAS log.
%PUT text;

Deletes the specified variables from the macro global symbol table.
%SYMDEL macro-variable-1 <...macro-variable-n><option>
### Masking Special Characters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%STR(argument)</code></td>
<td>Hides the usual meaning of a semicolon(;) so it appears as constant text.</td>
</tr>
<tr>
<td><code>%NRSTR(character-string)</code></td>
<td>Hides the usual meaning of an ampersand(&amp;) or a percent sign(%) so they appear as constant text.</td>
</tr>
<tr>
<td><code>%SUPERQ(argument)</code></td>
<td>Masks all special characters and mnemonic operators at macro execution but prevents further resolution of the value.</td>
</tr>
<tr>
<td>`%BQUOTE(character-string</td>
<td>text-expression)`</td>
</tr>
<tr>
<td>`%QUPCASE(character-string</td>
<td>text-expression)`</td>
</tr>
<tr>
<td><code>%SUBSTR(argument, position &lt;,length&gt;)</code></td>
<td>Produces a substring of a character string.</td>
</tr>
<tr>
<td><code>%QSCAN(argument, n &lt;,charlist&lt;,modifiers&gt;&gt;&gt;)</code></td>
<td>Searches for a word and masks special characters and mnemonic operators.</td>
</tr>
<tr>
<td><code>%QSYSFUNC(function( arguments) &lt;,format&gt;)</code></td>
<td>Executes functions and masks special characters and mnemonic operators.</td>
</tr>
</tbody>
</table>

### Conditional Processing

```sas
%IF expression %THEN text;
<%ELSE text>;  
%IF expression %THEN %DO;
   text and/or macro language statements;
%END;
%ELSE %DO;
   text and/or macro language statements;
%END;
%DO index-variable=start %TO stop <,BY increment>;
   text
%END;
```

### Options

- `OPTIONS MCOMPILENOTE= NONE | NOAUTOCALL | ALL;`
- `OPTIONS MPRINT | NOMPRINT;`
- `OPTIONS MLOGIC | NOMLOGIC;`
- `OPTIONS MAUTOSOURCE | NOAUTOSOURCE;`

### Creating Macros in SQL

```sql
PROC SQL NOPRINT;
SELECT column1<,column2,...> INTO :macro-variable-1<,:macro-variable-2,...> <TRIMMED> FROM table-1 | view-1 <WHERE expression> <other clauses>;
QUIT;
```

### DATA Step Interface

```sas
CALL SYMPUTX(macro-variable-name, value <,symbol-table>);
PUT(source,format.);
```

### Advanced Macro Techniques

- `%INCLUDE file-specification </SOURCE2>;
- DOSUBL(text-string);

### Default Autocall Library

- `%LOWCASE(argument)`
- `%QLOWCASE(argument)`
- `%LEFT(argument)`
- `%TRIM(argument)`
- `%CMPRES(argument)`
- `%DATATYP(argument)`

---

Copyright © 2019 SAS Institute Inc. Cary, NC, USA. All rights reserved.
Arrays

Defining an array

```
ARRAY array-name<
  <$> <length> <array-elements>
  <_TEMPORARY_> <(initial-values)>;
```

Referencing an array

```
array-name[element-number];
```

The number of elements must be enclosed in either parentheses ( ), braces { }, or brackets [ ].

Hash Objects

A hash object is an in-memory table that contains key and data components.

### Hash Object and Iterator Process

Declaring hash object or hash iterator object:

```
DECLARE hash object-name
  (<argument_tag-1:value-1, ...>);

DECLARE hiter object-name
  ('hash-object-name');
```

Defining a hash object:

```
object-name.ADD( );
object-name.DEFINEKEY('key-1' <, ...'key-n'>);
object-name.DEFINEDATA('data-1' <, ...'data-n'>);
object-name.DEFINEONE( );
object-name.OUTPUT( );
```

Using a hash object:

```
object-name.FIND( );
```

Retrieving a hash object with a hash iterator object:

```
object-name.FIRST( );
object-name.LAST( );
object-name.NEXT( );
object-name.PREV( );
```

### Unknown Number of Elements

Use an asterisk (*) within your brackets when defining an array.

Use the DIM function to return the number of elements in an array.

```
DIM(array-name);
```

### Two-Dimensional Arrays

```
ARRAY array-name
  [number-of-rows,number-of-columns];

array samplearray[3,2];
```

The example above creates an array named SampleArray which has 3 rows and 2 columns.
Picture Formats

**PROC FORMAT;**
**PICTURE format-name <(format-options)>**
<value-range-set-1='template-value' (template-options)>
<value-range-set-n='template-value' (template-options)>;
**RUN;**

Creating Custom Date, Time, Datetime Formats

**DATATYPE=DATE | TIME | DATETIME**
enables the use of directives in the picture as a template to format date, time, or datetime values.

**DEFAULT=length**
specifies the default length of the picture.

Creating Custom Numeric Formats

**MULTIPLIER=n**
specifies a number to multiply the value by.

**PREFIX='prefix'**
specifies a character prefix to place in front of the formatted value.

**ROUND**
rounds the value to the nearest integer before formatting.

Options

Perl Regular Expressions

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/.../</td>
<td>Starting and ending delimiter</td>
</tr>
<tr>
<td>(...)</td>
<td>Enables grouping</td>
</tr>
<tr>
<td></td>
<td>Denotes the OR situation</td>
</tr>
<tr>
<td>\d</td>
<td>Matches a digit (0-9)</td>
</tr>
<tr>
<td>\D</td>
<td>Matches a non-digit such as letter</td>
</tr>
<tr>
<td>\s</td>
<td>Matches a whitespace character</td>
</tr>
<tr>
<td>\w</td>
<td>Matches a group of characters</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character</td>
</tr>
<tr>
<td>[...]</td>
<td>Matches a character in brackets</td>
</tr>
<tr>
<td>[^...]</td>
<td>Matches a character not in brackets</td>
</tr>
<tr>
<td>^</td>
<td>Matches the beginning of the string</td>
</tr>
<tr>
<td>$</td>
<td>Matches the end of the string</td>
</tr>
<tr>
<td>\b</td>
<td>Matches a word boundary</td>
</tr>
<tr>
<td>\B</td>
<td>Matches a non-word boundary</td>
</tr>
<tr>
<td>*</td>
<td>Matches the preceding character 0 or more times</td>
</tr>
<tr>
<td>+</td>
<td>Matches the preceding character 1 or more times</td>
</tr>
<tr>
<td>?</td>
<td>Matches the preceding character 0 or 1 times</td>
</tr>
<tr>
<td>(n)</td>
<td>Matches exactly n times</td>
</tr>
<tr>
<td>\</td>
<td>Overrides the next metacharacter such as a ( or ?)</td>
</tr>
</tbody>
</table>

Creating Functions

**PROC FCMP OUTLIB=libref.table.package;**
**FUNCTION function-name(arguments) <length>;**

<table>
<thead>
<tr>
<th>Programming statements . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN(expression);</td>
</tr>
<tr>
<td>ENDSUB;</td>
</tr>
<tr>
<td>QUIT;</td>
</tr>
</tbody>
</table>

Using Custom Functions

**OPTIONS CMPLIB=libref.table | (libref.table-1...libref.table-n)**

Advanced Functions

**LAG<n>(column);**

**COUNT(string, substring<,modifiers>);**

**COUNTC(string, character-list<,modifiers>);**

**COUNTW(string, <,delimiters><,modifiers>);**

**FIND(string, substring<,modifiers><,start-position>);**

**FINDC(string, character-list<,modifiers> <,start-position>);**

**FINDW(string, word<,delimiters><,modifiers><,start-position>);**

**PRXCHANGE Function**

PRXCHANGE function performs a substitution for a pattern match

**PRXCHANGE (Perl-regular-expression, times, source)**